EPFL

Thèse n° 7890

Multi-Objective Management of Multiprocessor Systems: From Heuristics to Reinforcement Learning

Présentée le 30 octobre 2020

à la Faculté des sciences et techniques de l'ingénieur
Laboratoire des systèmes embarqués
Programme doctoral en génie électrique

pour l'obtention du grade de Docteur ès Sciences

par

Arman IRANFAR

Acceptée sur proposition du jury

Prof. P. Frossard, président du jury
Prof. D. Atienza Alonso, directeur de thèse
Prof. M. Van Der Schaar, rapporteuse
Prof. M. Zapater, rapporteuse
Prof. A. Burg, rapporteur

École polytechnique fédérale de Lausanne

2020

I was gratified to be able to answer promptly,
and I did.
I said I didn't know.
— Mark Twain

To my compassionate parents;
To my supportive brother;
And to my wonderful wife.

Acknowledgments

Pursuing my PhD at Embedded Systems Laboratory (ESL) under supervision of Prof. David Atienza was a great opportunity. I wish to express my sincere appreciation to him, who has convincingly guided and encouraged me to be professional. Without his persistent support, this PhD dissertation would not have been realized. I would like to take this opportunity to also thank my thesis jury members, Prof. Andreas Burg, Prof. Mihaela Van Der Shcaar, Prof. Pascal Frossard, and Prof. Marina Zapater, for taking their valuable time out of their busy schedule to review this manuscript and providing me with their insightful comments.

Besides, there have been several other people who have helped me a lot during my PhD. First of all, I would like to thank Dr. Marina Zapater who helped me a lot in editing my publications and always supported me. Also, I would like to thank Prof. Katzalin Olcoz, Prof. Jose Flich, Dr. Federico Terraneo, and Prof. Samuel Xavier-de-Souza with whom I had joint collaborations for several publications.

Additionally, I would like to seize this opportunity to acknowledge all my colleagues at ESL, with whom I interacted on a daily basis, making memorable moments, from time-to-time coffee breaks and birthday parties, to all other exciting ESL outdoors activities. Especially, I would like to thank Homeira for organizing lab events and my trips to different conferences in many places in Europe, Asia, and South America; Rodolphe and Mikael for all their beyond-the-imagination supports and efforts in all IT-related issues; Ruben, Adriana, Alexandre, Tomas, Miguel, Pablo, Leila, our post-doctoral fellows, with whom I worked indirectly during my PhD; Soumya, Anthongy, Marina, and Renato for the amazing moments we shared in our office; Dionisije, Halima, Ignacio, Alireza, Marco, Saleh, Una, Niloofar, Gregoir, Arteem, Elizabeta, Yasir, Loris, Lara, Benoit, Szabolcs, Wellington, Damián, Luis, Darong, and André, my colleagues and friends at ESL. I would like to express my especial gratitude to Dr. Amir Aminifar and Ali Pahlevan, for being such great friends and all their insightful comments that led me enhance my research quality.

During these years of living in Switzerland, there were many great Iranian friends who made life with all its ups-and-downs sweet and enjoyable for me. Hereby, I would like to thank them all: Ashkan, Mahsa, Mohammad, Shakiba, Omid, Nastaran, Shayan, and Hamed. Also, I sincerely thank my friends living abroad who always supported me when the road became tough: Amir, Arash, Paniz, Sadegh, and Soheil.

The last but here the most, I would like to express my heartfelt gratitude to my entire family, and particularly, my parents, brother, father-in-law, mother-in-law, and my beloved wife, Shadi.

Acknowledgements

Lausanne, September 28, 2020 Arman Iranfar

Abstract

Since the maximum operating frequency of processors started to saturate due to intolerable power dissipation, multiprocessor systems, such as Multiprocessor Systems-on-Chip (MPSoCs) and multi-core servers, have been playing a key role to meet the performance demand of many applications. The presence of multiple processors poses additional challenges in management of these systems with respect to well-known objectives and constraints including power, performance, temperature, and lifetime reliability. In particular, finding an optimal scheme able to utilize all available design- and run-time techniques, such as thread allocation and migration, consolidation, Dynamic Voltage and Frequency Scaling (DVFS), dynamic power management (DPM), and dynamic thermal management (DTM), is either impractical or infeasible within a reasonable time. Therefore, heuristics are vastly used to obtain the near-optimal or sub-optimal solutions.

Conventional multi-objective management of multiprocessor systems mostly focuses on hot spots as the main factor of lifetime reliability. Nonetheless, for modern multiprocessor systems and workloads, thermal stress, defined as any rapid change of temperature in time or space, has become the dominant factor in determining the Mean Time-To-Failure (MTTF). Together with the advances in multiprocessor systems, cooling technologies have been also progressively improving. As a result, existing DTM policies should adapt to these emerging challenges and technologies to further improve the lifetime reliability.

To address the multi-objective management of multiprocessor systems, I first propose a holistic, yet fast thermal stress-aware heuristic approach. The results demonstrate that the lifetime reliability can increase by up to 47% with only 4% performance degradation if thermal stress mechanisms are properly considered when applying traditional DTM techniques, such as DVFS, processor consolidation, and thread migration. Then, I show how emerging cooling technologies, such as two-phase liquid-cooling thermosyphon, necessitates adapting conventional heuristics to gain the greatest possible advantage from all its potential. The results indicate that if emerging cooling technologies are accompanied by proper DTM techniques, thermal hot spots and thermal stress can further decrease by up to $10^{\circ}C$ and 45%, respectively, with 45% less cooling power consumption.

Although heuristics are still among the most popular methodologies for multi-objective management of multiprocessor systems, a successful heuristic usually requires in-depth knowledge of the application, workload, and the underlying processing system. Moreover, accurate workload prediction and throughput estimation are keys in efficient proactive power and performance management of multiprocessor platforms. Nevertheless, mastering the com-

putational demand of certain application domains, where the workload varies rapidly due to the input data (e.g., video coding), is very challenging. To address this problem, I propose a machine learning-based framework for workload prediction and throughput estimation using hardware events available on modern multiprocessor systems. The proposed machine learning framework leverages supervised and unsupervised learning to cluster, classify, and predict workload on multiprocessor systems, achieving 3.4x higher throughput with 15% less power consumption for High Efficiency Video Coding (HEVC), as a test-case application.
Traditionally, performance, power, energy, and temperature have been considered as the main design objectives and constraints of multiprocessor systems. However, with the advent of new applications domains, such as real-time video streaming and Deep Learning, Quality-of-Service (QoS), defined in various terms, is added as a new important design objective or constraint. Such additional objectives and constraints and additional runtime design parameters exacerbate the already-existing challenges in management of multiprocessor systems. Besides, these new application models expose several internal design parameters that provide a trade-off between computational complexity, performance, power, and QoS. These parameters, either specified at design time or adaptively tuned at run time, exponentially increase the design space of multi-objective management of multiprocessor systems. Moreover, a comprehensive runtime management of multiprocessor systems should take advantage of all available runtime parameters. One of these important parameters that has been used hardly by conventional DTM and DPM approaches is adaptive cooling, such as adaptive fan speed control. The reason mainly lies in the fact that including such additional parameters eventually results in an extremely large design space which requires novel solutions. In fact, for such problems, heuristics can hardly provide a holistic solution. Also, providing the supervised learning algorithms with sufficient training data representative of the whole design space is infeasible. Reinforcement Learning (RL), on the contrary, is able to dynamically learn from very complicated environments, by adjusting actions based on a continuous feedback from the environment. Consequently, in this thesis, I address runtime management and design space search of large and dynamic environments through RL. In particular, I first propose an RL-based framework to enable proactive fan speed control along with DVFS and workload allocation, providing up to 40% cooling power savings without any thermal constraint violations. Second, I address multimedia workload allocation of HEVC encoder on heterogeneous Systems-on-Chip (SoCs) through RL, achieving 20% higher compared to state of the arts. Then, I propose an RL-based approach that enables joint optimization of application- and system-level parameters, improving power consumption, performance, and average temperature of multiprocessor systems by 13%, 15%, and 10%, respectively, while improving the video quality and video compression of HEVC encoders, as a use-case application, by up to 1.19 dB and 24%. Although the well-studied Single-Agent RL (SARL) is very efficient in multi-objective management of multiprocessor systems, given an extremely huge design space, a single learning agent may not be able to sufficiently explore different runtime parameters within a reasonable amount of time. Therefore, I propose a Multi-Agent Reinforcement Learning (MARL) approach for multi-objective runtime management of multiprocessor systems. In this approach, the design space is split into smaller independent sub-spaces such that agents accurately

explore the assigned design sub-space, and cooperatively maximize the design objectives. I use HEVC encoder as a test-case application, where MARL can enhance QoS violations by 5x, while speeding up the learning phase by 15x. Finally, there are new applications whose parameters, defined at design time, can significantly affect the QoS and performance. Today, Deep Convolutional Neural Networks (DCNNs) are very popular with hundreds or thousands of application-level design parameters, known as hyperparameters. Thus, I propose a novel MARL-based approach for efficient design space search of DCNNs at design time, such that their internal parameters can be set to the values which ultimately maximize the QoS and performance.
To recap, in this thesis, I reveal several already-existing and emerging challenges in multi-objective management of multiprocessor systems, and address them through novel solutions, from heuristics to RL, depending on the complexity of the problem.

Keywords: *Multi-core server, Multiprocessor System-on-Chip (MPSoC), multi-objective optimization, dynamic power management (DPM), dynamic thermal management (DTM), liquid cooling, thermal stress, thermal cycling, machine learning (ML), reinforcement learning (RL), Q-learning (QL), Quality-of-Service (QoS), Convolutional Neural Network (CNN), High Efficiency Video Coding (HEVC)*

Zusammenfassung

Seitdem die maximale Operationsfrequenz der Prozessoren in den Sättigungsbereich eintrat, die mit den untragbaren Leistungsverlusten begründet wird, gewannen die Multiprozessoren vor allem die Multiprozessoren auf Platine (MP-CoCs) und Multi-Core-Server an Bedeutung, um die Leistungsanforderungen der neuen Anwendungssystemen zu erfüllen. Der Einsatz von Multiprozessoren führt aber zu neuen Herausförderungen in Systemmanagement, Leistungsreglung, Wärmedämmung und System-Zuverlässigkeit. Besonders ist der Einsatz eines optimalen Schemas zur Einstellung aller Entwurf- und Laufzeitmethoden wie Thread-Zuordnung, Konsolidierung, dynamische Spannung- und Frequenzeingruppierung (DVFS), dynamische Leistungsreglung (DPM) und dynamische Wärmereglung (DTM) entweder unpraktisch oder zeitbezüglich undurchführbar. Daher die Heuristiken werden eingesetzt, um eine nahe-optimale Lösung zu erreichen. Die konventionalen Multi-objektiv Management der Multiprozessorsystemen fokussieren sich am meistens auf die so genannten Hot-Spots, die als Hauptfaktoren der Lebenszeit-Resilienz gelten. Allerdings ist für moderne Multiprozessoren die thermische Belastung (wird als jegliche rapide Änderung der Temperatur in Zeit oder Raum definiert) zu einem dominanten Faktor zur Bestimmung der durchschnittliche Fehlschlag-Zeit (MTTF) geworden. Dabei mit den Entwicklungen in Multiprozessoren sind die Abkühlungstechnologien entwickelt. Daher wird für die zurzeit geltenden DTM-Vorschriften eine Anpassung zur Erfüllung dieser neuen Herausforderungen zur Aneignung dieser neuen Technologien vorausgesetzt, um die Lebenszeit-Zuverlässigkeit der Systeme zu verbessern. Um das Multi-Objektive-Management von Multiprozessorsystemen anzugehen, wird in dieser Arbeit zunächst einen ganzheitlichen und dennoch schnellen heuristischen Ansatz vorgestellt, der sich der thermischen Belastung bewusst ist. Die Ergebnisse zeigen, dass die Lebenszeit-Zuverlässigkeit kann um bis zu 47% erhöht werden, obwohl die Leistung nur um 4% beeinträchtigt wird, wenn die Stressmechanismen bei der Anwendung traditioneller DTM-Techniken, DVFS, Prozessorkonsolidierung und Thread-Migration eingesetzt werden. Dann wird es gezeigt, wie sich Abkühlung Technologien wie das Zweiphasen-Thermosiphon zur Flüssigkeitskühlung eine Anpassung herkömmlicher Heuristiken erfordern, um den größtmöglichen Vorteil aus all ihrem Potenzial zu ziehen. Die Ergebnisse weisen zusätzlich darauf hin, dass, wenn aufkommende Kühltechnologien von geeigneten DTM-Techniken begleitet werden, können die thermischen Brennpunkte und die thermische Belastung weiter um bis zu 10 ± C bzw. 45% abnehmen und das mit 45% weniger Kühlstromverbrauch. Obwohl Heuristiken immer noch zu den beliebtesten Methoden für das Management mit mehreren Zielen gehören, erfordert bei Multiprozessorsystemen eine erfolgreiche Heuristik norma-

lerweise fundierte Kenntnisse der Anwendung, der Arbeitslast und des zugrunde liegenden Verarbeitungssystems. Darüber hinaus sind eine genaue Workload-Vorhersage und Durchsatzschätzung die wichtigen Schlüssel für ein effizientes und proaktives Leistungsmanagement von Multiprozessor-Plattformen. Die Beherrschung des Rechenaufwands bestimmter Anwendungsbereiche, in denen die Arbeitslast aufgrund von Änderungen der Eingabedaten (z. B. Videocodierung) schnell variiert, ist sehr herausfordernd. Um dieses Problem anzugehen, wird in dieser Arbeit ein auf maschinellem Lernen basierendes Framework für die Workload-Vorhersage und Durchsatzschätzung mit Hilfe von Hardwareereignissen, die auf modernen Multiprozessorsystemen verfügbar sind, vorgeschlagen. Das vorgeschlagene Framework für maschinelles Lernen nutzt überwachtes und unbeaufsichtigtes Lernen, um die Arbeitslast auf Multiprozessorsystemen zu gruppieren, zu klassifizieren und vorherzusagen. Dabei wird ein 3,4-fach höherer Durchsatz bei 15% weniger Stromverbrauch für HEVC (High Efficiency Video Coding) als Testfallanwendung erzielt. Traditionell werden Leistung, Energie und Temperatur als Hauptentwurfsziele und Einschränkungen von Multiprozessorsystemen betrachtet. Mit dem Aufkommen neuer Anwendungsbereiche wie Echtzeit-Video-Streaming und Deep Learning wird die in verschiedenen Begriffen definierte Quality-of-Service (QoS) als neues wichtiges Entwurfsziel oder neue Einschränkung hinzugefügt. Solche zusätzlichen Ziele und Einschränkungen sowie zusätzliche Laufzeitentwurfsparameter verschärfen die bereits bestehenden Herausforderungen beim Management von Multiprozessorsystemen. Außerdem stellen diese neuen Anwendungsmodelle mehrere interne Entwurfsparameter bereit, die einen Kompromiss zwischen Rechenkomplexität, Leistung und QoS bieten. Diese Parameter, die entweder zur Entwurfszeit spezifiziert oder adaptiv zur Laufzeit angepasst werden, vergrößern der Entwurfsraum für die Verwaltung von Multiprozessorsystemen mit mehreren Objektiven in einer exponentiellen Weise. Darüber hinaus soll ein umfassendes Laufzeitmanagement von Multiprozessorsystemen alle verfügbaren Laufzeitparameter verwenden. Einer dieser wichtigen Parameter, der von herkömmlichen DTM- und DPM-Ansätzen kaum verwendet wurde, ist die adaptive Kühlung, wie beispielsweise die adaptive Steuerung der Lüftergeschwindigkeit. Der Grund liegt hauptsächlich in der Tatsache, dass das Einbeziehen solcher zusätzlichen Parameter letztendlich zu einem extrem großen Entwurfsraum führt, der neuartige Lösungen erfordert. Tatsächlich können Heuristiken für solche Probleme kaum eine ganzheitliche Lösung bieten. Es ist auch nicht möglich, den überwachten Lernalgorithmen ausreichende Trainingsdaten zur Verfügung zu stellen, die für den gesamten Entwurfsraum repräsentativ sind. Im Gegensatz dazu kann Reinforcement Learning (RL) dynamisch aus sehr komplizierten Umgebungen lernen, indem Aktionen basierend auf einem kontinuierlichen Feedback aus der Umgebung angepasst werden. Infolgedessen beschäftige ich mich in dieser Arbeit mit dem Laufzeitmanagement und der Suche nach Entwurfsräumen in großen und dynamischen Umgebungen über RL. Insbesondere schlage ich zunächst ein RL-basiertes Framework vor, das eine proaktive Steuerung der Lüftergeschwindigkeit zusammen mit der DVFS- und Workload-Zuweisung ermöglicht und bis zu 40% Kühlleistung einspart, ohne dass thermische Einschränkungen verletzt werden. Zweitens befasse ich mich mit der multimedialen Arbeitslastverteilung von HEVC-Encodern auf heterogenen Systems-on-Chip (SoCs) über RL, die im Vergleich zum Stand der Technik 20% höher sind. Dann schlage ich einen RL-basierten Ansatz

vor, der eine gemeinsame Optimierung der Parameter auf Anwendungs- und Systemebene ermöglicht, den Stromverbrauch, die Leistung und die Durchschnittstemperatur von Multiprozessorsystemen um 13%, 15% bzw. 10% verbessert und gleichzeitig die Videoqualität und Videokomprimierung von HEVC-Encodern als Anwendungsfallanwendung um bis zu 1,19 dB und 24% verbessert. Obwohl der gut untersuchte Single-Agent RL (SARL) bei der Verwaltung von Multiprozessorsystemen mit mehreren Zielen sehr effizient ist, kann ein einzelner Lernagent angesichts eines extrem großen Entwurfsraums möglicherweise nicht in der Lage sein, verschiedene Laufzeitparameter innerhalb einer angemessenen Menge von Zeit ausreichend zu untersuchen. Daher schlage ich einen MARL-Ansatz (Multi-Agent Reinforcement Learning) für ein multi-zielbasierendes Laufzeitmanagement von Multiprozessorsystemen vor. Bei diesem Ansatz wird der Entwurfsraum in kleinere unabhängige Unterräume aufgeteilt, sodass die Agenten den zugewiesenen Entwurfsunterraum genauer untersuchen und die Entwurfsziele kooperativ maximieren. Ich verwende den HEVC-Encoder als Testfallanwendung bei der MARL, der QoS-Verstöße um das 5-fache verbessern und die Lernphase um das 15-fache beschleunigen kann. Schließlich gibt es neue Anwendungen, deren zur Entwurfszeit definierte Parameter die QoS und Leistung erheblich beeinflussen können. Heutzutage sind Tief Konvolution Nerven Netzwerke (DCNNs) bei hunderten oder tausenden von Entwurfsparametern auf Anwendungsebene, sogenannten Hyperparametern, sehr beliebt. Daher schlage ich einen neuartigen MARL-basierten Ansatz für die effiziente Suche nach Entwurfsräumen von DCNNs zur Entwurfszeit vor, sodass deren interne Parameter auf die Werte eingestellt werden können, die letztendlich die QoS und Leistung maximieren. Zusammenfassend möchte ich in dieser Arbeit einige bereits bestehende und aufkommende Herausforderungen beim Multiobjektivmanagement von Multiprozessorsystemen hervorheben und diese je nach Komplexität des Problems durch neuartige Lösungen von Heuristik bis RL angehen.

Contents

List of Figures

List of Tables

Acronyms

ARIMA Auto-Regressive Integrated Moving Average

ASR Adaptive Search Range

AVC Advanced Video Coding

AVFS Adaptive Voltage and Frequency Scaling

BraTS Brain Tumor Segmentation

CU Coding Unit

DCLC Direct Contact Liquid Cooling

DELC Dual Enclosure-Liquid Cooling

DPM Dynamic Power Managemnt

DTM Dynamic Thermal Management

DVFS Dynamic Voltage and Frequency Scaling

DVS Dynamic Voltage Scaling

EM Electromigration

FPS Frame per Second

GOP Group of Pictures

HEVC High Efficiency Video Coding

HPC High-Performance Computing

IPC Instruction Per Cycle

IPMI Intelligent Platform Management Interface

IPS Instruction Per Second

ISIC International Skin Imaging Collaboration

JCT-VC Joint Collaborative Team on Video Coding

LCU Largest Coding Unit

LLC Last Level Cache

MARL Multi-Agent Reinforcement Learning

ML Machine Learning

MPSoC Multiprocessor System-on-Chip

MTTF Mean Time to Failiure

OS Operating System

PCA Principal Component Analysis

PQL Pareto Q-Learning

PUE Power Usage Effectiveness

PWM Pulse-Width Modulation

QL Q-Learning

QoS Quality of Service

QP Quantization Parameter

RAPL Running Average Power Limit

RL Reinforcement Learning

RMSE Root Mean Square Error

SARL Single-Agent Reinforcement Learning

SGD Stochastic Gradient Descent

SoC System-on-Chip

STG Spatial Thermal Gradient

TC Thermal Cycling

TDDB Time Dependent Dielectric Breakdown

TTC Thermal Test Chip

TTG Temporal Thermal Gradient

WPP Wavefront Parallel Processing

1 Introduction

Since 1990s to early 2000s, clock frequency accounted for more than 80% of the increase in the uniprocessor performance at each technology node [1]. Although performance was the primary design objective since the beginning of the microprocessors era in 1970s, other key aspects, such as energy consumption, power dissipation, and temperature, started to become the main concerns of microprocessor designers. This was mainly due to the fact that with each new generation, microprocessors faced decreased physical dimension and increased number of transistors; thus, scaling the operating frequency ran into the physical barriers [2].

One breakthrough that helped microprocessors to keep up with the ever-increasing demand of performance without a consequent increase in the clock frequency was integrating multiple cores in a processor. With the introduction of multi-core processors, performance, thermal, power, and energy concerns could be temporarily relieved through higher performance by the same operating frequency of microprocessors and more distributed power dissipation across the processor. Nonetheless, as the demand for higher performance kept increasing, Multiprocessor Systems-on-Chip (MPSoCs) were introduced, where multiple processors, each containing one or more cores are integrated on the same SoC (System-on-Chip). The multiprocessor systems term, however, encompasses a variety of different systems, such as MPSoCs, multi-core and multiprocessor servers, GPUs (Graphical Processing Unit) with multiple CUDA and Tensor cores, ZYNQ SoCs, etc. Throughput the remainder of this thesis, by multiprocessor systems, I refer to any system composed of multiple cores or processors, either integrated on the same chip or communicating through any Inter-Processor Communication (IPC) protocols.

1.1 Multi-Objective System-Level Management of Multiprocessor Systems

Although multiprocessor systems have been introduced to address the high performance demand while mitigating power dissipation and thermal issues, none of these goals can be achieved in the lack of proper system-level management. Compared to the uniprocessor

systems, multi-objective system-level management of modern multiprocessor systems is more challenging and, if poorly managed, not only the same performance, power and thermal concerns of the uniprocessors persist, but also unprecedented issues and challenges with respect to these design objectives and constraints can arise.

In particular, while workload allocation and scheduling is vital for taking the most advantage of multiprocessor systems, finding the optimal workload mapping and scheduling is known to be NP-hard. Moreover, today's multiprocessors come with a wide range of operating frequency. In order to maximize performance while satisfying the design constraints, such as power consumption and thermal hot spots, the operating frequency should be adjusted according to the performance requirement and these design constraints. Applying this technique, known as Dynamic Voltage and Frequency Scaling (DVFS) [3], becomes even more challenging for modern systems where per-core frequency adjustment is provided for more fine-grained power and performance control. Finally, as a consequence of these new features, power, performance, and lifetime reliability management of multiprocessor systems is more burdensome than any time.

In the following subsections, I provide an overview of different design objectives and constraints considered in the scope of my thesis.

1.1.1 Power and Performance Management

According to Amdahl's Law in the multi-core era [4], the maximum speedup (S) achievable through parallelization can be theoretically obtained as follows:

$$S = \frac{1}{1 - f + \frac{f}{n}}, \tag{1.1}$$

where f is the fraction of computation that can be parallelized ($0 \leq f \leq 1$), and n represents the number of processors. The speedup per watt of such a multiprocessor system is computed by [5]:

$$\frac{S}{W} = \frac{1}{1 + (n-1)k(1-f)}, \tag{1.2}$$

where $0 \leq k \leq 1$ is the fraction of power the processor consumes in idle mode. Despite their incompleteness and simplicity for today's modern multiprocessor systems, equations (1.1) and (1.2) indicate that while multiprocessor systems can increase the performance through parallel processing, the average power consumption would grow dramatically, such that $\frac{S}{W} \to 0$. As a result, in spite of the fact that increasing the performance has been traditionally the primary objective for processor designers, nowadays, a joint optimization of power consumption and performance is vital.

Moreover, power consumption has become one of the first-order design constraints in many

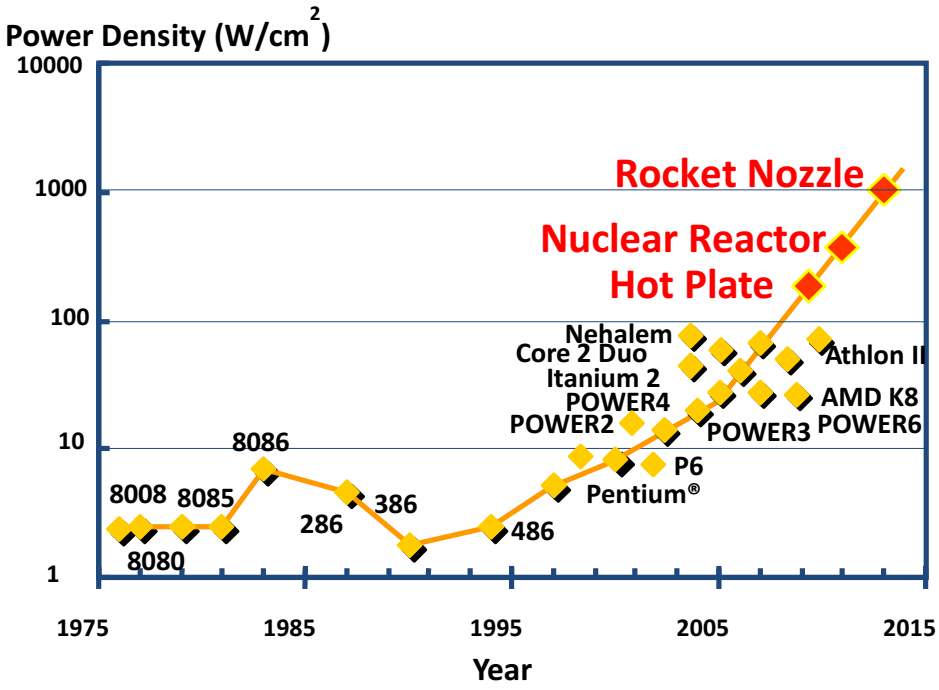


Figure 1.1 – Power density over time [10]

multiprocessor systems. For embedded, high performance, and large scale systems, power consumption is the main concern due to battery lifetime, heat dissipation, and electricity billing costs, respectively [6]. More specifically, if the power consumption increases by a factor of 2, battery lifetime of embedded systems may degrade by a factor of 3 [7]. In addition, as a rule of thumb, every 1 W of power in high performance computing systems leads to 1 W of power spent on cooling. Finally, the electricity consumption of data centers across the U.S is estimated to reach 139 billion kWh in 2020 [8, 9] costing an annual bill of around $20 billion.

1.1.2 Lifetime Reliability and Thermal Management

The increase in the speed of multiprocessor systems along with the shrinkage of the feature size at each technology node has led to higher power density and, thus, higher peak and average temperature across the chips. Figure 1.1 shows how over the recent years, the power density of uniprocessors and multiprocessor systems have increased. On one hand, temperature increases super linearly as the power density increases. On the other hand, leakage power and temperature are interrelated through a positive feedback. Therefore, proper thermal management is essential to have a functional multiprocessor system.

Another important issue arising with high power density and, thus, temperature, is the IC (Integrated Circuit) failure rate [12], where a 10-15°C difference in temperature may halve the life span. Moreover, the availability of more resources in comparison with uniprocessors leads to more non-uniformity of the temperature profile on multiprocessor systems. Spatial thermal

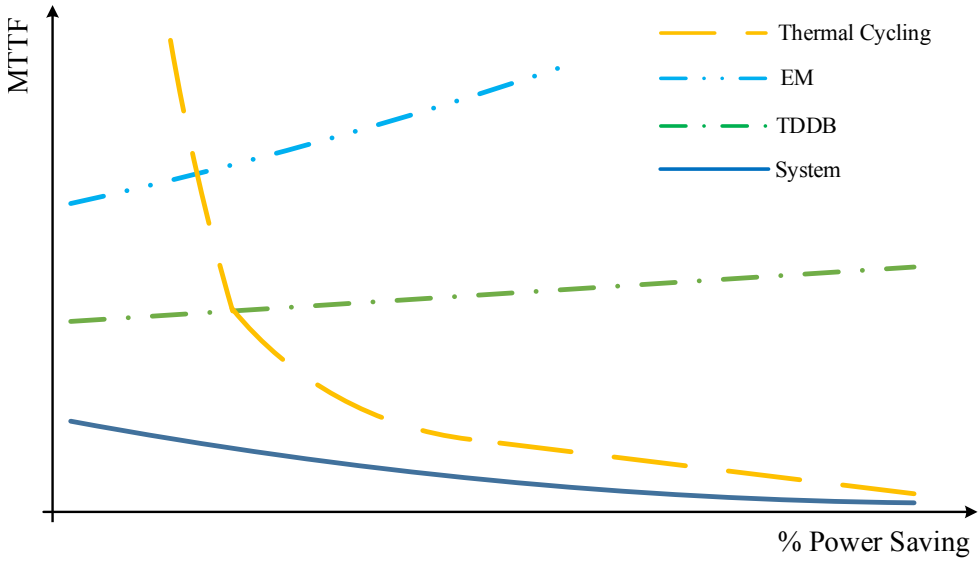


Figure 1.2 – MTTF vs. percentage of power saving provided by common DPM techniques [11]

gradients across the chip deteriorate the system reliability and degrade its performance. Also, the variety of the workloads, which could be processed at the same time, may cause large temporal thermal variations at a single point on the chip. More importantly, thermal cycling, as a new dominant failure mechanism, leads to degrading the performance and reliability of modern processors.

Several power management techniques including Dynamic Voltage Scaling [13], task allocation and scheduling [14], and throttling [15, 16] help reducing the chip average temperature by lowering the average power consumption. Although these approaches reduce hard failures corresponding to Electromigration (EM) and Time-Dependent-Dielectric-Breakdown (TDDB), they do not take into account thermal stress as an important factor in lifetime reliability. The study performed by Coskun et al. [17] reveals that the increase in the amount of power saving, which is usually followed by peak and average temperature reduction, improves the mean time-to-failure (MTTF) by reducing the EM and TDDB occurrences, whereas it causes the overall MTTF of the system to fall down, since the MTTF related to thermal cycling decreases faster. Particularly, common DPM (Dynamic Power management) and DTM (dynamic thermal Management) approaches are effective in decreasing the total power consumption and peak/average temperature. Nonetheless, such techniques cause rising and falling of the temperature not only more frequently but also with higher amplitudes, hence, reducing reliability of the system. For instance, for metallic structures, when a thermal cycle amplitude increases from $10°C$ to $20°C$, the lifetime reliability may decrease by up to 16 times [11]. Figure 1.2 shows how conventional DPM techniques can adversely result in MTTF degradation. This suggests that, if not performed properly, power management techniques may deteriorate the lifetime of multiprocessor systems by aggravating the thermal stress.

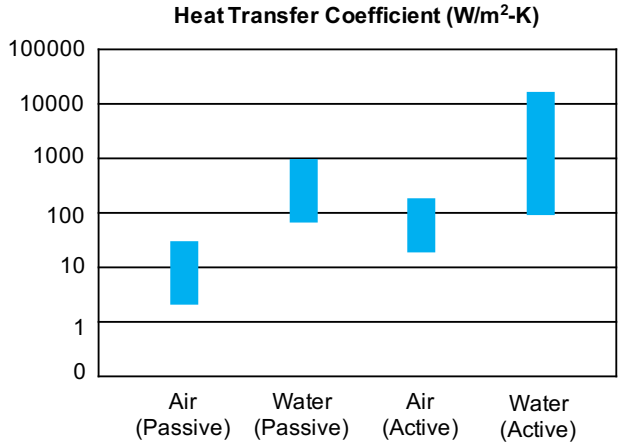


Figure 1.3 – Heat transfer coefficient of different cooling systems [21]

1.1.3 Cooling

Another important drawback with high power density is the cooling cost. As a rule of thumb, every 1 W of power in IC leads to 1 W of power spent on cooling, if designed for the worst-case scenario [18]. Moreover, IC and system packaging/cooling cost increases super-linearly with power consumption [19]. This increasing cooling cost, consequently, urges processor designers and researchers to seek thermal solutions to keep the temperature lower than a threshold. Nevertheless, cooling is inevitable in today's multiprocessor systems. Each system, however, requires a particular cooling method. In particular, while heat spreaders may be the only option for mobile embedded systems, for high-performance and large-scale computing systems more advanced cooling technologies have to be used. Each of these cooling methods has pros and cons. Inexpensive passive cooling through heat sinks is not effective for high performance computing systems. Fans, as the most common active air cooling system, contribute to higher electricity bill. Liquid cooling provides higher efficiency in heat removal than air cooling, however, it also requires additional power supply. Alternatively, emerging two-phase cooling systems, such as thermosyphons [20], can take advantage of gravity and provide efficient cooling with no extra power consumption in theory, while operating inaudibly.

Figure 1.3 shows the heat transfer coefficient achievable through different cooling systems, with respect to the convection type [21]. In order to achieve efficient heat removal, the thermal profile of multiprocessor systems along with the capabilities of the cooling system need to be studied. Indeed, particular potential of a cooling system may significantly influence the strategies taken in traditional power and thermal management of multiprocessor systems. Moreover, with the advent of new cooling technologies, such as thermosyphon, it is necessary to adapt DTM approaches to their new features.

1.2 Emerging Application Models and Quality of Service Requirements

Traditionally, Quality-of-Service (QoS) is known as the desirable performance, in terms of either throughput or latency, for many applications, such as web search and data analytics. However, for many emerging applications, such as bio-medical imaging and video streaming, QoS cannot anymore simply account for performance. Moreover, many new trending application, e.g., multimedia applications, expose several internal parameters through which system-level objectives and constraints can be considerably affected. As a consequence, power, performance, and thermal management of multiprocessor systems can no longer solely rely on conventional design parameters, such as operating frequency and workload allocation, but rather designers need to take into account application-level parameters to jointly optimize along with these well-studied system-level parameters. Obviously, with application-level parameters included in the design space and adding emerging QoS requirements to the design objectives and constraints, multi-objective management of multiprocessor systems becomes more challenging.

1.2.1 Video Coding

In 2015, real-time entertainment already accounted for more than 74% of downstream network traffic in North America, with streaming services, including Netflix, YouTube, and Amazon Video, accounting for 57% of the global share [22]. According to The Global Internet Phenomena Report by Sandvine [23], by the end of 2019, 65% of the worldwide mobile traffic was video streaming. The network traffic share of video streaming is expected to experience a new spike as several other video providers, such as Disney+, HBO Max, and Apple, will soon launch their services. Such a shift of services and application was, indeed, predicted by the ITRS in 2009 [24]. For video streaming services and applications performance can keep its common definition, i.e., framerate known as frame per second (FPS), represents the throughput. For real-time streaming, meeting the required FPS (e.g., 24, 30, or 60 Hz, depending on the application) is one of the first-order design objectives, and accounts for the main challenge. However, QoS and Quality-of-Experience (QoE) of the user will also depend on the video quality, usually measured as the Peak Signal-to-Noise ratio, and bitrate. The latter strongly depends on the user end, nonetheless, video providers can play an important role by detecting the user's bandwidth and adjusting the bitrate accordingly. The recent video coding standards such as High Efficiency Video Coding (HEVC), a.k.a x265, and Advanced Video Coding (AVC), a.k.a x264, provide several internal application-level parameters, adjustable at run-time. These parameters can be used to provide a trade-off between the computational complexity and the output video in terms of performance (particularly, throughput), quality and bitrate. On the other hand, the computational complexity can considerably affect the power consumption and thermal profile of the video providers' servers.

The increasing popularity of streaming applications and services implies a shift in the nature of

workloads with which multiprocessor systems need to deal. Multimedia workloads are heavily dependent on the input data (e.g., raw videos). Even though, similar to many applications, multimedia is composed of several sequential or parallel process phases, the amount of CPU workload of each depends on features of the input video frames. The main features include size, amount of motion and complexity of the texture, as well as the temporal and spatial locality between frames and within a frame. On one hand, since the video content can be anything, it is very difficult to accurately predict the next coming frames (hence, upcoming workload) to proactively apply the most appropriate multi-objective management policy. On the other hand, content (input) variation and, thus, workload variation, can be very fast. Therefore, it is essential to have a very low-overhead multi-objective management policy such that it can adapt to the high variant workload in a short amount of time.

1.2.2 Deep Learning

A report by Amazon [25] demonstrates that the budget of businesses and industries of all sizes on machine learning has increased by 540% in 2019 compared to 2018. The key role of Deep Learning in many application domains, such as computer vision, medical imaging, image processing, machine translation, language processing, etc., is known as the main reason of such considerable growth.

Convolutional Neural Networks (CNNs), as a very important member of Deep Learning family, have emerged to be a very successful alternative to traditional Artificial Intelligence (AI) and image processing algorithms due to its higher accuracy in image classification, semantic segmentation, face recognition, etc. CNNs, however, are CPU/GPU-intensive workloads. Moreover, since CNNs require very large datasets for training, they can be memory-intensive. As a result, many users tend to use cloud-based environments, such as Amazon Web Service (AWS) and Google Cloud, for training and inference, while others may still rely on private infrastructures. In both cases, the application accuracy and execution time are of considerable importance. CNNs architecture contains several so-called hyperparameters that can be set at design time. These hyperparameters can significantly affect accuracy, execution time (at both training and inference), model size, and power consumption. In fact, for single CNN architecture, different hyperparameters can alter the training time from hours to days, model size from a few Megabytes to Gigabytes, and inference time from milliseconds to tens of seconds.

1.3 Multi-Objective Management of Multiprocessor Systems in Literature

There are two main strategies for system-level management of multiprocessor systems: static and dynamic, applied at design- and run-time, respectively. Traditionally, static strategies were used to solve the problem of task allocation on multiprocessor systems [26–32] using

computationally-intensive search algorithms in order to find the (near-)optimal solution. However, these time-consuming static strategies do not take into account the dynamic behavior of applications. Moreover, static approaches lack flexibility because the entire set of applications and workloads should be known and studied well in advance. Even worse, if the applications or the underlying multiprocessor system change slightly, a major re-computation is necessary to obtain a new static strategy compatible with these changes. In particular, Jia et al. [27] propose a design space exploration framework and use the derived Pareto-optimal design point for energy-efficient task allocation on MPSoCs. Their work considers the amount and number of available processors, memory, and network at design time. However, their proposed mapping is only applicable to the explored platform. Furthermore, Integer Linear Programming (ILP) is among the most well-known methods to derive optimal task allocation and scheduling [33, 34, 32]. These works, nevertheless, are mainly limited to finding the optimal number of processors and do not consider other system-level parameters.

In contrast to static approaches, in a dynamic strategy [35–50] the running application and the system are monitored continuously or within intervals such that it can well adapt to the workload changes and performance requirements. Dynamic approaches, nonetheless, have to employ sufficiently fast algorithms. Dynamic methods can also be applied together with static methods, where static design space exploration are first used to derive one or more optimal policies and, then, dynamic methods modify the optimal solution to adapt to the workload changes [40]. Such a combination of dynamic and static approaches, however, do not suit cases where workload can change dramatically at run-time.

A wide range of different approaches, including optimal solutions, control theory, heuristics, evolutionary and genetic algorithms, and machine learning, have been used for dynamic multi-objective management and optimization of multiprocessor systems. Despite the fact that optimal multi-objective system-level management is NP-hard [18], several works leverage optimal solutions while trying to reduce the run-time overhead. In particular, Hanumaiah et al. [51] solve a convex optimization problem for performance optimization of thermally-constrained multi-core processors. A convex optimization problem for thermal stress-aware power management is solved by Kamal et al. [52]. Murali et al. [53] form a convex optimization problem for thermal-aware DVFS. These works, however, evaluate their work on simulation- or emulation-based frameworks, thus, the overhead of solving the convex problem remains in question.

For many applications, solving an optimization problem within a sufficiently short amount of time is infeasible, especially for multi-objective scenarios. Therefore, many existing works leverage simple and fast heuristics to cope with the multi-objective management of multiprocessor systems [54, 46]. In this context, Neshatpour et al. [55] use heuristics to apply DVFS and application migration for enhancing power, performance, and temperature profile of multiprocessor systems. Moghaddam and Ababei [56] combine DVFS and thread migration for dynamic lifetime reliability management. Xie and Hung [57] address thermal-aware task allocation and scheduling on MPSoCs through a heuristic algorithm. Rahmani et al. [58, 59]

propose multi-objective DPM approaches through heuristics and PID controllers. Del Sozzo et al. [60] use binary search to find the best system-level parameters for workload-aware power optimization. Nonetheless, none of these heuristic approaches can be considered as a comprehensive solutions for multi-objective management of multiprocessor systems, since they either do not consider lifetime reliability, or do not address new dominant failure mechanisms.

Control theory has been also deployed in multi-objective management of multiprocessor systems. In particular, Wang et al. [61] address thermal-constrained power management of MPSoCs, while Skadron et al. [62] propose a DTM approach. However, in these works, the controller has a corrective behavior. Thus, proactive power and thermal management cannot be guaranteed.

Evolutionary and Genetic Algorithms (GA) have been popular approaches for task allocation on multiprocessor systems because they can provide a fast search of the design space [63]. Moreover, several works have proposed GA-based solutions for multi-objective run-time management of multiprocessor systems. In particular, Pillai et al. [64] use GA for energy and performance optimization. Miao et al. [65] propose a GA-based approach for multi-objective optimization of chip multiprocessors. Kumar and Palani [66] apply optimal DVFS by using GA to optimize power-performance product of multiprocessor systems. GAs, however, are based on random heuristics and they can be easily trapped in local optimum.

Recently, Machine Learning and in particular, Reinforcement Learning (RL) has been successfully used for system-level management of multiprocessor systems. In this context, Gupta et al. [67] use deep Q-Learning (QL) to maximize the power efficiency of heterogeneous multiprocessors. However, authors evaluate their work through well-known benchmarks and do not address challenges of new application models and QoS requirements. Also, the run-time design parameters are limited to three levels of operating frequency and determining the number of active processors. Moghaddam [68] addresses dynamic energy optimization of chip multiprocessor systems under performance constraint through Deep Neural Networks (DNN). For this purpose, their work uses DVFS, discarding other run-time parameters. Similarly, ul Islam et al. [69] and Zhang et al. [70] propose an RL-based framework for energy optimization of embedded systems through DVFS, while neglecting other system-level design parameters. In a similar way, Otoom et al. [71] take advantage of QL to independently compute the optimal voltage and frequency of the cores in multi-core platforms for power and performance optimization.

1.4 Thesis Contributions

In this section, I briefly explain the contributions of my thesis in order of appearance.

1.4.1 Heuristic Multi-Objective Management of Multiprocessor Systems

The first two contributions of my thesis tackle with adapting heuristics, as the most common method dealing with multi-objective run-time management of multiprocessor systems.

1.4.1.1 Lifetime Reliability Optimization

Modern multiprocessor systems are equipped with several dynamic power, performance, and thermal management parameters. In particular, Intel is leveraging DVFS, P-states, and C-states, and memory throttling at OS-level to optimize the performance considering thermal and power constraints [72, 73]. Along with these new advances, researcher have been addressing multi-objective run-time management of multiprocessor systems through well-known techniques such as DVFS, thread migration, and processor consolidation. These techniques, if applied in their traditional way, may be sufficient in reducing thermal hot spots through decreased power consumption. However, today, lifetime reliability of the multiprocessor systems are threatened by other factors known as thermal stress rather than the well-studied phenomena such as TDDB and EM. Therefore, I propose a heuristic thermal stress-aware power and performance management approach to optimize the lifetime reliability of multiprocessor systems. The main contributions can be briefly stated as follows:

- I study and include three different thermal stress mechanisms, namely, Spatial Thermal Gradients, Temporal Thermal Gradients, and Thermal Cycling, in run-time thermal-aware power and performance management of multiprocessor systems,
- I propose a fast heuristic algorithm for near-optimal per-core DVFS,
- I propose a low-overhead heuristic for thread migration and processor consolidation,
- For comparison, I reformulate an existing DVFS convex optimization for considering the spatial thermal gradient,
- I validate the scalability of my proposed heuristics when the number of cores increases,
- I validate the efficiency of the proposed methods when confronting large workload variations.

1.4.1.2 Adapting to New Cooling Technologies

Conventional heuristic approaches that deal with power and thermal management of multiprocessor systems usually discard the undeniable role of cooling systems. Moreover, recently, there have been advances in cooling devices and technologies. In particular, one of the most promising two-phase liquid cooling devices, thermosyphon, has been validated successfully on multi-core servers. However, existing thermal-aware run-time management approaches are not well prepared to take the most advantage of this new cooling prototype. Moreover, the

design of a multiprocessor system should adapt to the features of the applied cooling system. Therefore, thermosyphon, as the newest cooling technology should be first studied. Hence, first, I assess the workload- and platform-aware design of a thermosyphon for power-hungry multiprocessor Systems. Afterwards, I propose a thermal-aware workload mapping strategy specifically tailored to the designed thermosyphon to further reduce thermal hot spots and spatial gradients while meeting the workload requirements.

The main contributions can be summarizes as follows:

- I investigate and show the potential and limitations of a two-phase thermosyphon for power-hungry processors,
- I propose a workload- and platform-aware design and adaptation of a thermosyphon, together with a thread mapping heuristic policy for multi-core enterprise servers when a two-phase thermosyphon is used,
- I evaluate the proposed design and mapping strategy under different workloads requirements with respect to the lifetime reliability compared to state-of-the-art heuristic thermal management approaches,
- I evaluate and compare the cooling power consumed through the proposed design and mapping heuristics compared to the state-of-the-arts.

1.4.2 Machine Learning Framework for Multi-Objective Management

The recent shift of applications and services to multimedia poses a new challenge in multi-objective management of multiprocessor systems. The workload of multimedia application, such as High efficiency Video Coding (HEVC) as the latest coding standard, is both memory- and CPU-intensive and it changes dramatically depending on the input data. Existing power and performance management of multiprocessor systems mostly use heuristics based on the modeled application graph. However, these approaches do not suit applications whose workload can change abruptly. On the contrary, machine learning (ML) provides a strong tool when it comes to pattern recognition and prediction. Modern multiprocessor systems can record hundreds of hardware event through a group of registers named performance counters. These events can be used as a very rich dataset for ML-based approaches. Therefore, I first propose an ML-based framework for workload prediction and throughput estimation of time-varying applications in multiprocessor systems. Then, I show how such an accurate workload prediction and performance estimation can be used in run-time power and performance management. In particular, the main contributions are as follows:

- I propose a low-overhead workload prediction approach based on the hardware events, applicable to any state-of-the-art many-core platform,

- I develop an ML-based framework which provides the future throughput estimate under different number of threads and operating frequencies,
- I study and use High Efficiency Video Coding (HEVC) as the test-case application,
- I compare the proposed ML-based framework with a neural network-based approach in terms of prediction and estimation accuracy,
- For comparison with respect to power and performance optimization, I propose a heuristic for application-specific workload allocation and DVFS,
- I show how the accurate workload prediction and throughput estimation per system configuration can save power in a multi-core server while satisfying the performance requirements.

1.4.3 Reinforcement Learning for Runtime Management and Design Space Search

Traditionally, power dissipation, energy consumption, performance, and thermal profile are known as the main objectives and constraints of run-time management of multiprocessor systems. However, the recent change in the type of trending applications and services imply that QoS and Quality of Experience cannot be discarded as one of the first-order design objectives or constraints. Two very famous example of such application are multimedia, in particular the recent computationally complex coding standards, and Deep Learning (DL). In the former, quality of the videos along with the video compression compose the QoS and QoE. In the latter, accuracy of the given task to the Convolutional Neural Networks and Recurrent Neural Networks as two main instances of DL form the QoS. Therefore, traditional approaches of multi-objective management of multiprocessor systems need to adapt to these emerging objectives.

Besides, these application have several internal parameters that can be either set at design time or tuned at run-time. As a result, researchers and system designers have to cope with two large sets of parameters: application-level, and system-level parameters. Ignoring either of these two sets of parameters will ultimately lead to sub-optimal run-time management of multiprocessor systems.

Conventional approaches such as heuristics, GAs, and classical ML are not able to deal with such a dynamic and large design space. Alternatively, Reinforcement Learning (RL) can effectively learn from the interactions with the environment (application, multiprocessor system, and design space) to come up with a fast design space exploration or run-time management policy. Thus, through the last five contributions of my thesis, I show how RL can be used for multi-objective design space exploration and run-time management of multiprocessor systems.

1.4.3.1 Efficient Proactive Cooling

Conventional run-time power and thermal management approaches usually discard the undeniable role of cooling systems. While for passive cooling a static design may be adequate, in case of active cooling its run-time parameters should be considered along with other system-level parameters. Fans, as one of the most common active cooling devices, can be controlled dynamically at run-time to improve power, performance, and thermal management of multiprocessor systems. However, given that fan speed (voltage) can be set at multiple levels, design space of the run-time management grows exponentially. As a consequence, achieving a desirable multi-objective run-rime management scheme through the existing approaches becomes more challenging and tedious.

Thus, in this part of my thesis, I use RL to proactively adjust fan speed along with DVFS and adaptive workload allocation with respect to the temperature, power consumption, and required performance. The main contributions can be summarized as follows:

- I propose an RL-based DTM policy that leverages fan speed, DVFS and dynamic core assignment to maximize the performance and to minimize fan power subject to thermal constraints,
- I validate the proposed RL-based DTM policy on a thermal test chip,
- I validate the efficiency of the proposed methods when confronting large and random workload variations,
- I compare the proposed RL-based proactive fan speed control policy with an RL-based approach with a fixed fan speed and a heuristic adaptive fan control policy.

1.4.3.2 Workload Allocation on Heterogeneous Multiprocessor Systems

Task allocation plays a significant role in power and performance management of multiprocessor systems. Traditionally, designers would distribute different tasks of a given application across processors based on the application graph or prior knowledge of the application features through heuristics, GAs, and classic ML. Such an approach, however, cannot provide satisfactory results considering new application domains, in which the workload (hence, the execution time) varies significantly with respect to the input data. A famous example of such applications is multimedia. Moreover, the recent generation of multimedia application, HEVC, comes with several internal parameters tunable at run-time to provide a trade-off between the algorithm complexity, QoS, power consumption, and performance.

Therefore, I propose an RL-based run-time management approach for power and performance optimization of heterogeneous multiprocessor systems. The proposed approach is able to automatically allocate different video inputs (in terms of resolution and motion) to general-purpose cores and hardware accelerators, while setting the frequency of both. The main

contributions of this part of my thesis are as follows:

- I address DVFS and workload allocation on heterogeneous multiprocessor systems through an RL-based approach for power and performance management,
- As a case study, I consider HEVC encoder in a multi-user environment,
- The proposed RL-based approach can learn the best DVFS and allocation settings with respect to the input video features, HEVC internal parameters, and available resources,
- I compare the RL-based approach with a state-of-the-art heuristic load balancing method.

1.4.3.3 Multi-Objective System- and Application-Level Runtime Management

Multi-objective management of multiprocessor systems cannot be thoroughly addressed through existing approaches when considering new application models and QoS requirements. Hence, I propose an RL-based approach that enables joint adaptation of application- and system-level parameters. As the case study, I provide a detailed study of HEVC encoder, analyzing various internal parameters and their impact on conventional design objectives and constraints, as well as the QoS. In particular, in the proposed approach, I consider power consumption, performance and temperature of a multi-core server, in addition to the video quality (PSNR) and bitrate as the QoS of HEVC encoders. My main contributions are as follows:

- I provide a detailed study of HEVC encoder, as the most recent coding standard, with respect to various application-level parameters, input features, and their impacts on the QoS, power consumption, performance, and temperature of multiprocessor systems,
- I propose an RL-based approach to utilize both application- and system-level parameters in order to increase the QoS while satisfying power and thermal constraints without performance degradation. My approach is able to observe the obtained performance and encoding efficiency, as well as the server power consumption and temperature, and learn how to dynamically set both encoding parameters and operating frequency at run-time for any arbitrary videos and contents.
- I develop a resolution-aware video allocation strategy to reduce thermal hot spots while maintaining the desired performance,
- I compare the proposed RL-based approach to an optimal solution,
- For evaluation, I consider several possible scenarios on the servers of video providers. Specifically, I show how the RL-based approach can benefit from resource availability to improve encoding efficiency.

1.4.3.4 Multi-Agent Reinforcement Learning for Multi-Objective Runtime Management

When a multiprocessor system hosts multiple different applications, or multiple instances of the same application at the same time, it is necessary to simultaneously satisfy the QoS of each, while meeting other design objectives and constraints of the system, such as power consumption. Obviously, this a more challenging task than managing only one application instance, even if it is distributed to multiple available resources. Moreover, for the new applications, such as HEVC, each instance has its own internal parameters to be set dynamically at run-time. Since these parameters affect the amount of resources required to process an instance of the application, an optimal policy to set the parameters would be the one that takes into account other applications need and requirements. Such a scenario results in an extremely large and dynamic design space.

Although for large and dynamic design spaces RL is very promising for multi-objective run-time management, when the design space becomes extremely large, traditional RL, which uses only one learning agent, may not be able to achieve the optimal policy. The reason lies in the fact that the learning agent must explore the design space by itself, thus, it may either end up with a sub-optimal policy, or it takes too long to converge to an optimal one. On the contrary, if the design space could be split to several smaller sub-spaces, then multiple learning agents can independently explore each smaller design space and share their experience such that an optimal policy is achieved. This type of RL is called cooperative multi-agent RL (MARL) [74].

Therefore, I present a MARL-based run-time management strategy for QoS-aware power and performance optimization of multiprocessor systems in a multi-application environment. As a test-case application, I assume a multi-user environment for real-time HEVC encoding, where a random number of streams with different features (frame size, length, and content) need to be processed at the same time. To address this problem, I propose to decompose the design space into simpler independent sub-spaces. Then, each agent is in charge of independently exploring a particular design sub-space to attain sufficient knowledge about the environment at faster pace. Finally, each agent exploits its own knowledge and experience jointly with the others' to behave optimally along with all agents in the environment.

My main contributions to the state of the art are as follows:

- I show how adaptation of application- and system-level parameters can be decomposed to provide fast and efficient run-time management of multi-user real-time HEVC encoding,
- I develop a MARL-based power and performance management of multiprocessor systems that optimizes video quality subject to video compression as the QoS metrics,
- For comparison, I develop a heuristic to jointly set application- and system-level parameters,
- I show how the MARL-based approach outperforms the proposed heuristic and SARL-

based solution in serving multiple simultaneous users while achieving real-time encoding.

1.4.3.5 Hyperparameter Optimization of Convolutional Neural Networks

Nowadays, Deep Learning is one of the most trending applications. CNNs, as an important member of DL family, play a key role in many application domains, such as computer vision, medical imaging, and image processing. CNNs are one of the most memory- and CPU/GPU-intensive applications. The amount of resources required by CNNs for training and inference, however, highly depends on several internal parameters. These so-called hyperparameters are set at design time and affect the output accuracy as well as the training and inference time. Designing Deep CNNs (DCNNs) with more than tens of layers, and each with several hyperparameters, is a tedious task due to the extremely large design space. Nevertheless, optimizing DCNNs is inevitable as the DCNN architecture and its hyperparameters strongly affect both application and the underlying system.

Therefore, as the last contribution of this thesis, I propose a MARL-based approach that automates hyperparameter optimization of DCNNs, under arbitrary objectives and constraints at both application and system levels. The main contributions are as follows:

- I propose a novel multi-agent RL-based approach that eliminates the human effort in hyperparameter optimization of DCNNs,
- My new definition of RL elements enables splitting the design space to smaller subspaces, providing faster, yet accurate search,
- The proposed approach is data-driven, i.e., given a CNN architecture, it tunes the hyperparameters according to the input data to the CNN,
- While my proposed approach can take into account multiple arbitrary constraints and objectives imposed by the application or the processing platform, I consider model accuracy, training and inference time, and model size,
- I show that my proposed solution is capable of fine-tuning any arbitrary DCNN, by applying it to different architectures and well-known datasets.

1.5 Thesis Organization

The remainder of this thesis is organized as follows:

Chapter 2 presents two heuristics for multi-objective management of multiprocessor systems, with the focus on lifetime reliability.

Chapter 3 addresses multi-objective management of multiprocessor systems through machine learning for emerging application where the workload can change abruptly.

Chapter 4 introduces reinforcement learning as a very powerful method to address multi-objective management of multiprocessor systems, especially for new applications whose parameters need to be set at design time, or adjusted at run-time along with other system-level parameters.

Chapter 5 concludes the thesis, summarizes its main contributions, and provides new ideas for future research in the same direction.

2 Heuristic Thermal-Aware Runtime Management of Multiprocessor Systems

2.1 Introduction

Multiprocessor systems play a key role in modern computational platforms due to their higher performance [75]. However, the increase in the speed of these systems along with the decrease in the chip dimensions has led to higher power consumption, more power density, and frequent hot spots. Therefore, proper multi-objective system-level management of these systems is crucial. Compared to the uniprocessor systems, system-level management of modern multiprocessor systems is more challenging. First of all, workload mapping is an issue only in multiprocessing. Second, modern processors make real-time thread migration possible. Finally, these processors usually come with a wide range of operating frequency that can be independently set for each core at run-time. All these system-level parameters have to be considered in multi-objective run-time management of multiprocessor systems.

Static strategies at design time cannot meet power, performance, and thermal requirements in presence of workload variations. In contrast, Dynamic Thermal Management (DTM) [76] and Dynamic Power Management (DPM) [3] have been used in different forms, such as convex optimization, Genetic Algorithms, Machine Learning (ML), and heuristics. Among these approaches, heuristics are the most popular thanks to their simplicity and low-overhead execution [54].

Lifetime reliability of multiprocessor systems is strongly influenced by DTM and DPM. Although conventional DTM and DPM approaches can improve the lifetime reliability by reducing the thermal hot spots across the chip, they can also contribute to degradation of lifetime reliability. In fact, DTM and DPM approaches use DVFS, thread migration, and throttling techniques. All these techniques can cause large temporal thermal variations and, in particular, large and frequent thermal cycles [75]. Moreover, the nature of multiprocessor systems in addition to the DPM and DTM approaches can lead to more non-uniform thermal profile across the chip. This non-uniformity, often called spatial thermal gradient, also deteriorates lifetime reliability [77]. Thermal stress influences system reliability and, in particular, determines the MTTF at moderate temperatures [78]. Thus, reducing the thermal hot spots is not

solely enough to achieve comprehensive thermal management for multiprocessor systems. In fact, today, the dominant factor in lifetime reliability of multiprocessor systems is known to be thermal stress, either in time or space, rather than hot spots [77]. Thus, a comprehensive multi-objective management should consider thermal stress as one of the first-order design objectives or constraints. In my thesis, any rapid temperature change, in either time or space, is regarded as a thermal stress mechanism.

In addition to proper DTM and DPM, proper cooling plays an important role in thermal management and lifetime reliability of modern multiprocessor systems, by alleviating thermal hot spots and providing more uniform thermal profile across the chip. Although passive cooling, mainly realized through heat spreaders and heat sinks, is low-cost and simple to design, they are not efficient for high-performance and power-hungry processors. On the contrary, active cooling is more efficient in heat removal. However, active cooling adds to the total power consumption and, thus, to the electricity bill. In particular, nowadays, data centers consume more than 2% of the global energy consumption [79] with cooling energy accounting for about 33% of the share [80–82]. Power Usage Effectiveness (PUE), defined as the ratio of total facility energy to IT equipment energy, indicates how efficient a data center is. While PUE improved from 2.5 in 2007 to 1.65 in 2013 [83], not major improvements have been recently reported. This is mainly due to inefficiency of air-cooling systems at traditional cooling facilities. A recent study by Cisco shows that through a set of modifications in all its facilities, a PUE drop from 1.48 to 1.36 would save US$2 million/year [84]. One of these modifications is leveraging liquid-based cooling systems. Larger heat transfer coefficient of water of other refrigerants can significantly improve the amount of heat removal. One major drawback of most conventional liquid cooling systems is that they require pumping power which ultimately prevents the PUE to drop significantly. Two-phase liquid cooling, where phase change between vapor and liquid occurs can provide even higher heat transfer coefficient. Among all recent advances in cooling technologies, micro-scale two-phase liquid cooling designed and manufactured by Seuret et al. [20] achieves a PUE of 1.05. However, this two-phase cooling technology, called thermosyphon, needs to be studied and assessed at all its potential and limits. Once studied, it is vital that conventional DTM approaches adapt to this new cooling device so that the true improvement in thermal control and lifetime reliability can be observed.

In this chapter, first I propose a comprehensive heuristic for power, performance, and thermal management of multiprocessor systems, focusing on lifetime reliability optimization with respect to thermal stress mechanisms. Then, after assessing thermosyphon, I propose a heuristic for cooling-aware workload allocation to further improve lifetime reliability of multiprocessor systems.

2.2 Lifetime Reliability Mechanisms

Thermal profile directly affects the lifetime reliability and, thus, Mean Time-to-Failure (MTTF) of multiprocessor systems through different mechanisms. In what follows, I briefly overview these critical failure mechanisms.

2.2.1 Electromigration (EM)

Electromigration occurs in aluminum and copper interconnects due to the mass transport of conductor metal atoms in the interconnects [85]. The MTTF due to EM is computed as follows [86]:

$$MTTF_{EM} \propto (J - J_{crit})^{-n} exp(\frac{E_{aEM}}{KT}) \tag{2.1}$$

where J and J_{crit} are, respectively, current density and critical current density to initiate EM in interconnect, E_{aEM} is the activation energy for EM, K represents the Boltzmann's constant, T is the absolute temperature in Kelvin, and n is a constant that depends on the interconnect metal. Thus, according to Eq. (2.1), $MTTF_{EM}$ degrades at high temperatures.

2.2.2 Stress Migration (SM)

Stress migration occurs when the metal atom in interconnects migrates due to the mechanical stress caused by different thermal expansion rates of different materials in the device. The MTTF due to SM is modeled as follows [86]:

$$MTTF_{SM} \propto |T_0 - T|^{-n} exp(\frac{E_{aSM}}{KT}) \tag{2.2}$$

where T_0 is the stress-free temperature, T is the operating temperature, and E_{aSM} denotes the activation energy for SM. As indicated by Eq. (2.2), the higher the temperature is, the more decreases the lifetime reliability.

2.2.3 Time-Dependent Dielectric Breakdown (TDDB)

TDDB is also known as gate-oxide breakdown, and occurs when a conductive path appears in the dielectric. The MTTF due to TDDB is computed as follows [87]:

$$MTTF_{TDDB} \propto (\frac{1}{V})^{a-bT} exp(\frac{X + \frac{Y}{T} + ZT}{KT}) \tag{2.3}$$

where a, b, X, Y, and Z are fitting parameters [87], and V represents the operating voltage.

2.2.4 Temporal and Spatial Thermal Gradients

Temporal Thermal Gradient (TTG) is defined as the rate of temperature changes over time. For a given time, the rate of the temperature changes from one point to another indicates the Spatial Thermal Gradient (STG). Both STG and TTG pose a critical impact on the system lifetime reliability [11, 88]. However, when considering STG, power and thermal management techniques must be applied regarding current status of more than one core. In contrast, TTG is more affected by the operating frequency of a single core and its workload.

TTG and STG can be formulated in MTTF equations of TDDB and EM [89]. In particular, Lu et al. [89] model $MTTF_{EM}$ and $MTTF_{TDDB}$ by assuming that the stress is a function of time and space denoted by $sigma(x,t)$. Then, the time to failure can be obtained as follows:

$$t_{failure} = \frac{\sigma_{th}^{-1}}{E(\beta(T(t))} \tag{2.4}$$

where σ_{th} is the stress threshold with respect to time and space, and $E(\beta(T(t)))$ is the expected value of the following:

$$\beta(T(t)) = A\frac{exp(\frac{-Q}{kT(t)})}{kT(t)} \tag{2.5}$$

where A is a constant.

2.2.5 Thermal Cycling

Thermal cycling phenomenon is another important thermal stress mechanism. By definition, when the temperature rises up (drops down) and goes back to the initial value a thermal cycle occurs [90] and it can be counted by Dowining simple rainflow-counting algorithm [91]. The expansion coefficient mismatch between the layers results in thermomechanical stresses and contributes to several failure mechanisms such as dielectric/thin film cracking, fractured bond wire, solder fatigue, and cracked die [92]. Thermal cycling (TC) tends to reduce the whole system MTTF as the number of cycles or cycles amplitude increases. Large amplitudes are normally induced due to improper task scheduling on a single core. In contrast, number of thermal cycles increases especially by the run-time power management techniques which frequently turn cores on and off [11]. The number of cycles that can result in the failure due to the i^{th} thermal cycle is obtained from the modified Coffin-Manson equation as follows [90]:

$$N_{TC}(i) = A_{TC}(\delta T_i - T_{th})^{-b} exp(\frac{E_{aTC}}{KT_{max_i}}) \tag{2.6}$$

where δT_i is the maximum thermal amplitude change of the i^{th} thermal cycle, T_{th} is the threshold temperature at which inelastic deformation begins, b is the Coffin-Manson exponent constant, E_{aTC} is the activation energy, T_{max_i} is the maximum temperature in the i^{th} cycle, and A_{TC} is an empirically determined constant [90]. The MTTF related to thermal cycling can

be obtained by:

$$MTTF_{TC} = \frac{N_{TC}\Sigma_{i=0}^{m} t_i}{m} \tag{2.7}$$

where m is the total number of cycles. For metallic structures, when δT increases from $10^{\circ}C$ to $20^{\circ}C$, the lifetime reliability may decrease up to 16 times [11].

2.3 Trends in Cooling Methodologies and Technologies

Cooling methodologies can be divided in two main categories, active cooling and passive cooling. While the former consumes energy for cooling, the latter does not need any. Each of these two types of cooling methodologies can be in form of air-cooling, liquid-cooling, or two-phase cooling.

2.3.1 Air Cooling

Air cooling systems have been vastly employed at different levels from chips to data centers. At chip level, fans and heatsinks [93] are the most popular systems due to their design simplicity. At rack level, server placement in a rack plays an important role in heat flux [94], whereas at room level airflow configuration is known as the main design parameter [95]. Nonetheless, air cooling approaches are inefficient especially when encountering power-hungry servers [95]. The heat transfer coefficient of commercial passive air cooling systems (heatsinks) usually ranges from 5 to 50 (W/m2-K) [21], that is insufficient High-Performance Computing (HPC) servers. In addition, the heat transfer coefficient of active air cooling systems (fans) hardly surpasses 100 (W/m2-K) [21], while adding to the electricity billing cost, unable to improve the PUE. For instance, one of the most successful air-cooling systems have reported a PUE of as large as 1.65 [83].

2.3.2 Single-Phase Liquid Cooling

In single-phase liquid cooling, the coolant is a liquid and no phase change occurs during the cooling process. On the contrary to air cooling, liquid cooling systems benefit from high heat transfer coefficient and are capable of removing high heat flux. While air cooling systems are said to be able to support a 13 kW rack in common cases, this value goes up to approximately 200 kW for single-phase liquid cooling systems [96]. In fact, even passive liquid cooling systems generally outperform active air cooling systems due to their larger heat transfer coefficient [21]. As a result, successful prototypes of liquid-based cooling systems can be found both in literature and industry. Dual Enclosure-Liquid Cooling (DELC) unit in a typical Megawatt data center is expected to save $90-$240K per year [97]. In addition, the framework developed by Kadhim et al. [98] shows that Direct Contact Liquid Cooling (DCLC) systems can reduce the PUE down to 1.17. Finally, single-phase liquid-cooling has been successfully exploited by IBM [99].

2.3.3 Two-Phase Liquid Cooling

Two-phase cooling is another liquid cooling method in which liquid-vapor phase change occurs. The use of two-phase cooling, rather than single-phase liquid-based cooling systems, is strongly motivated due to their reduced mass flow-rates, lower pumping power, and smaller facility size [100, 101], while providing higher heat transfer coefficients and more uniform temperature profiles [95]. Two-phase cooling systems usually take advantage of liquid refrigerant with a boiling point of less than 50°C (versus 100°C for water), thus, heat from the servers more easily is absorbed by the liquid. In addition, while single-phase liquid cooling systems are claimed to support 200 kW per rack when attached to a chilled water system, this value is 250 kW for two-phase liquid cooling systems [102].

There are different prototypes of two-phase liquid cooling, In the following, first, I briefly overview immersion cooling and, then, detail the working principles of thermosyphon, as the most recent two-phase cooling prototype.

2.3.3.1 Immersion Cooling

One prototype of two-phase cooling is immersion cooling [103], where the whole server is immersed into a cold bath. Immersion cooling can decrease the PUE down to around 1.07 as it does not need fans, chillers, and Computer Room Air Handlers (CRAH). However, a major redesign of the server, especially for protecting the CPU, storage, memory, and optical devices is required. Moreover, the amount of liquid used is considerable, thus, the bath is heavy and costly.

2.3.3.2 Thermosyphon

Thermosyphon is another two-phase liquid cooling technology. A typical thermosyphon is composed of a closed loop with four main parts including downcomer, evaporator, riser, and condenser [104, 105]. The schematic and working principles of a thermosyphon is shown in Figure 2.1. The evaporator is located on top of a heat source (i.e., the heat spreader of a CPU), and initially contains a refrigerant in liquid state in its micro-channels. The heat from the CPU increases the evaporator temperature, and the refrigerant partially evaporates. The two-phase mixture composed of vapor and liquid ascends towards the condenser. The heat exchange between the coolant (i.e., cold water) and the hot two-phase mixture makes the two-phase mixture fall through the pipe thanks to the gravity. Since gravity plays a major role in the loop, the height of a thermosyphon is of significant importance, especially for a micro-scale design where the dimensions are constrained by the platform. This thermosyphon is gravity-driven, thus, it eliminate the pumping power.

Early thermosyphon prototypes [106], however, had a very large footprint area ($1m \times 1m$) making them impractical in commercial servers. Nonetheless, recent work by Lamaison et al. [105] and Seuret et al. [20] led to the design of micro-scale thermosyphons with a footprint

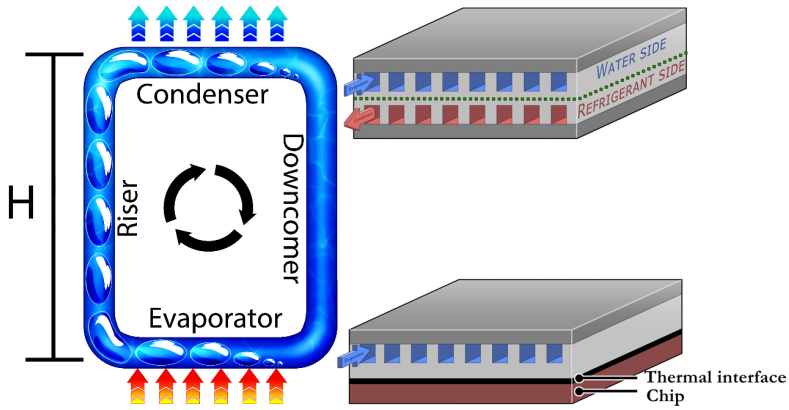


Figure 2.1 – Working principles of two-phase thermosyphon

Figure 2.2 – Thermosyphon prototype designed and manufactured by Seuret et al. [20]

size of only $5^{cm} \times 5^{cm}$ which can be placed directly on top of a CPU. Since a thermosyphon is placed on top of the processor package, in contrast to immersion cooling, it does not require any changes in design and fabrication of existing processors and servers. Moreover, as stated by Seuret et al. [20], this thermosyphon can further reduce the PUE to 1.05. Thus, such a design, if industrialized in a way that fully exploits all its potential, can save more money than any other cooling systems.

2.4 State-of-the-Art on Multi-Objective Thermal Management

In the following subsections, I first provide a literature review on power and thermal management. Then, I overview related works whose main objective is to improve the lifetime reliability with respect to thermal stress. Finally, I briefly review the recent works in cooling-aware thermal management.

2.4.1 Power and Thermal Management

Power and thermal management of multiprocessor systems is quite rich in literature. When power consumption started to become one of the most significant issues of these systems, researchers simply focused on power management policies through which peak temperature could also be controlled [107–111]. Although power management approaches could, to some extent, alleviate the thermal hot spots across the chip, increasing power density of multiprocessor systems made bare power management insufficient to deal with hot spots and led authors to propose thermal management policies at both design [18, 112, 113] and run time [114–119]. In particular, Chantem et al. [18] and Mutapcic et al. [112] propose optimal solutions for task scheduling and processor speed, respectively. Zhang and Chatha [113] maximize performance of a periodic application, and Mulas et al. [114] present a thermal balancing policy. Speedup of multi-core processors under thermal constraints is addressed by Hanumaiah and Vrudhula [115]. An OS-level technique for job scheduling is proposed by Zhou et al. [116]. Al Faruque et al. [117] address the runtime thermal aging of multi- and many-core architectures through software agents. The thermal impacts of the adjacent cores on the thermal profile is considered by Liu et al. [118]. Singla et al. [119] propose a dynamic thermal and power management using temperature prediction. All these works, however, fail to consider thermal stress as a new dominant factor in lifetime reliability of modern multiprocessor systems.

2.4.2 Thermal Stress-Aware Power Management

Considering the thermal stress, as an important factor in lifetime reliability of today's multiprocessor systems, in power and thermal management increases the complexity of the system-level management due to the contradictory behavior of peak temperature reduction techniques with thermal stress reduction methods. Even though there are several works considering thermal stress, they rarely provide a comprehensive solution to cope with all thermal stress mechanisms along with power constraints. For instance, although Choi et al. [120] assess the trade-offs between different schemes of handling temporal and spatial thermal gradients, they discard power management and thermal cycling. In addition, Yang et al. [121] propose a new task scheduling method for reducing the temporal temperature gradient. Nonetheless, their work does not consider thermal cycling and spatial gradient. In what follows, I review the main works which address the direct reduction of thermal cycling of multiprocessor systems.

Chantem et al. [75] describe an online task assignment and scheduling technique for maximizing the lifetime reliability of heterogeneous architectures. Ukhov et al. [122] propose a steady state temperature-aware task mapping and scheduling of a multi-core architecture by considering the thermal cycle effect. An online learning method, using a multivariate loss function which considers hot spots, thermal cycles, spatial gradients, and average load altogether for temperature management, is proposed by Coskun et al. [11]. Mercati et al. [123] use a hierarchical controller based on an aging sensor to improve the performance of multiprocessor systems. Unfortunately, none of these works consider power and/or performance

either as an objective or a design constraint.

Both static and dynamic methods are employed by Coskun et al. [88] to reduce the hot spots, spatial gradients and thermal cycles. In the static strategy, an integer linear programming scheduling method optimizes the power and temperature subject to the performance constraint. The optimization is based on balancing the thermal hot spots and suppressing the temperature variation without being concerned about the spatial gradient. In the dynamic method, a heuristic algorithm allocates ready jobs to the coolest processor with idle neighbors. Authors use Adaptive-Random [124] technique to consider the temperature histories of the cores as well as their current temperatures. Nonetheless, in this work, the proposed consolidation policy does not consider the adverse effect of thermal cycle. Machine learning is leveraged by [125] and [126] for thermal cycle reduction. Among them only the latter [126] considers all thermal stress mechanisms, yet the efficiency of the Q-learning-based approach has not been evaluated for rapid workload variations, which pose more temporal thermal gradients. Finally, Kamal et al. [52] propose a convex optimization solution and uses both processor consolidation and deconsolidation along with DVFS for reducing thermal stress. However, the formulated convex optimization problem for DVFS does not consider spatial thermal gradients and the run-time overhead is a major concern.

2.4.3 Cooling-Aware Thermal Management

A large number of DTM policies have been proposed in the literature [127]. However, many of them discard the cooling impact on DTM policies [18], or simply consider the power consumption of active cooling in a power model [128], rather than providing an adaptive control scheme to further improve the DTM efficiency.

Nevertheless, a few works consider the design parameters of active cooling in DTM. These works mainly focus on fan speed. In particular, an optimization framework is proposed by Dousti and Pedram [129] to find the optimal fan speed. TECfan [130] and the work of Zapater et al. [131] use a look-up table created offline for different fan speeds and workloads. Hanumaiah and Vrudhula [132] formulate the multiprocessor temperature as a convex function of fan speed to find the optimal task migration, DVFS, and fan speed. Core temperatures are estimated using neural networks for preemptive fan control by Acun et al. [133]. Chan et al. [134] propose a thermal management framework with respect to fan speed impact on performance and its energy cost.

Chan et al. [135] define a convex optimization problem to find the optimal fan speed. Finally, Kim et al. [136] use a fan speed controller to guarantee server operation stability without directly considering optimization of fan power.

For servers and data centers, cooling-aware workload allocation has been addressed by a few works. In this context, Liu et al. [137] schedule IT workloads with respect to cooling efficiency. Wang et al. [138] propose a workload scheduling algorithm for data centers based

on heat transfer coefficient of data center resources and thermal features of workload. A thermal-aware job placement strategy is proposed by Banerjee et al. [139] while considering the impact of cooling system on the power consumption of data centers. Job scheduling and node allocation for over-provisioned HPC systems is addressed by Cao et al. [140] through cooling awareness. Finally, Sabry et al. [141] address job scheduling in 3D stack architectures to maximize liquid cooling efficiency.

Despite such rich literature, micro-scale two-phase thermosyphon, which is one of the most promising next-generation cooling technologies, able to satisfy performance requirements and thermal constraints, has neither been studied nor considered in DTM so far.

2.5 Proposed Thermal Stress-Aware Power and Thermal Management Framework

In spite of the importance of thermal variation in performance and reliability of multiprocessor systems, most of power and thermal management techniques solely aim at power consumption/peak temperature reduction regardless of what adverse impacts their policies could have on the lifetime reliability of the target system. Several power management techniques including Dynamic Voltage Scaling (DVS) [13], DVFS [3] and task allocation and scheduling [14] help reducing the chip average temperature by lowering the average power consumption. Although these approaches reduce hard failures corresponding to TDDB and EM [11], they do not take into account thermal stress as a dominant factor in reliability of the modern multiprocessor systems [124]. A study by Coskun et al. [17] reveals that the increase in the amount of power savings, which is usually followed by peak/average temperature reduction, improves the MTTF by reducing the EM and TDDB occurrences, while causes the overall MTTF of the system to fall down, since the MTTF related to thermal cycling decreases faster. Particularly, DPM and DTM approaches usually utilize DVFS, thread migration, and clock gating [142] to decrease the total power consumption and peak/average temperature. However, such techniques cause temperature variations not only more frequently, but also with higher amplitudes, hence, reducing the system reliability. As a result, a comprehensive approach which considers thermal stress, power consumption, peak temperature, and performance objectives altogether, is vital.

In this section, I propose a framework for a comprehensive **The**rmal **S**tress-aware **Po**wer and **T**emperature (TheSPoT) management of multiprocessor systems. In this context, the main contribution of TheSPoT compared to the state-of-the-art DTM and DPM heuristics [143, 144] is that it can leverage thermal stress-aware heuristics to further improve the MTTF of the system. The overall view of TheSPoT is shown in Figure 2.3. TheSPoT is a thermal stress-aware power and thermal management framework which employs various controlling knobs, including DVFS, core consolidation/deconsolidation and thread migration. As a starting point, I consider VPTM which is a hierarchical dynamic power/thermal management framework for heterogeneous MPSoCs [145], and modify it in order to make it applicable for thermal

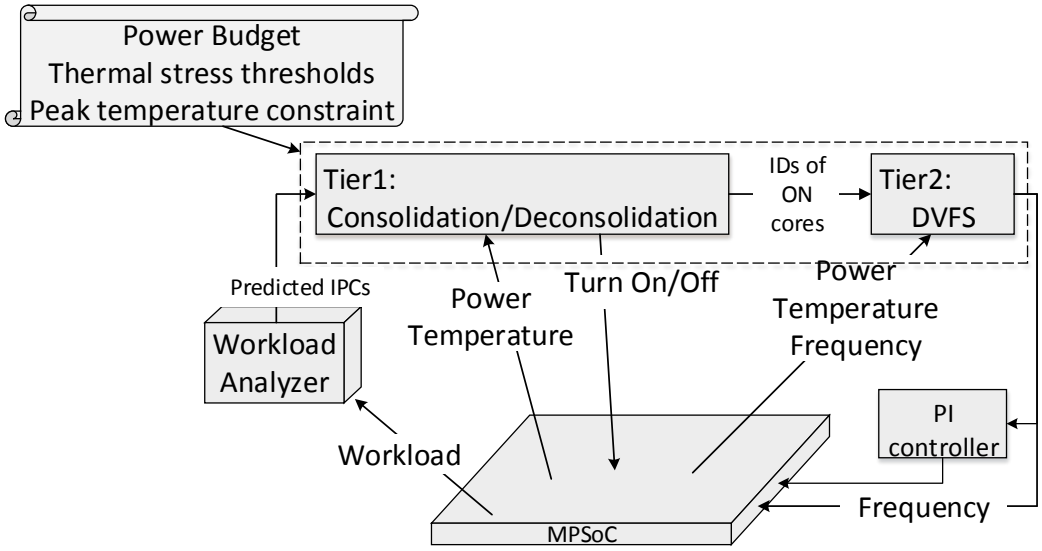


Figure 2.3 – TheSPoT framework

stress reduction. TheSPoT, similar to VPTM, contains a workload analyzer providing the IPS (instruction per second) of the running application by applying a moving average calculation. In contrast to VPTM, TheSPoT includes Tier1 and Tier2 modules modified for thermal stress-aware power and temperature management. Tier1 and Tier2 modules are called at the beginning of their corresponding decision epochs. Tier1 performs the core consolidation and deconsolidation to avoid thermal emergencies while it reduces both spatial and temporal thermal gradients. Tier2 is in charge of per-core DVFS in order to satisfy power budget, peak temperature and thermal stress constraints while considering performance as a primary objective.

In particular, Tier1 receives the predicted IPC (instruction per cycle) values provided by the workload analyzer, reads the current per-core power and temperature, and is aware of power budget and peak temperature constraints. Then, Tier1 delivers the IDs of running cores to Tier2 after having performed the consolidation/deconsolidation according to the thermal stress considerations. Afterwards, Tier2, which is aware of the per-core current operating frequency, power and temperature, recalculates the most appropriate frequencies of the cores to satisfy thermal stress, power, and peak temperature constraints. While the algorithm used for Tier1 is consistent, I propose two different algorithms for DVFS in Tier2. First, the optimal frequencies and voltages of the cores are determined by solving a convex optimization problem. Thereafter, in the second algorithm, I propose a heuristic algorithm to avoid large runtime overhead of the convex optimization solution.

On one hand, in the proposed convex optimization approach, the performance objective (IPS, which is directly dependent on the frequency) is followed by power, peak temperature, and thermal gradient ($\nabla\theta$) constraints. In this formulation, the power and peak temperature constraints are fixed constraints, while $\nabla\theta$ is dynamically changing at runtime based on

the temperature history to provide more opportunities for performance enhancement. The thermal gradient constraint includes both spatial and temporal thermal gradients in this formulation.

On the other hand, the same objective and constraints are defined in the proposed heuristic approach. By considering a margin around the thermal stress thresholds more opportunities are provided to increase the performance. This is similar to the approach taken throughout the proposed convex optimization approach. After following the guidelines introduced in Section 2.5.3.2, the maximum possible frequency that satisfies the thermal gradient constraints is determined. However, given this frequency, the power and temperature constraints must be satisfied. If not, the frequency is reduced until these constraints are met.

Finally, a closed-loop proportional-integral (PI) controller, based on actual measurements, modifies the decisions taken by Tier2 and fine-tunes the DVFS settings at runtime [145]. It makes the power and thermal management robust to workload variations and addresses the overestimation/underestimation caused by the DVFS technique, similar to AVFS proposed by AMD [143].

2.5.1 Heuristic Core Consolidation and Deconsolidation

Tier1 is in charge of core consolidation and deconsolidation as explained in the following subsections.

2.5.1.1 Consolidation

For consolidation, first, a tuple of (i, j), corresponding to the source and destination cores, are selected. The i^{th} core ($Core_i$) is selected if its IPS is smaller than a predefined constant value $IPS_{CONST,i}$, and the cost of its thread migration to $Core_j$ is smaller than $Cost_{Migration}$. $Core_j$ is selected if the consolidation of its thread and the threads of $Core_i$ does not lead to an IPS of more than the maximum IPS allowed for $Core_j$.

Next, for each tuple, the difference between the maximum and the minimum temperatures of the chip is estimated assuming that the consolidation is performed and $Core_i$ is turned off. Therefore, a power of zero for $Core_i$ is assumed while the power of $Core_j$ is elevated by assuming that IPS_j^{new} is equal to summation of IPS values before consolidation, i.e.,$IPS_j + IPS_i$. In particular, the power of the destination core is estimated from the power model proposed by Ghasemazar et al. [145] as follows:

$$P(f,\theta) = d.f^{\beta} + l.f + k_{\theta}.\theta, \tag{2.8}$$

where f and θ are frequency of the core and the temperature, d, l, and k_{θ} are empirical coefficients for dynamic power consumption, temperature-independent and temperature-dependent components of leakage power dissipation, respectively, and β has a value between

2 and 3. Moreover, for power and temperature models I use the same methodology as in [145].

The frequency of the j^{th} core (f_j) is increased such that the core can handle the IPS value required after the consolidation. Therefore, the frequency is obtained by:

$$f_j^{new} = \frac{IPS_i + IPS_j}{IPS_j} \times f_j, \tag{2.9}$$

where f_j^{new} is the frequency of the j^{th} core after the consolidation. This frequency calculation is used in power and thermal models. On the other hand, the target platform provides a number of discrete frequency values to which this calculated value should be mapped. Therefore, my methodology can tolerate inaccuracies in frequency calculation.

Thereafter, based on the relation between the temperature and the power, $\theta(t+\Delta t)$ for all the units of the multi-core processor is obtained as follows [146]:

$$\theta(t+\Delta t) = A.\theta(t) + B.P(t), \tag{2.10}$$

where $\boldsymbol{A}$ and $\boldsymbol{B}$ are $n \times n$ (n is equal to the number of units of the target multiprocessor system) coefficient matrices. These matrices are dependent on the floorplan and technology and are extracted using Hotspot [147, 148]. $\theta(t+\Delta t)$ is an $n \times 1$ vector whose i^{th} row contains the temperature of the i^{th} unit. I use Eq. (2.8) and (2.10) in a loop to model the positive feedback between leakage power and temperature.

After estimating the temperatures of the units, the temperature difference between the coolest and the hottest units of the cores is considered as the temperature cost ($Cost_{\theta,k}$) of the k^{th} tuple. Finally, by using a merit function, the tuple with the smallest cost is selected as follows:

$$M_k = Cost_{Migration,k} + Cost_{\theta,k}, \tag{2.11}$$

where $Cost_{Migration,k}$ is the migration cost of the k^{th} tuple. Three cost types are defined for thread migration: a fixed cost to transfer a few kilobytes of architectural state to the other core, a cost of draining and refilling the pipeline, and warm-up cost for caches [149]. The last two costs are extracted from the sniper simulator [150], while for the first one, in this work, I consider 300 cycles, following the cost model proposed by Van Craeynest et al. [151]. In Eq. (2.11), for the first term, I normalize the migration cost to the maximum value obtained in the iteration. Similarly, the latter is normalized to the maximum temperature difference between the cores in that iteration.

2.5.1.2 Deconsolidation

The core deconsolidation may be performed under two cases. In the first case, the temperature of a core reaches a value higher than the temperature constraint (θ_{Const}) while its frequency

Algorithm 2.1: Proposed optimal DVFS

```
1 if Tier2 epoch then
2     Calculate STG for all pairs of adjacent cores
3 forall core do
4     Determine thermal constraints based on the importance of STG
5     Formulate the convex optimization problem
6     Solve the convex optimization problem
7 forall core do
8     Apply frequencies
```

is equal to its minimum value (f_{min}). In this case, if the core has more than one thread, one thread is chosen to be migrated to another core, rather than turning off the core. It helps decreasing the temporal thermal gradients of the core. In the second case, the frequency of the core is at its maximum value and the core contains more than one thread. Here, the thread with the highest IPS from the core is selected to be migrated to another core. This leads to the performance increase of the source core. In both cases, the destination core for the selected thread is chosen based on the same method used in consolidation.

2.5.2 Optimal DVFS

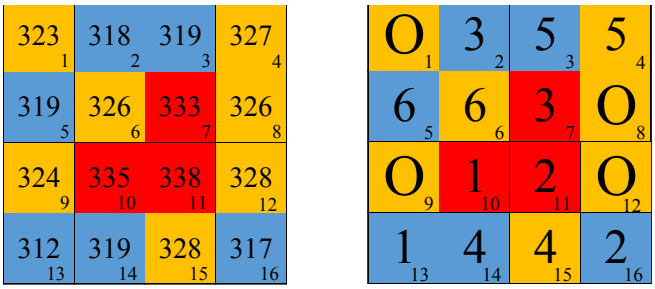
In this section, I include the spatial thermal gradients in the formulation of the convex optimization problem in Tier2. The overall approach for applying the DVFS is shown in Algorithm 2.1.

2.5.2.1 Determining Spatial Thermal Gradient

The constraints used for performing the DVFS are determined based on the existence of the spatial gradient. Thus, I define the spatial thermal gradient as the absolute value of the temperature difference between the two components divided by their corresponding distance measured from their centers. I use ArchFP [152] to obtain the floorplan of the multiprocessor system. I use the center-to-center distance of the cores as the distance between them.

Figure 2.4 shows a schematic of a 16-core processor and illustrates the process of determining spatially stressed cores. In this process, after determining the STG of the adjacent cores, only the values above the STG threshold (STG_{th}) are considered.

In Figure 2.4b, the 10^{th} and the 13^{th} cores are numbered by 1 since they appeared to have the largest temperature difference regarding Figure 2.4a. The 11^{th} and the 16^{th} cores are numbered by 2 as they have the second largest temperature difference after excluding the first pair. Each core may be considered as a stressed core only with one another core. If there are more than one candidates, the two adjacent cores with the highest difference are chosen.



(a) (b)

Figure 2.4 – a) Average core temperature (K), b) Numbering the core pair under spatial stress based on the algorithm

2.5.2.2 Defining Thermal Stress Constraints

To select the optimal frequency of each core, I adapt the formulation proposed by Ghasemazar et al. [145] as the base for Tier2. In addition to the maximum temperature and maximum power constraints, I suggest adding the temporal and spatial thermal gradient constraints. Hence, the increase and decrease rates of the temperature are limited to $\nabla\theta_{INC}$ and $\nabla\theta_{DEC}$, respectively, as follows:

$$\frac{\theta_i(t+\Delta t)-\theta_i(t)}{\Delta t} < \nabla\theta_{INC_i} \tag{2.12}$$

$$\frac{\theta_i(t)-\theta_i(t+\Delta t)}{\Delta t} < \nabla\theta_{DEC_i} \tag{2.13}$$

where $\theta_i(t)$ is the current temperature of the i^{th} unit, Δt is the Tier2 epoch duration, and $\theta_i(t+\Delta t)$ is the temperature of that unit after Δt. Since $\theta_i(t+\Delta t)$ is a function of the frequency, this constraint sets the upper and lower bounds on the frequency change (through the bounds on the thermal variation) of the i^{th} unit in each Tier2 epoch. In order to control the amplitude of the thermal cycle along with the temporal thermal gradient in the DVFS process, I propose to adjust the values of $\nabla\theta_{INC}$ and $\nabla\theta_{DEC}$ dynamically. This adjustment is performed based on the current temperature, the peak and valley temperatures of the unit up to the current point (denoted by θ_P and θ_V, respectively), and a temperature difference threshold (Δt_{Th}). Moreover, the maximum of the absolute value of the temporal gradient is determined by $\nabla\theta_{MAX}$.

Figure 2.5 illustrates the way that the parameters θ_P and θ_V are determined. At the beginning, the valley temperature is equal to the first valley (θ_{V1}). However, the second valley is not considered as a new θ_V since it is not lower than the previous one. Later, θ_{V3} which is lower than the current valley, is considered as a new one. A similar procedure is used for determining the peak temperature. This approach works based on minimizing the thermal cycle amplitude. In the proposed approach, only if the peak (valley) temperature becomes higher (lower), it should be considered in the algorithm for adjusting the frequency. This situation results in an opportunity to improve the performance by not limiting the temperature increase or decrease

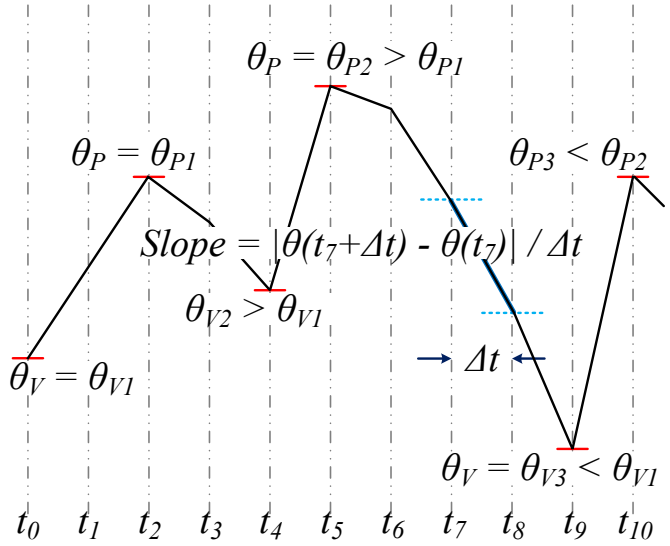


Figure 2.5 – Peak and valley temperatures as well as temporal temperature gradients (Slope). For the sake of simplicity, the transition at the beginning of each epoch has been neglected.

Algorithm 2.2: Temporal thermal constraints when the temperature is increasing

if $\theta_{C,i} > \theta_{P,i}$ **then**
  $\nabla\theta_{INC_i} = \alpha\nabla\theta_{MAX}$
else if $\theta_{P,i} - \theta_{C,i} > \Delta\theta_{Th}$ **then**
  $\nabla\theta_{INC_i} = \nabla\theta_{MAX}$
else
  $\nabla\theta_{INC_i} = \alpha\nabla\theta_{MAX} + (1-\alpha)\nabla\theta_{MAX}\frac{e^{\frac{\theta_{P,i}-\theta_{C,i}}{\Delta\theta_{Th}}}-1}{e-1}$
if $\theta_{C,i} < \theta_{V,i}$ **then**
  $\nabla\theta_{DEC_i} = \alpha\nabla\theta_{MAX}$
else if $\theta_{C,i} > 0.5(\theta_{P,i} - \theta_{V,i}) + \theta_{V,i}$ **then**
  $\nabla\theta_{DEC_i} = \nabla\theta_{MAX}$
else
  $\nabla\theta_{DEC_i} = \alpha\nabla\theta_{MAX} + (1-\alpha)\nabla\theta_{MAX}\frac{e^{\frac{\theta_{C,i}-\theta_{V,i}}{\Delta\theta_{Th}}}-1}{e^{0.5}-1}$

rate. At the beginning of each Tier2 epoch, before solving the convex optimization problem, the temporal thermal gradient constraints are specified. Prior to the constraint formulation, adjacent cores are evaluated to determine whether they are bearing spatial thermal gradients more than a threshold value. If a core does not belong to any pair of the spatially stressed cores, the formulation explained next is used for specifying the thermal gradient constraint. In this formulation, if the temperature of the i^{th} unit in the last Tier2 epoch duration has increased (i.e., positive slope), $\nabla\theta_{INC_i}$ and $\nabla\theta_{DEC_i}$ are defined based on Algorithm 2.2.

In Algorithm 2.2, $\theta_{C,i}$, $\theta_{P,i}$, and $\theta_{V,i}$ represent the current, peak, and valley temperatures of

Algorithm 2.3: Temporal thermal constraint when temperature is decreasing

1 **if** $\theta_{C,i} < \theta_{V,i}$ **then**
2 $\quad \nabla\theta_{DEC_i} = \alpha\nabla\theta_{MAX}$
3 **else if** $\theta_{C,i} - \theta_{V,i} > \Delta\theta_{Th}$ **then**
4 $\quad \nabla\theta_{DEC_i} = \nabla\theta_{MAX}$
5 **else**
6 $\quad \nabla\theta_{DEC_i} = \alpha\nabla\theta_{MAX} + (1-\alpha)\nabla\theta_{MAX}\frac{e^{\frac{\theta_{C,i}-\theta_{V,i}}{\Delta\theta_{Th}}}-1}{e-1}$
7 **if** $\theta_{C,i} > \theta_{P,i}$ **then**
8 $\quad \nabla\theta_{INC_i} = \alpha\nabla\theta_{MAX}$
9 **else if** $\theta_{C,i} < 0.5(\theta_{P,i} - \theta_{V,i}) + \theta_{V,i}$ **then**
10 $\quad \nabla\theta_{INC_i} = \nabla\theta_{MAX}$
11 **else**
12 $\quad \nabla\theta_{INC_i} = \alpha\nabla\theta_{MAX} + (1-\alpha)\nabla\theta_{MAX}\frac{e^{\frac{\theta_{P,i}-\theta_{C,i}}{\Delta\theta_{Th}}}-1}{e^{0.5}-1}$

$Core_i$ and α is a predefined value between 0 and 1. Assume that temperature is increasing. In this formulation, the temperature increase rate is calculated based on the current temperature and the peak temperature up to the previous thermal cycle. This peak temperature is considered as the reference. If the current temperature exceeds θ_P, the increase rate is limited to $\alpha\nabla\theta_{MAX}$, i.e., the smallest temporal thermal gradient constraint (Lines 1 and 2) in this algorithm. If the difference between the current and the peak temperature is more than $\Delta\theta_{Th}$ (Lines 3 and 4), the increase rate is set to its maximum value ($\nabla\theta_{MAX}$). Finally, when the current temperature becomes closer to the peak temperature, the temperature increase rate is reduced exponentially (Line 6). However, the rate cannot be reduced to a value smaller than $\alpha\nabla\theta_{MAX}$.

Moreover, if the slope of the temperature in the last epoch is positive, choosing a lower frequency may help reducing the temperature. Hence, in addition to the temperature increase rate constraint, the decrease rate constraint ($\nabla\theta_{DEC}$) should be determined. The decrease rate constraint is calculated based on the current temperature and the valley of the previous thermal cycle (Lines 7-12). If temperature is increasing in the current epoch, further increase is more probable than the decrease. Hence, in my approach, the temperature increase rate constraint is defined more conservatively than the decrease rate constraint. According to my setup, I experimentally found a small value (say, < 0.1) appropriate for α.

The discussion above is true when the temperature increases during the last Tier2 epoch. If the temperature decreases during the last Tier2 epoch Algorithm 2.3 is followed. In this algorithm, I consider the same principals as explained in Algorithm 2.2, to determine the increase and decrease rate constraints.

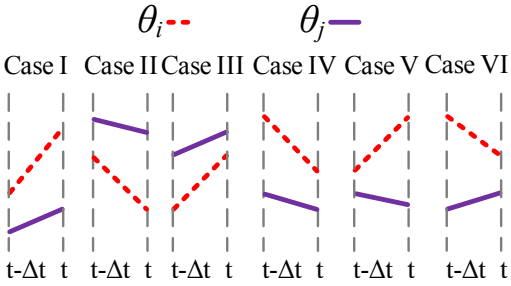


Figure 2.6 – Six cases for temperature trends of a pair of cores under spatial stress

Algorithm 2.4: Increase and decrease rate constraints in Case I and II

if *Case I* **then**
  $\nabla\theta_{INC_j} = \alpha\nabla\theta_{MAX}$
  if $\theta_{P,j} - \theta_{C,j} > \Delta\theta_{Th}$ **then**
    $\nabla\theta_{INC_j} = \nabla\theta_{MAX}$
  else
    $\nabla\theta_{INC_j} = \alpha\nabla\theta_{MAX} + (1-\alpha)\nabla\theta_{MAX}\frac{e^{\frac{\theta_{C,j}-\theta_{L,i}}{\Delta\theta_{Th}}}-1}{e-1}$
if *Case II* **then**
  $\nabla\theta_{DEC_i} = \alpha\nabla\theta_{MAX}$
  if $\theta_{C,j} - \theta_{V,j} > \Delta\theta_{Th}$ **then**
    $\nabla\theta_{DEC_j} = \nabla\theta_{MAX}$
  else
    $\nabla\theta_{DEC_j} = \alpha\nabla\theta_{MAX} + (1-\alpha)\nabla\theta_{MAX}\frac{e^{\frac{\theta_{L,j}-\theta_{C,j}}{\Delta\theta_{Th}}}-1}{e-1}$

2.5.2.3 Formulating Spatial Gradients in Thermal Stress Constraints

When the spatial thermal gradient for a pair of cores is large enough, the thermal gradient constraints explained in Section 2.5.2.2 are formulated differently for both cores. The goal, here, is to modify the constraints given by Algorithms 2.2 and 2.3 to make it more possible to decrease the difference of θ_i and θ_j.

Six cases can occur when the STG value of the two cores is large enough, as shown in Figure 2.6. In Case I, in order to diminish the STG, the increase rate constraint of the core with the higher change rate (say, $Core_i$) should be smaller than that of the other one (say, $Core_j$). Thus, the increase rate constraints of $Core_i$ and $Core_j$ are modified as shown in Algorithm 2.4, where $\theta_{L,j}$ is the temperature of $Core_j$ measured in the last decision time. The decrease rate constraints for both cores are obtained from Algorithm 2.2 due to the STG unimportance. In Case II, the increase rate constraints of both cores are determined by the algorithms introduced in the previous subsection (due to STG unimportance), while the decrease rates are obtained from Algorithm 2.4.

For Cases III and IV, shown in Figure 2.6, the temperature change is such that the STG is lessened as time passes. Hence, I use the constraints given for the cores with no spatial gradient. In Case V, θ_i and θ_j are diverging. To achieve a smaller spatial thermal gradient in the next epoch, both $\nabla\theta_{INC_i}$ and $\nabla\theta_{DEC_j}$ should be limited to the lowest temporal gradient constraint to lower the temperature difference between the two cores. This case is the worst one among the others considered. Thus, $\nabla\theta_{INC_i}$ and $\nabla\theta_{DEC_j}$ are given by:

$$\nabla\theta_{INC_i} = \alpha\nabla\theta_{MAX} \tag{2.14}$$

$$\nabla\theta_{DEC_j} = \alpha\nabla\theta_{MAX} \tag{2.15}$$

where $\nabla\theta_{DEC_i}$ and $\nabla\theta_{INC_j}$ remain unchanged.

In Case VI, where the STG is decreasing, both $\nabla\theta_{DEC_i}$ and $\nabla\theta_{INC_j}$ need to be modified moderately, as follows:

$$\nabla\theta_{DEC_i} = \alpha\nabla\theta_{MAX} + (1-\alpha)\nabla\theta_{MAX}\frac{e^{\frac{\theta_{L,i}-\theta_{C,i}}{\Delta\theta_{Th}}}-1}{e-1} \tag{2.16}$$

$$\nabla\theta_{INC_j} = \alpha\nabla\theta_{MAX} + (1-\alpha)\nabla\theta_{MAX}\frac{e^{\frac{\theta_{C,j}-\theta_{L,j}}{\Delta\theta_{Th}}}-1}{e-1} \tag{2.17}$$

where $\nabla\theta_{INC_i}$ and $\nabla\theta_{DEC_j}$ remain unchanged.

2.5.2.4 Convex Optimization Problem

Having obtained the thermal stress constraints, I form a convex optimization problem including power and thermal constraints and the frequency domain by:

$$\begin{aligned} Maximize \sum_{i=1}^{\|cores\|} IPS_i X_i \;\; s.t. \\ A.\theta + B.P < \theta_{CONST} \\ P < P_{budget} \\ f_{MIN} < f < f_{MAX} \\ P = D.f^{\beta} + L.f + K_{\theta}.\theta \\ \frac{A.\theta(t) + B.P) - \theta(t)}{\Delta t} < \nabla\theta_{INC_i} \\ \frac{\theta(t) - (A.\theta(t) + B.P)}{\Delta t} < \nabla\theta_{DEC_i} \end{aligned} \tag{2.18}$$

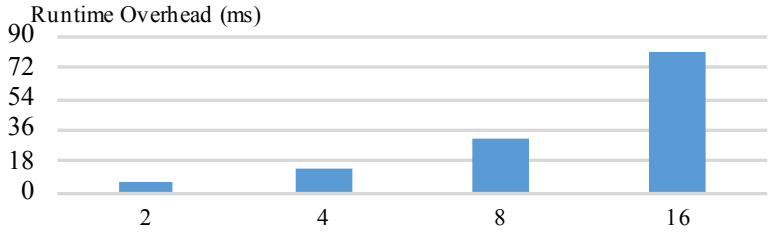


Figure 2.7 – Runtime overhead of the optimization solution for different number of cores

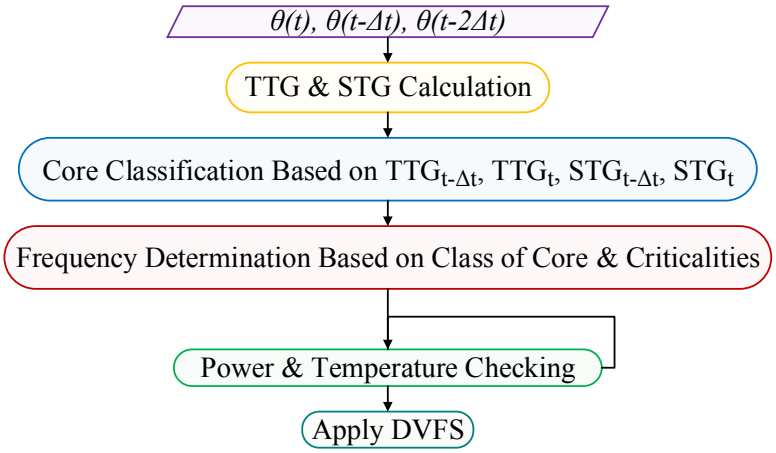


Figure 2.8 – The proposed flowchart of the heuristic DVFS algorithm in Tier2

After determining the frequency of $Core_i$, its corresponding voltage $V_{supply,i}$ is also calculated using:

$$V_{supply,i} = V_{min,i} + (V_{max,i} - V_{min,i}) \times \frac{f_i - f_{min,i}}{f_{max,i} - f_{min,i}}, \tag{2.19}$$

where $V_{min,i}$, $V_{max,i}$, $f_{min,i}$, and $f_{max,i}$ show the minimum supply voltage, maximum supply voltage, minimum frequency and maximum frequency of $Core_i$, respectively.

2.5.3 Heuristic DVFS

Although the proposed DVFS approach brings about the optimal frequencies for the power and thermal management problem constrained by thermal stress, it may fail to deal with real-time application due to a large runtime overhead. Figure 2.7 shows the runtime overhead for *facesim* benchmark from PARSEC benchmark suit [153] obtained from the proposed optimal solution.

As the number of cores increases, the runtime overhead rises super-linearly and makes this solution infeasible for multi-core SoCs. The decision intervals of DVFS in literature ranges from a few milliseconds to seconds in literature [154, 11]. As I discuss it later in Section 2.5.4, I consider 5ms intervals for DVFS. Thus, according to this setup, the relative runtime

overhead is as large as 16 times of the decision intervals for the case of 16-core MPSoCs. Hence, due to the large computational overhead of the optimal solutions, I propose a new heuristic DVFS algorithm in Tier2, based on rules and motivated by the runtime objectives and constraints. The flowchart of the algorithm is shown in Figure 2.8. The proposed heuristic DVFS considers the thermal stress constraint and available power and temperature budgets and has the objective of increasing the frequency (and the performance) as much as possible.

In this algorithm, first, the temporal and spatial thermal gradient (TTG and STG) of each core are calculated. Then, the cores are classified based on the values of $STG(t)$, $STG(t-\Delta t)$, $TTG(t)$, and $TTG(t-\Delta t)$ of each core to decide on the existence of a kind of thermal stress for the core. In this notation, t and $t-\Delta t$ correspond to the current and last time epochs, respectively.

Here, predefined threshold values, TTG_{th} and STG_{th}, are used for the core classification with respect to the thermal stress type. The classification includes cores under TTG (C_{TS}), under STG (C_{SS}), under both TTG and STG (C_{TSS}), and Relaxed (C_R) which are discussed, in detail, in Section 2.5.2.2. Based on the assigned class and the trend of the core temperature variation, the frequency of each core is determined. Before applying the calculated frequencies, the temperature and power consumption of each core in the next epoch are predicted to check whether the power and/or temperature constraints are not violated.

2.5.3.1 Core Classification

First, the class of each core based on the temporal and spatial thermal gradients measured in the current and previous epochs is determined.

C_{TS}. If a core is under TTG, it is labeled as C_{TS}. I classify the stressed cores based on the following criteria:

- the lowest criticality ($C_{L,TS}$): $(1-\gamma)TTG_{th} < TTG(t) < TTG_{th}$ and $TTG(t-\Delta t) < TTG(t)$ (increasing gradient) or $TTG_{th} < TTG(t) < (1+\gamma)TTG_{th}$ and $TTG(t) < TTG(t-\Delta t)$ (decreasing gradient),
- the medium criticality ($C_{M,TS}$): $(1+\gamma)TTG_{th} < TTG(t) < TTG(t-\Delta t)$, and
- the highest criticality ($C_{H,TS}$): $TTG_{th} < TTG(t)$ and $TTG(t-\Delta t) < TTG(t)$.

Here, γ is a coefficient between 0 and 1 which is determined through 10 simulations with small inputs. Figure 2.9 illustrates the regions corresponding to each criticality level considering the trends of temperature change.

C_{SS}. The spatial thermal gradient is defined based on the temperature variation of two neighbor cores and, hence, the spatial stress is considered only for pairs of cores. Here, I define the

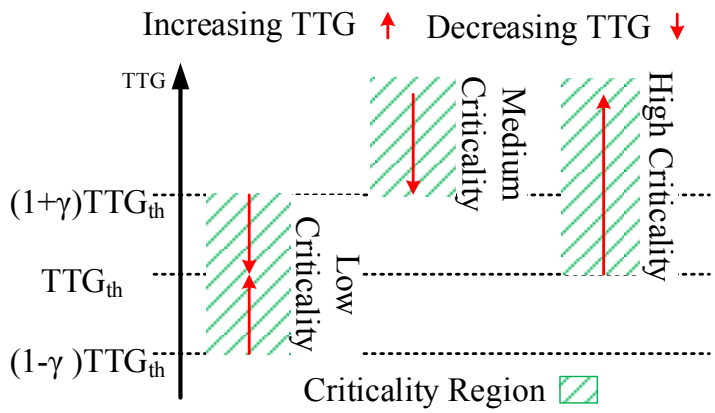


Figure 2.9 – Regions for different TTG criticality levels

coefficient λ between 0 and 1 obtained from simulations. Also, similar to the previous case, I consider three levels of criticality:

- the lowest criticality ($C_{L,SS}$): $(1-\lambda)STG_{th} < STG^{i,j}(t) \leq STG_{th}$ provided that $STG^{i,j}(t-\Delta t) < STG^{i,j}(t)$ or $STG_{th} < STG^{i,j}(t) < (1+\lambda)STG_{th}$ provided that $STG^{i,j}(t) < STG^{i,j}(t-\Delta t)$,
- the medium criticality ($C_{M,SS}$): $(1+\lambda)STG_{th} < STG^{i,j}(t) < STG^{i,j}(t-\Delta t)$, and
- the highest criticality level ($C_{H,SS}$): $STG_{th} < STG^{i,j}(t)$ and $STG^{i,j}(t-\Delta t) < STG^{i,j}(t)$.

Here, $STG^{i,j}(t)$ is the current spatial gradient for the pair of $Core_i$ and $Core_j$. The regions for the STG criticality level can be demonstrated by replacing TTG by STG and γ by λ in Figure 2.9.

C_{TSS}. If in a pair of cores under STG, there is at least one core under TTG, the whole pair is labeled as C_{TSS}.

C_R. When a core is not under any kind of stress, it is labeled as C_R.

2.5.3.2 DVFS

To perform an appropriate DVFS scheme that alleviates the thermal stress and provides higher performance, the following guidelines are considered:

- **G1:** To reduce the temporal gradient, the frequency needs to be changed in a way to oppose the direction of current temperature trend.

Algorithm 2.5: Frequency change required for cores with C_{TS} and C_{SS} labels

```
forall C_TS do
    if frequency decrease required then
        use(CF_level,TTG)
    else
        use(CF_level,TTG + CF_performance)
forall C_SS do
    forall core in the pair do
        if frequency decrease required then
            if temperature ascending then
                use(CF_level,STG − CF_performance)
            else
                No change
        else
            if temperature ascending then
                use(CF_performance)
            else
                use(CF_level,STG + CF_performance)
```

- **G2:** The amount of decrease or increase in the frequency of a core must strongly depend on its stress type and criticality.
- **G3:** Since obtaining a higher performance is the main goal, reducing the thermal gradient is preferred to be solved by increasing the frequency rather than decreasing it.
- **G4:** Since in Case V (Figure 2.6) the STG worsens more quickly than the other cases, the frequency should change more.
- **G5:** In case of C_{TSS}, alleviating STG and TTG can be achieved through trade-offs between spatial and temporal gradients.
- **G6:** Excessive change of the frequency may either turn a relaxing core into a stressed one, or adversely affect the other stress type, or cause thermal stress in the opposite direction.

Algorithm 2.5 describes the frequency change applied for the cores labeled as C_{TS} and C_{SS}. In this algorithm, $CF_{level,TTG}$ represents the change in frequency based on the criticality level of the TTG, and $CF_{performance}$ is the change in frequency to obtain higher performance (G3). Similarly, $CF_{level,STG}$ represents the change in frequency based on the criticality level of the STG.

Based on G5, the application of the DVFS scheme by considering $C^{i,j}_{STG}$, C^{i}_{TTG}, and C^{j}_{TTG}, may require some compromise. First, I note that when the TTG value of a core is more than that of

the other's, it does not necessarily mean that its TTG-related criticality level is also higher (see Figure 2.9). Also, for a pair of cores labeled as C_{TSS}, there may be only one temporally stressed core and the TTG criticality for the other is considered to be zero.

Algorithm 2.6 shows the proposed DVFS scheme for the cores labeled as C_{TSS}. The most appropriate DVFS settings are those that consider Algorithm 2.5, as the baseline and, at the same time, provide trade-offs wherever the frequency changes suggested for C_{TS} and C_{SS} do not agree with each other. Algorithm 2.6 determines proper frequency changes to simultaneously consider cores under TTG and STG. The term "changes" is replaced by "decreases" or "increases" based on the appropriate change suggested by Algorithm 2.5. Also, C^i_{TTG} denotes the criticality level of TTG for $Core_i$.

The frequency of C_R cores could be increased to achieve a higher performance. Since an excessive increase in the frequency leads to a thermal stress (G6), the process should be performed carefully. For this reason, when the TTG is (is not) positive, the frequency of the relaxed core is increased by two (three) steps. These numbers are obtained for my simulations where the frequency range is divided by 15 to determine the frequency steps for the target multiprocessor system. Finally, 4 (4), 3 (3), and 2(2) are considered for $CF_{H,TS}(CF_{H,SS})$, $CF_{M,TS}(CF_{M,SS})$ and $CF_{L,TS}(CF_{L,SS})$, respectively.

2.5.3.3 Power and Temperature Checking

Before applying the frequencies obtained from Section 2.5.3.2, the temperature and power consumption of the cores in the next epoch are predicted. First, based on the model given by Eq. (2.10) which depends only on the current temperature and power consumption, the next temperature of each core is calculated. Then, using the new temperature and frequency, the total power consumption is obtained based on Eq. (2.8). Afterwards, the total power consumption (P_{total}) is compared with the power constraint (P_{const}). If P_{total} is larger than P_{const}, the frequency of the most power-consuming core (f_{mpc}) is lowered one step. This procedure continues until P_{total} becomes lower than P_{const}, or f_{mpc} equals to the minimum frequency. Using one step of frequency decrease is motivated since the initial frequencies are computed according to detailed guidelines to minimize the spatial and temporal gradients while considering performance. Any large deviation from these pre-calculated frequencies, thus, may result in performance overhead or violation of guideline G6. Figure 2.10 shows the whole procedure.

After considering the total power consumption, the temperature of each core is predicted. If the temperature of any core exceeds θ_{const}, its preassigned frequency (f_i) is decreased one step. This procedure lasts until the predicted temperatures of all the cores become lower than θ_{const}, or f_i is no longer larger than $f_{min,i}$. If the temperature and power constraints could not be met by the frequency reduction, the control of the algorithm is transferred to Tier1 which can invoke consolidation/deconsolidation procedure. It is preferred to satisfy θ_{const} and P_{const} in Tier2 rather than in Tier1 since the consolidation procedure may lead to turning

Algorithm 2.6: Frequency change required for cores with C_{TSS} labels

if *Case I or II* **then**
  if $C^i_{TTG} \geq C^j_{TTG}$ **then**
    f_i **changes** $max(C^i_{level,TTG}, C^i_{level,STG})$
    f_j **changes** $C^j_{level,TTG}$
  else if $C^i_{TTG} = C^j_{TTG}$ **then**
    f_i **changes** $max(C^i_{level,TTG}, C^i_{level,STG} + 1)$
    f_j **changes** $C^j_{level,TTG}$
  else
    f_i **changes** $C^i_{level,TTG}, C^{i,j}_{level,STG}$
    f_j **changes** $C^j_{level,TTG}$
if *Case III or IV* **then**
  if $C^i_{TTG} \geq C^j_{TTG}$ **then**
    f_i **changes** $CF^i_{level,TTG} - 1$
    f_j **changes** $CF^j_{level,TTG}$
  else
    f_i **changes** $CF^i_{level,TTG}$
    f_j **changes** $max(C^j_{level,TTG}, C^{i,j}_{level,STG})$
if *Case V* **then**
  f_i **changes** CF^i_{TTG}
  f_j **changes** $max(C^j_{level,TTG}, C^{i,j}_{level,STG})$
if *Case VI* **then**
  f_i **changes** $C^i_{level,TTG} - 1$
  f_j **changes** $C^j_{level,TTG} - 1$

a core off which reduces the performance compared with the case when the core is running even with the minimum frequency.

2.5.4 Experimental Setup

I evaluate the efficiency of TheSPoT in tackling thermal cycling and thermal gradients using the PARSEC [153] benchmark suit. For comparison, I implement the dynamic power/thermal management approach proposed by Coskun et al. [88] which employs DVFS (including $f_{min,i}$ and $f_{max,i}$) and thread migration.

The simulation framework is implemented in the Sniper multi-core simulator [150]. The power consumption and the temperature of the multiprocessor system are estimated using McPAT [155] and Hotspot [147], respectively. To extract the floorplan of the target multiprocessor system, I use ArchFP [152], where the areas of different parts are extracted using McPAT based

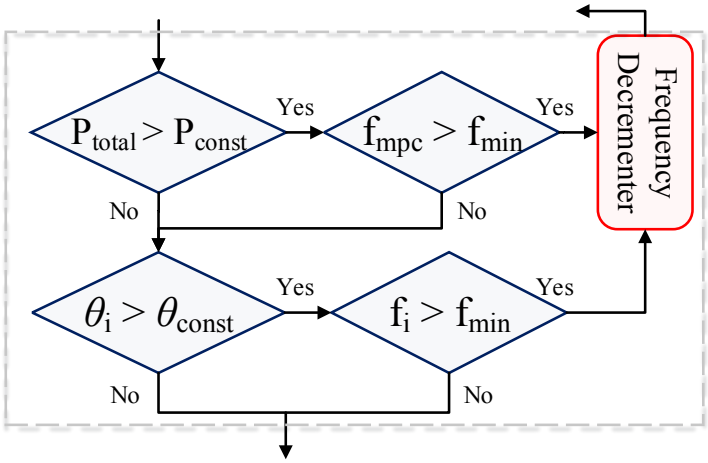


Figure 2.10 – Proposed flowchart of Power and Temperature Checking

Table 2.1 – Thermal values

TTG_{th}	TTG_{th}	$\theta_{ambient}$	θ_{const}
0.80 K/ms	0.25 K/mm	310K	340K

on a 45nm technology. TheSPoT is implemented using Python programming language. Also, for the case of my optimal approach, the convex problem of Tier2 is solved by using NLOPT [156]. This tool-chain carefully takes into account any change in workload on any core and provides the corresponding performance, power and temperature values such that thermal gradients can be considered accurately. Moreover, I rely on McPAT support for modeling the wake-up power and delay overheads. Finally, in order to consider DVFS overhead, I use a micro-architectural parameter provided by Sniper simulator and set it to $10\mu s$ [157].

For all simulations, Tier1 and Tier2 epochs are 10ms and 5ms long, respectively. For a fair comparison, I consider 10ms decision interval in the work of Coskun et al. [88]. For all simulation scenarios l consider large inputs. Table 2.1 shows the ambient temperature, temperature constraint, and the threshold values for TTG and STG. The temperature constraint is defined by the user and considered as the critical temperature of core. Moreover, TheSPoT is valid for any threshold values. In addition, I have used the same threshold values for the three algorithms for all the studies. Also, it is clear that lower values of the thresholds provide less thermal stress at the cost of more performance reduction (mainly due to frequency reduction and the migration overheads). Hence, based on my simulations, I found these values to provide a trade-off between the stress reduction and the performance. I consider 15 degrees as the minimum amplitude for counting the thermal cycles [88]. Using this value, the total number of thermal cycles for all the epochs is counted. In addition, the amplitude of thermal cycles for each simulation scenario is attained by accumulating thermal cycle amplitudes. For the performance (time required to finish processing a job by a benchmark for a given input), I invoke the number of Tier2 epochs used for finishing the job.

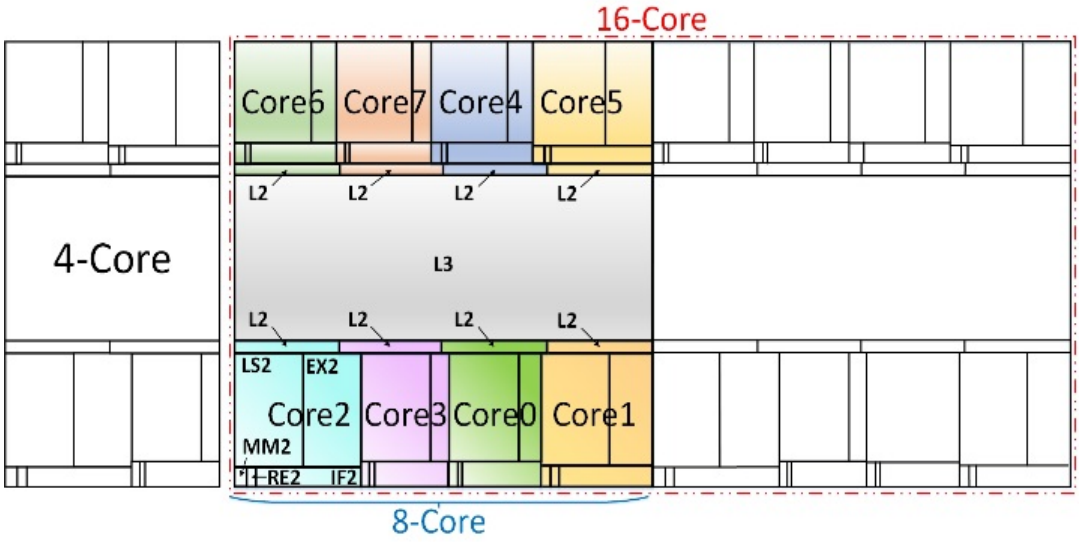


Figure 2.11 – Floorplan of the 4-core, 8-core, and 16-core processors

Table 2.2 – Design parameters of the target multiprocessor system architecture

Core	$Power_{const}$ (Watt)	Dispatch Width	Freq. boundaries (GHz)	L3 (MB)
4	70	4, 6, 8, 2	$F = \{f_{b1}, f_{b2}, f_{b3}, f_{b4}\}$	8
8	120	4, 6, 8, 2, 4, 6, 4, 2	F, F	32
16	200	4, 6, 8, 2, 4, 6, 4, 2, 4, 6, 8, 2, 4, 6, 4, 2	F, F, F, F	64

I consider 4-, 8-, and 16-core x86 processors. Each processor is based on Nehalem Intel architecture and derived from Gainestown model code-name. Each core comes with one L1 (32 KB) and one L2 (256 KB) private caches, while one L3 cache whose size depends on the number of cores is shared among the cores. All cores are out-of-order and can carry out up to two threads simultaneously. Each core consists of five separate functional units including instruction fetch (IF), renaming (RE), execution (EX), load/store (LS), and memory management (MM). The floorplans of the multiprocessor systems studied are shown in Figure 2.11. Units are only labeled for the second core.

Table 2.2 shows the dispatch width, frequency boundaries, power constraints, and L3 cache size (MB). I consider 4 different frequency (GHz) boundaries, $f_{b1} = [1.2, 2.5]$, $f_{b2} = [1.3, 2.66]$, $f_{b3} = [1.2, 2.5]$, and $f_{b4} = [1, 3]$.

2.5.5 Experimental Results

2.5.5.1 Thermal Stress Reduction

Table 2.3 presents the achieved reduction in STG, TTG, TCN (thermal cycle number), and TCA (thermal cycle amplitude) along with the performance overhead (Perf. Ovh.) of the proposed TheSPoT with heuristic and optimal DVFS normalized to those obtained from state-of-the-art (SoA) [88], for the 8-core MPSoC. In the rest of this section, I refer to TheSPoT with heuristic

Table 2.3 – Average reduction in spatial temperature gradient, temporal temperature gradient, thermal cycle number, and thermal cycle amplitude, and performance overhead

Benchmark	Optimal TheSPoT (%)					Heuristic TheSPoT (%)				
	STG	TTG	TCN	TCA	Perf. Ovh.	STG	TTG	TCN	TCA	Perf. Ovh.
blackscholes	18	10	32	18	3.5	24	11	35	21	4.8
bodytrack	15	11	40	23	4.2	25	11	41	23	5
canneal	10	12	26	25	4	16	10	19	19	6.1
dedup	21	25	34	29	5.1	35	24	38	34	8.3
facesim	18	14	17	23	5.7	27	16	20	26	6
freqmine	5	11	27	19	3.8	22	14	23	31	5.5
vips	5	14	17	19	4.1	14	13	19	12	4.3
x264	16	10	22	26	4.5	26	17	25	23	4.8
ferret	22	18	35	36	5.6	32	21	34	41	7.9

and optimal DVFS approaches as *heuristic* and *optimal*, respectively.

The achieved reduction in thermal stress strongly depends on the nature of benchmarks. For the benchmarks where the workload variations do not cause high temporal or spatial thermal gradients, the proposed approaches do not provide considerable TTG/STG reductions. This is due to the fact that only a few thermal stress violations occur and the predefined thermal stress constraints and thresholds rarely trigger TheSPoT framework to further improve the lifetime reliability. TheSPoT specially outperforms SoA [88] for benchmarks such as *ferret* and *dedup* featuring different functions with different characteristics at the same time [149]. This improvement occurs because TheSPoT makes decisions based on thermal variations and not only the peak temperature. Conversely, the work proposed by Coskun et al. [88] triggers decisions mainly based on the peak temperature.

To better understand how TheSPoT is effective in increasing lifetime reliability in terms of MTTF, I exploit the same methodology and formulation suggested by Srinivasan et al. [85]. In addition, I modify MTTF formulation of the TDDB and EM [89] in order to include spatial and temporal thermal gradients impact on lifetime reliability. Overall, the MTTF of the proposed optimal and heuristic approaches increased on average, by 35% and 47%, respectively, compared with that obtained by SoA [88]. I considered SM, EM , TDDB, and Thermal Cycling as the most significant failure mechanisms. Also, the proposed heuristic TheSPoT outperforms the optimal TheSPoT in terms of MTTF improvement since the heuristic solution explicitly considers both spatial and temporal gradients as the first order design objectives while trying to increase the performance. On the contrary, the primary objective of the proposed optimal approach is to maximize the performance. Therefore, the optimal TheSPoT achieves less performance overhead in comparison with the heuristic solution.

Due to lack of access to some technological parameters, I report the relative improvement achieved compared to that of [88] as the reference work. However, considering a typical Intel server operating at the ambient temperature of $35°C$, the estimated MTTF would be approximately 200000 hours[1]. Thus, assuming no thermal stress-aware power management the MTTF of the system is 200000 hours, whereas the heuristic TheSPoT, optimal TheSPoT

[1] https://www.intel.com/content/www/us/en/support/server-products/000007224.html

Figure 2.12 – Number of thermal violations occurred in one run of *blackscholes* benchmark for different number of cores when no thermal stress-aware approach applied

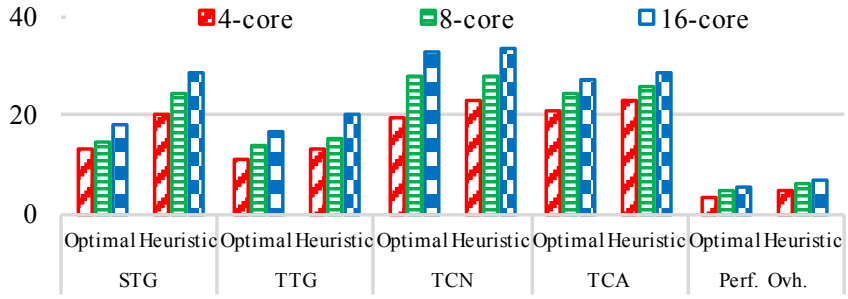


Figure 2.13 – Average reductions (%) in STG, TTG, TCF, TCA, and performance overhead for TheSPoT compared to SoA [88] for 4-, 8- and 16-core MPSoCs

and SoA [88] will result in 455700, 418500, and 310000 hours, respectively.

Figure 2.12 shows the average number of thermal stress violations (STG, TTG, and TC), counted regarding the preset threshold values as the number of cores on the multiprocessor system changes for a basic power and temperature management approach which only considers peak power/temperature values under *blackscholes* benchmark. As shown in the figure, when no thermal stress-aware power and thermal management technique is evoked for the target multiprocessor system more thermal variation occurs both spatially and temporally if the number of cores increases. When the available resources scales, the scheduler faces more choices to run the jobs at each decision time. However, it is unaware of the decision impact on workload variations and, hence, temperature variations across the chip result in more thermal stress violations.

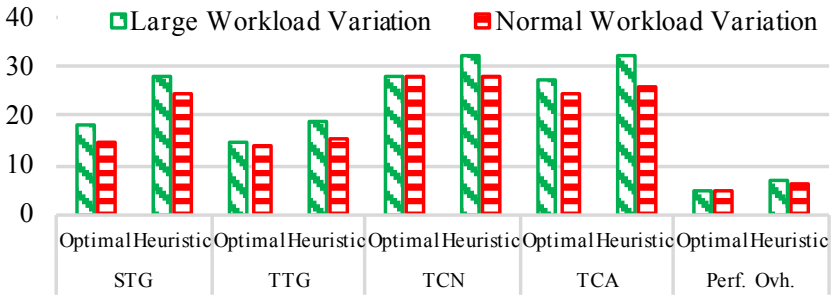


Figure 2.14 – Average reductions (%) in STG, TTG, TCF, TCA, and performance overhead for TheSPoT compared with SoA [88] for different workload variations

Figure 2.13 provides the average reduction percentages of STG, TTG, TCN, and TCA along with the performance overhead obtained from the proposed methods compared with those of SoA [88] for 4-core, 8-core and 16-core MPSoCs. My thermal stress-aware approaches outperform SoA [88] with respect to the thermal stress reduction with only a negligible performance overhead as the number of cores increases. In TheSPoT, as the number of cores increases, Tier1 is able to find better source and destination cores for consolidation and deconsolidation, which leads to a higher reduction in thermal stress occurrences. In contrast, SoA [88] assigns the ready jobs to the coolest core with idle neighbors, which increases the risk of high amplitude thermal cycles.

When I scale the platform, both proposed approaches efficiently reduce thermal stress. Nevertheless, the optimal approach fails to be applicable for many-core processors due to the large runtime overhead, while my heuristic algorithm comes with only 5ms runtime overhead at each decision intervals even for larger number of cores. As aforementioned, the larger the thermal variation is in time or space, the more efficient my thermal stress-aware approaches are. To demonstrate this hypothesis, I compare the above simulation scenario, running all benchmarks separately and then averaging the results, called normal workload variation, with a new scenario where all benchmarks are released and run simultaneously (large workload variation). Figure 2.14 reveals more reduction in thermal stress parameters when the thermal variations (workload variations) are larger. However, this achievement comes with approximately 1% more performance overhead. On the contrary, although SoA [88] considers temperature variations, it uses peak temperature as the trigger. Thus, it cannot properly control the thermal stress.

2.5.5.2 Comparison of Performance and Runtime Overhead

On average, for the proposed heuristic(optimal) approach, STG, TTG, TCN, and TCA are, respectively, decreased by 25(14)%, 15(14)%, 28(28)%, and 26(24)% compared with those of SoA [88] with only 6(4.5)% performance degradation. The performance overhead of the proposed approaches in comparison to SoA [88] originates from, first, the reduced average of the operating frequency, and second, more frequent thread migrations as shown in Table 2.4. The proposed technique by Coskun et al. [88] operates with the maximum available frequency unless a thermal emergency occurs; then, it works with the minimum frequency. Nevertheless, if the number of peak temperature violations increases for a specific benchmark, the overall performance overhead of TheSPoT would decrease compared with that of SoA [88]. Both optimal and heuristic approaches reveal almost the same number of thread migrations, since they employ the same approach for consolidation and deconsolidation. Therefore, the difference in the performance overhead is mainly due to the operating frequency as the optimal approach looks for the optimal frequencies while the heuristic one provides near-optimal values.

TheSPoT is able to take advantage of available hardware and knobs dedicated to power and thermal management of modern multiprocessor systems [72]. However, any software im-

Table 2.4 – Total number of thread migrations, and average operating frequencies of on cores

Benchmark	Number of Thread Migrations			Average Frequency of ON cores (GHz)		
	Optimal	Heuristic	[88]	Optimal	Heuristic	[88]
blackscholes	9	11	9	2.11	2.05	2.2
bodytrack	15	15	5	2.21	2.16	2.24
canneal	7	8	4	1.98	1.85	2.1
dedup	64	60	57	1.9	1.74	2.21
facesim	292	312	286	1.77	1.75	2.23
freqmine	23	18	10	1.85	1.75	2.04
vips	14	13	11	1.83	1.8	2.04
x264	77	59	32	1.92	1.88	2.13
ferret	22	20	10	1.88	1.63	2.15

plementation is accompanied by runtime overhead. In contrast to the optimal solution, the proposed heuristic algorithm comes with only 5ms computational runtime overhead for 8-core MPSoC. This 5ms overhead is almost constant when using larger number of cores. On the other hand, the computational overhead of SoA [88] is the same as that of my heuristic approach. All in all, the efficacy of TheSPoT is not limited to choosing a 10ms decision epoch (the same interval has been used in several simulation-based works [154]). Although there is trade-off between the thermal stress reduction and runtime overhead of any thermal aware approach [88], when larger decision epochs are used, TheSPoT still considerably outperforms SoA [88] with respect to the achieved MTTF enhancement. However, both approaches encounter slight degradation in the thermal stress reduction. In particular, the MTTF obtained (I performed experiments with *facesim*, and x264 benchmarks on the 8-core processor) from TheSPoT and SoA [88] decreases by 9% and 6%, respectively when using 100ms decision epoch instead of 10ms epochs.

As a trade-off between runtime overhead and thermal stress reduction, I chose 10ms to focus more on the thermal stress reduction. I recall that, by using the same experimental setup for both TheSPoT and SoA [88], I conduct a fair comparison, showing the same runtime overhead but 47% MTTF enhancement for my proposed approach. Reporting the algorithm performance overhead (degradation/improvement) and its runtime overhead separately provides a better insight into comparing different approaches since the runtime overhead, regardless of the decision epoch time, is constant for each scenario.

The heuristic approach ends up with the near-optimal frequency, on average 2% less performance when compared to the proposed optimal solution. Nevertheless, this performance reduction comes with MTTF improvement. This MTTF enhancement comes from detailed guidelines based on a longer thermal profile history. Specifically, the difference is more obvious for STG reduction, since the proposed heuristic approach considers STG more explicitly when determining the frequencies of the cores.

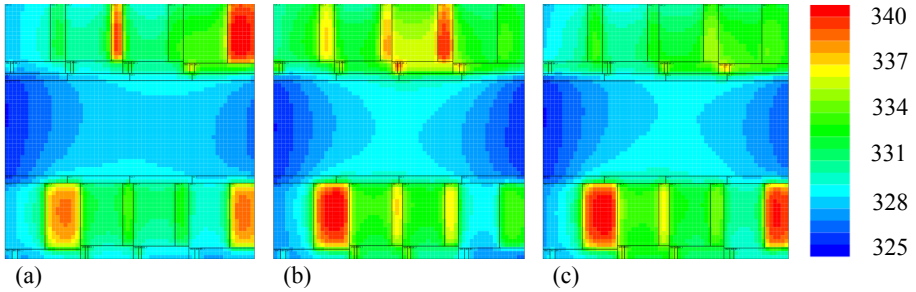


Figure 2.15 – Thermal map (K) obtained from a) [88], b) optimal, and c) heuristic approaches under *facesim* benchmark for 8-core MPSoC

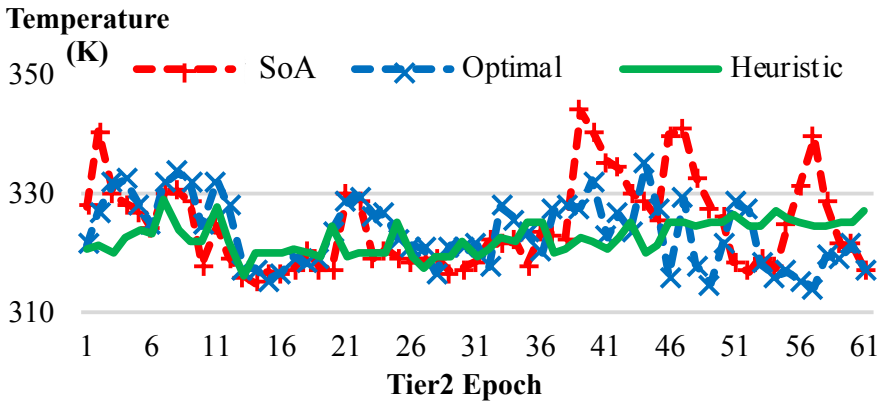


Figure 2.16 – Average temperature (K) of the first core under *facesim* benchmark

2.5.5.3 Evaluation of Thermal Stress-Aware Power Management

In this subsection, I show how my thermal stress-aware techniques are able to manage the power and temperature while maintaining fewer thermal stress violations compared with SoA [88]. For this purpose, I choose *facesim* whose simulation results almost conform to the average values.

Figure 2.15 shows the thermal profiles of the 8-core processor obtained by TheSPoT and SoA [88]. As shown, the spatial temperature gradients obtained by TheSPoT are smaller than those of SoA [88], even though in the selected window of *facesim* simulation, the maximum temperatures across the chip in all three cases are similar.

The average temperature of the 1^{st} core, depicted in Figure 2.16 for the first 61 intervals (Tier2 epoch), reveals more temperature variations for the SoA [88]. As several threads are launched at the same time, thread migration and core consolidation as well as DVFS add to the thermal variation observed on a single core. Hence, large thermal cycles can be noticed not only for the start and end of a simulation, bus also more peak temperature variation are observed for this core when the thermal management of SoA [88] is applied. In particular, SoA [88] fails to prevent large thermal variation, since it is not the main trigger of its DTM policy. The total power consumption (Watts) of the MPSoC over time is shown in Figure 2.17. The average

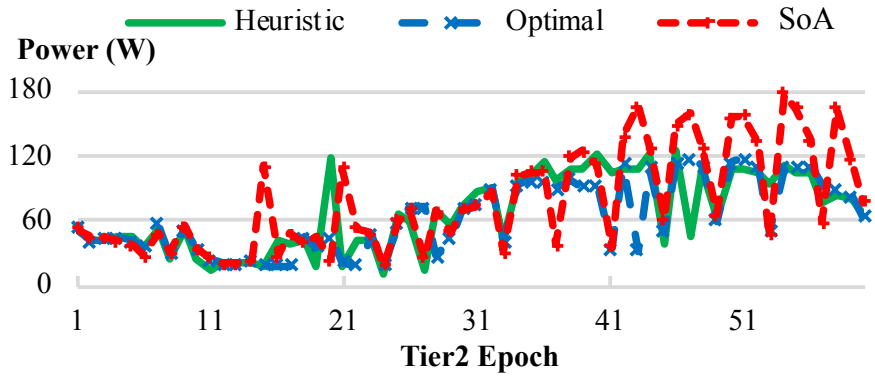


Figure 2.17 – Total power consumption (Watts) of the 8-core MPSoC under facesim

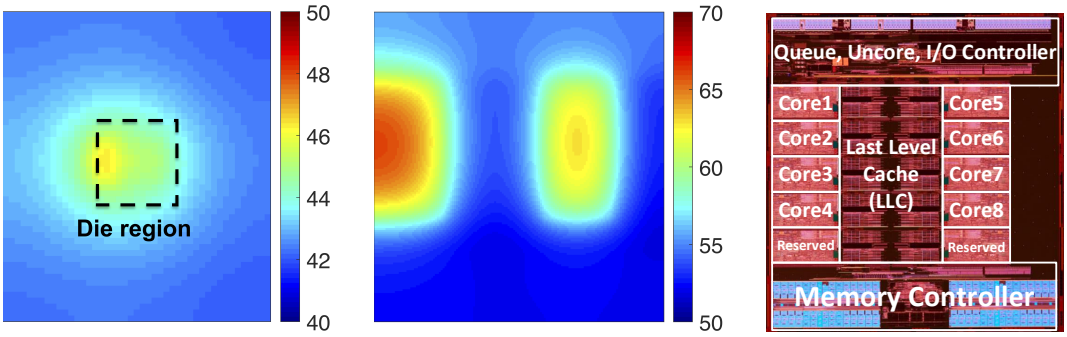
power consumption attained by SoA is higher than those resulted from TheSPoT. The power consumption exceeds the power constraint (120 watts for 8-core MPSoC) at a few points since SoA does not provide any mechanism to control it.

2.6 Proposed Workload- and Cooling-Aware Thermal Management

A holistic DTM policy should take into account all available design- and run-time parameters, including the available cooling system. Specially, cooling systems are widely used in thermal management of high-performance and power-hungry multiprocessor systems. Micro-scale thermosyphon, is a gravity-driven two-phase liquid cooling technology that can be directly mounted on top the processor, providing the highest cooling efficiency in terms of PUE among all existing state-of-the-art cooling systems and technologies. However, similar to other technologies, thermosyphon has particular features demonstrating its potential and limit. Without understanding these features and adapting conventional DTM methods it is not possible to take the most possible advantage of this recently manufactured cooling technology.

Apart from mechanical hardware design and manufacturing challenges [20], platform- and workload-awareness play a significant role in thermosyphon efficiency for removing high heat fluxes. This, in return, can enhance performance metrics of power-hungry servers. Moreover, once designed and manufactured, a thermosyphon still provides the user with adjustable parameters to tune its performance according to the workload requirements. Although similar profound research [124] is available for more conventional cooling technologies, two-phase micro-scale thermosyphons have not been studied yet.

Considering that the evaporator size (Figure 2.1) scales linearly with respect to the CPU dimension, the most important design-time parameters that drive heat flux removal are the filling ratio of the refrigerant, and the refrigerant type. Other important parameters affecting thermosyphon efficiency are the inlet coolant temperature and its flow rate. These parameters can be tuned at runtime according to the CPU workload. Existing DTM approaches, unaware of



(a) Package temperature

(b) Die temperature

(c) Die floorplan of Intel Xeon E5 v4

Figure 2.18 – Die thermal profile vs. package thermal profile when using thermosyphon with non-optimized design and workload mapping strategy.

these run-time parameters, cannot be effectively used for thermal management of a processor equipped with thermosyphon.

Seuret et al. [20] evaluate the thermosyphon efficiency considering a uniform heat flux over the whole chip. However, this is not a realistic assumption for current applications and multiprocessor systems, as different workload mappings lead to non-uniform heat flux, which ultimately cause hot spots and spatial thermal gradients [158]. Moreover, Seuret et al. [20] assume that the heat flux received by the thermosyphon is the total power generated by the die divided by the surface of the package. This assumption, however, is too simplistic, as the heat flux is larger on the package area right above the die, even in the presence of a heat spreader [158].

In addition, temperature of the processor package and die are reciprocally dependent. In fact, hot spots and spatial thermal gradients on the die are scaled-up of those on the package. For instance, a thermal hot spot of $46°C$ and spatial gradient of $0.5°C/mm$ on the package, may lead to a hot spot and spatial gradient of $66°C$ and $6.6°C/mm$ on the die, respectively. This is shown in Table 2.5 which tabulates the die and package temperature corresponding to the heat map of Figure 2.18. In this part of my thesis, I use 3D-ICE thermal simulator [158, 159] in order to obtain the die temperature. As indicated in Figure 2.18, it is of great importance to design a thermosyphon that achieves the most homogeneous thermal profile with the smallest thermal hot spots on the evaporator side. More importantly, Figure 2.18b demonstrates that despite its efficiency in removing large heat fluxes, the thermosyphon is not capable of alleviating thermal gradients on the die without an adequate thermal-aware workload mapping strategy. Thus, to improve the lifetime reliability, it is necessary adapt the existing DTM approaches to effectively exploit the thermosyphon merits.

In this work, I address the workload-aware thermosyphon design from a system-level perspective. Afterwards, knowing the thermosyphon limitation and potential, I propose a thermal-aware workload mapping strategy to minimize hot spots and thermal gradients.

Table 2.5 – Temperature comparison: die vs. package

	$\theta_{max}(°C)$	$\theta_{avg}(°C)$	$\nabla\theta(°C/mm)$
Die	66.1	55.9	6.6
Package	46.4	42.9	0.5

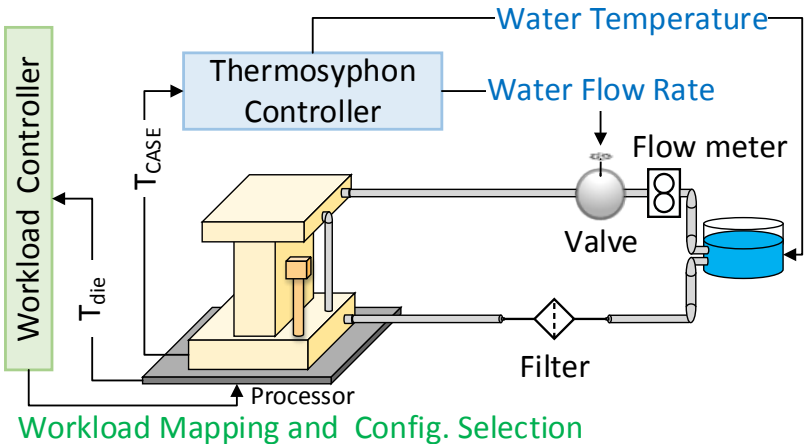


Figure 2.19 – Thermosyphon setup for DTM

A typical setup of thermosyphon on a processor is shown in Figure 2.19. The thermosyphon controller is in charge of dynamically tuning the water flow rate and temperature through a flow-meter, a valve, and a chiller to satisfy the thermal constraints. The workload controller, on the other hand, is in charge of adapting the system-level parameters, such as number of threads, thread allocation, and DVFS. To adjust the water temperature, one water cooling system (chiller) is used per rack in data centers. Therefore, all thermosyphons should work with the same water temperature. This limitation necessitates careful workload allocation to CPUs, as well as thermal-aware workload mapping on the cores to provide balanced temperature for all CPU packages.

2.6.1 Overview of System and Power Model

2.6.1.1 Server CPU Architecture and Floorplan

I consider the Xeon E5 v4 platform for my target workloads. This processor includes an 8-core Broadwell-EP CPU with dual clock frequency domains (Core and Uncore). The Uncore frequency scaling, as a new feature, allows the chip to dynamically set the frequency of the Uncore components which are proportional to the operating condition and workload. When this feature is disabled the Uncore frequency is fixed at the maximum supported frequency. The *likwid-setFrequencies* tool from the LIKWID tool suite [160, 161] provides a comfortable way to manually set the Uncore frequency. The memory subsystem comprises L1 instruction and data cache both of 32 KB, a private L2 of 256 KB, and a Last-Level Cache (LLC) of 25 MB. Figure 2.18c shows the die shot of the Broadwell CPU in 14nm process. The die area is 246 mm^2 and two cores are reserved for a deca-core CPU chip design.

2.6.1.2 Workload Configuration and QoS Requirement

I use the PARSEC 3.0 benchmark suite [162] featuring multi-threaded workloads. It provides a larger number of choices in terms of number of threads distributed among the physical cores to assess the target system. I evaluate the power consumption of PARSEC benchmarks as a function of the assigned number of cores (N_c), threads (N_t), and frequency scaling (f) that provide different thermal maps and profiles on the CPU die. Moreover, the threads that are allocated to the benchmark can have different mapping to the physical cores as each core has two physical threads. Hence, based on these characteristics, I consider different configurations per benchmark, i.e., (N_c, N_t, f). For instance, $(4, 4, f_{max})$ shows that the benchmark is running with the maximum frequency on four cores with one thread per core, while $(4, 8, f_{max})$ indicates all the threads of four cores are assigned to the benchmark.

QoS constraints are defined in terms of the maximum allowable degradation in workload execution time. In this work, I consider a QoS constraint of 1x, 2x, and 3x degradation [163], with respect to a baseline execution through a native 8-core CPU, with 16 threads, at maximum frequency for both core and Uncore. Figure 2.20 shows the normalized execution time for different configurations and workloads, with the QoS constraint set at 2x degradation. As shown in the figure, workloads meet QoS requirement for some configurations, providing different opportunities for workload mapping.

2.6.1.3 Power Model of Server Processor

I consider two main contributors to the overall power consumption of the CPU: 1) the core region composed of the cores and L1/L2 caches, and 2) Uncore components, which include LLC, memory controller, and IO subsystem. For power measurements, I use the Running Average Power Limit (RAPL) interface. I also leverage CPUPOWER and LIKWID utilities to set the core and Uncore frequency, respectively.

Core Power. Current servers can benefit from different idle power states (C-states). My target Intel processor is equipped with POLL, C1, C3, and C6 states [164]. POLL state represents the default working state of a core without any latency to resume execution. A higher C-state level shows a deeper sleep state with less power consumption but larger resuming latency. Table 2.6 shows the power measurements of the C-states for my target server.

When a core is in the C1 state, it is auto halt through maintaining the architectural state, cache information, and processing the requests for cache coherency. Some models also implement deeper state for C1 (i.e., C1E), reducing the voltage and frequency to enhance the power savings when all the cores stay on the C1 state. When a core goes to the C3 state, its L1 and L2 cache lines are flushed and are copied to the last-level cache (LLC). Moreover, the core clock stops and the architectural state is preserved with a significantly lower voltage. Finally, in C6 state, the core is power gated and the architectural state is stored in a SRAM in the Uncore.

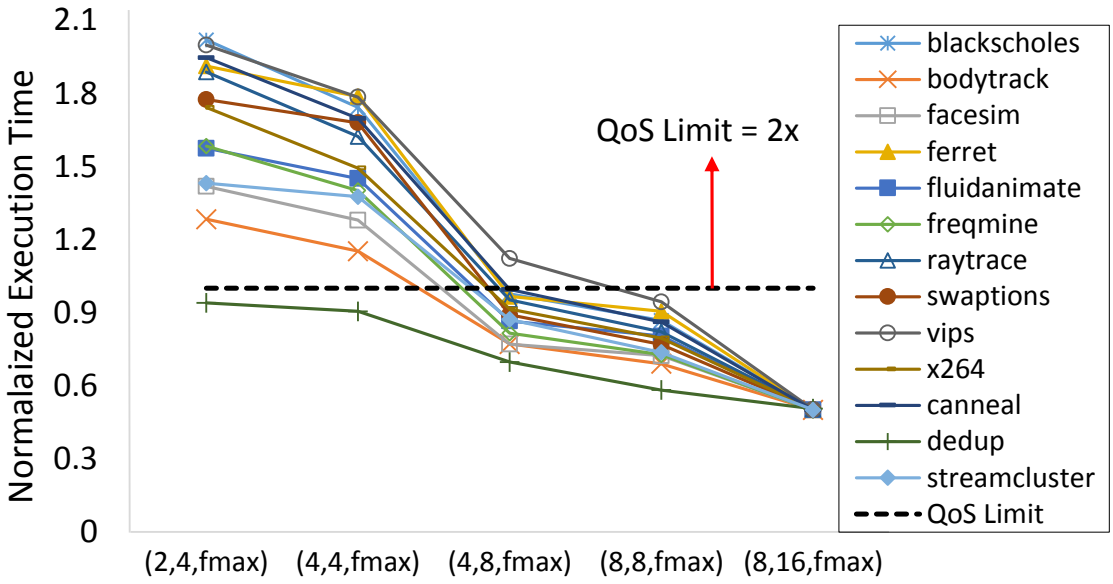


Figure 2.20 – Execution time normalized to QoS limit for some workload configurations @f_{max}

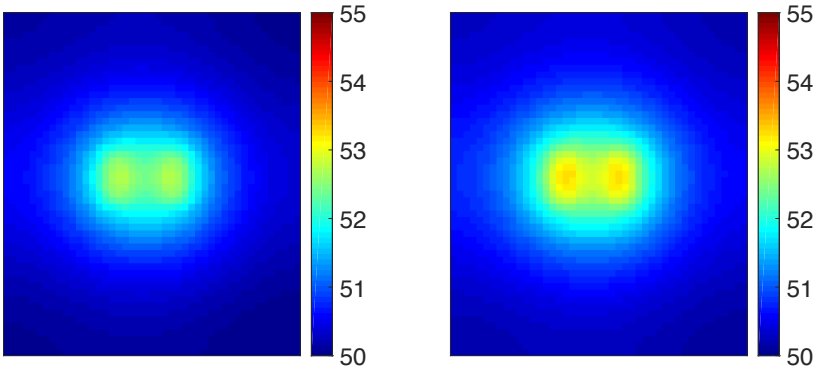
Table 2.6 – C-states power consumption of Xeon E5 v4 for all 8 cores

	Latency (s)	Power (W) @2.6GHz	Power (W) @2.9GHz	Power (W) @3.2GHz
POLL	0	27	32	40
C1	2	14	15	17
C1E	10	9	9	9

Nevertheless, cores are rarely in C3 and C6 states in today's servers in data centers due to the very large latency of re-enabling the core. Therefore, based on the different configurations for each benchmark, when a core is active but not running any workload, I change its state to the idle state with less latency (i.e., C1) to save more power and avoiding more QoS degradation.

Finally, for each benchmark, the dynamic power consumption is measured as a function of frequency for different configurations. In my case, to satisfy the defined QoS requirement, I consider three frequency levels: 2.6, 2.9, and 3.2 GHz.

Uncore Power. The static and dynamic LLC power model was extracted by measuring for a 25 MB capacity which is 2 W in the worst-case. I also empirically measured the memory controller and IO subsystem power consumption overhead of an Intel Xeon v4 CPU. This power consumption is split in two parts: (i) a constant component which accounts for the static, and (ii) a component proportional to the operating condition and uncore frequency. The constant part constitutes a 9 W overhead in all operating points, while the proportional one provides an 8 W variation from the minimum to maximum uncore frequency (i.e., 1.2 GHz-2.8 GHz).



(a) Package thermal map: Design 1 (b) Package thermal map: Design 2

Figure 2.21 – Package and die temperature for different orientations of thermosyphon on processor

Table 2.7 – Comparison between two different designs shown in Figure 2.21

Design		$\theta_{max}(°C)$	$\theta_{avg}(°C)$	$\nabla\theta(°C/mm)$
Package	#1	52.7	50.3	0.33
	#2	53.5	50.6	0.43
Die	#1	73.2	62.1	6.8
	#2	79.4	66.2	7.1

2.6.2 Design Optimization of Thermosyphon

In this subsection, I study the thermosyphon design optimization aspects from a system-level perspective including the orientation of inlet-outlet micro-channels on the evaporator, refrigerant type and its filling ratio, as well as water inlet temperature and its flow rate.

2.6.2.1 Thermosyphon Orientation

The thermosyphon orientation influences the position of evaporator inlet and outlet. Since the CPU die and package are not symmetric, the thermosyphon orientation also affects the number of micro-channels at evaporator side (assuming a constant channel width). Hence, for the same workload, different hot spots can appear depending on different orientations. Figure 2.21 shows two different designs of the thermosyphon for my target CPU when all cores are equally loaded. In the first design, the evaporator inlet and outlet are located on the east and the west, respectively, while in the second design, the evaporator inlet and outlet are located on the north and the south, respectively. Also, Figure 2.21 indicates that, for a fully utilized CPU, if less frequent hot spots with lower maximum temperature are required, the first design provides better results. Moreover, although the die is centered in the package, it creates larger hot spots on its left, since, as shown in Figure 2.18c, there is a dead area producing no power on the right side of the die. Consequently, due to the fact that evaporator inlet is always cooler than its outlet, an eastward flow of the refrigerant results in more homogeneous

thermal profile across the chip with smaller hot spots. Therefore, I choose the first design for the thermosyphon orientation to further alleviate spatial thermal gradients (STG).

2.6.2.2 Refrigerant and Filling Ratio

Refrigerant physical and thermal properties can considerably affect the thermosyphon efficiency in terms of heat removal. Since the type of refrigerant should be determined at design time, I consider the maximum workload (i.e., the worst case) and the T_{CASE_MAX} ($85°C$), which is the maximum temperature from the center of the heat spreader, as the thermal constraint for my design. Once the refrigerant is known, thermosyphon should be charged at a particular filling ratio, because it changes the thermosyphon efficiency in heat removal. For my design and target workload, I fill the thermosyphon with R236fa and a filling ratio of 55%.

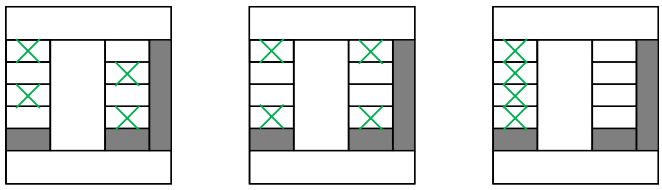
2.6.2.3 Water Temperature and Flow Rate

Although water temperature can be adjusted even at runtime, due to large response time, runtime adaptation of such a parameter may not be practical for workloads with critical deadlines. Water flow rate, however, can be adjusted dynamically due to its faster response time. The water flow rate and inlet water temperature affect the amount of electrical power consumed by the temperature control system. Hence, for an optimized thermosyphon design, the lowest flow rate and the highest inlet water temperature for which T_{CASE} remains below T_{CASE_MAX} for the worst case workload, should be the one used to maintain a low PUE. For my design and target workload, I consider a water flow rate of 7 Kg/h at $30°C$. The later is even higher than a typical room temperature, which helps reducing the cooling power consumption.

2.6.3 QoS- and Thermal-Aware Runtime Management

When meeting the QoS constraint requires the use of all CPU cores, there is no choice left for the workload scheduler to optimize the average and maximum temperature. However, as shown in Section 2.6.1, depending on the type of application and the user-defined QoS requirement, fewer cores than the maximum number of CPU cores ($N_{c,cpu}$) can be used to achieve larger power savings. Thus, when the number of required cores (N_c) is less than $N_{c,cpu}$, workload threads can be mapped optimally onto the cores to minimize the average temperature and/or number and values of hot spots according to the thermosyphon behavior.

Figure 2.22a-(c) shows three completely different workload mappings using four cores of an 8-core processor, with green crosses showing the occupied ones. Each of the scenarios shown illustrate a particular state-of-the-art workload mapping strategy. For instance, while Scenarios #1 and #2 are used for thermal-aware workload balancing [124], Scenario #3 shows a thermal mapping strategy proposed for liquid cooled SoC with micro-channels [141]. I consider two different cases. In the first case, I assume that all idle cores are set to POLL state, while in the second case, I consider C1 state. Table 2.8 shows the corresponding hot spot, the



(a) Scenario #1 (b) Scenario #2 (c) Scenario #3

Figure 2.22 – Die thermal profile vs. package thermal profile when using thermosyphon with non-optimized design and workload mapping strategy

Table 2.8 – Die temperature for three different scenarios of Figure 2.22

Die temperature	POLL			C1		
	Scen. #1	Scen. #2	Scen. #3	Scen. #1	Scen. #2	Scen. #3
$\theta_{max}(^{\circ}C)$	68.2	65.0	77.6	57.1	64.2	73.3
$\theta_{avg}(^{\circ}C)$	55.8	54.5	62.0	52.1	53.7	60.5
$\nabla\theta(^{\circ}C/mm)$	1.8	2.0	6.5	1.5	2.2	6.8

maximum spatial gradient, and average temperature of the die for these mappings.

When the CPU is in POLL state, the static power of idle cores is comparable to the dynamic power consumption of the active ones. In this case, scenario #2, which is a conventional thermal-aware workload balancing strategy (i.e., loading the CPU with the same workload starting from the corners), results in lower hot spots and average die temperature than the other two scenarios. While scenario #3 leads to the highest hot spot and average temperature, scenario #1 attains higher temperature but close to that of scenario #2. The reason lies in the fact that high power density between the active cores does not let the cores exchange heat properly.

When deeper C-states, such as C1, are used, scenario #1 is a better choice since there is not more than one hot spot (active core) on the same horizontal line. Therefore, evaporator micro-channels are more efficient in removing heat from the chip. Nonetheless, this is not the case when idle cores are set at POLL state, because idle cores still consume large amount of static power. Therefore, depending on the C-states used for idle cores (determined by the maximum latency tolerable for the application) different optimal mappings can be attained to alleviate thermal hot spots. The same discussion is also valid when 5 cores are used. Nonetheless, when using more than 5 cores, a more straightforward approach should be adopted. In this case, threads should be mapped to the cores starting from the corners, then mapped to the rest recalling that always fewer active cores on the same horizontal line are desirable. Following the discussion above, an optimal mapping that minimizes the number and magnitude of hot spots can be obtained for each particular configuration (number of cores to be used).

Algorithm 2.7 presents the proposed configuration selection and workload mapping, which minimizes power consumption and further reduces thermal hot spots and spatial gradients while meeting the QoS requirement. I consider a set of applications $\boldsymbol{A}$ whose threads should be

Algorithm 2.7: Configuration selection and thread mapping

Input : $\boldsymbol{A} = \{A_1, \ldots, A_n\}$, $\boldsymbol{D} = \{d_1, \ldots, d_n\}$, $N_{c,cpu}$, $\boldsymbol{N_c} = \{1, \ldots, N_{c,cpu}\}$, $\boldsymbol{N_t} = \{1, 2\}$, $\boldsymbol{f} = \{f_{min}, \ldots, f_{max}\}$, $\boldsymbol{S} = \{s_{core}, s_{llc}, s_{uncore}, s_{memcnt}\}$, $\boldsymbol{QoS} = \{q_1, \ldots, q_n\}$

Output: $CPU_i \xleftarrow{map} A_i@C^{opt}(N_c, N_t, f)$

; // Mapping A_i with optimal configuration to server i

forall $i \in \boldsymbol{A}$ **do**
  forall $j \in \boldsymbol{N_c},\ k \in \boldsymbol{N_t},\ l \in \boldsymbol{f}$ **do**
    $\boldsymbol{P_i} \leftarrow \mathbf{P}(\boldsymbol{N_c^j}, \boldsymbol{N_t^k}, \boldsymbol{f^l})$
    $\boldsymbol{Q_i} \leftarrow \mathbf{Q}(N_c^j, N_t^k, f^l)$
  $\boldsymbol{P_{sort}} \leftarrow \mathbf{Sort}^{asc}(\boldsymbol{P_i})$
  $C^{opt} \leftarrow$ Find the first configuration in $\boldsymbol{P_{sort}}$ *s.t.* $\boldsymbol{Q_i} > q_i$
  $\boldsymbol{H_i} \leftarrow \mathbf{H}(\boldsymbol{P_{sort}^{opt}}, S)$
  $CPU_i \leftarrow \mathbf{Map}(\boldsymbol{H_i}, d_i, A_i@C^{opt}(N_c, N_t, f))$

mapped to the multi-core CPU with $N_{c,cpu}$ cores. Each application requires a minimum QoS q_i and can tolerate up to d_i seconds delay for idle cores on the CPU. The goal is to find the number of cores (N_c), threads per core (N_t), and frequency (f) for which the power consumption is minimized and q_i is satisfied. The power consumption and the QoS resulted from each configuration j are known and stored in $\boldsymbol{P_i}$ and $\boldsymbol{Q_i}$ vectors, respectively, by $\mathbf{P}(N_c^j, N_t^k, f^l)$ and $\mathbf{Q}(N_c^j, N_t^k, f^l)$ obtained from profiling the application. I sort $\boldsymbol{P_i}$ in an ascending order and I select the first configuration for which $\boldsymbol{Q_i}$ is larger than q_i. Finally, knowing the area ($\boldsymbol{S}$) and the power consumption of each component, the heat generated by different components is estimated. Afterwards, based on the delay that each application can tolerate and the per-component estimated heat flux ($\boldsymbol{H_i}$), I follow the mapping strategy discussed earlier to minimize the value and number of hot spots.

Finally, during runtime, I increase water flow rate only if a thermal emergency (i.e., $T_{CASE} \geq T_{CASE\_MAX}$) occurs and lowering the frequency violates the QoS requirement.

2.6.4 Experimental Results and Discussion

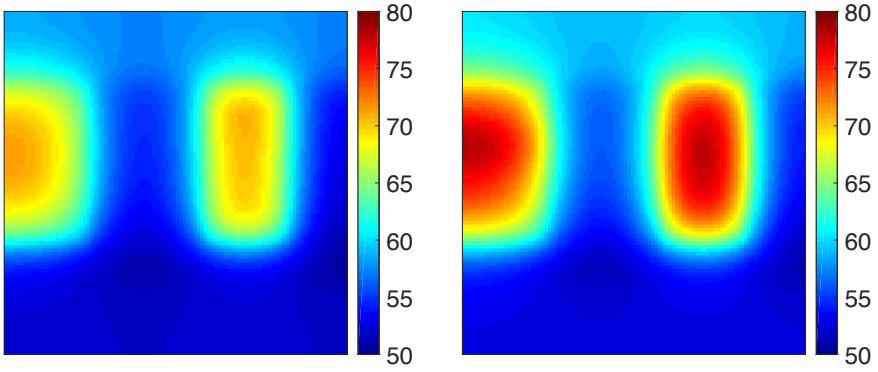
I use the simulation framework of Seuret et al. [20]. In order to provide a fair comparison, I compare my thermosyphon design and thermal-aware workload mapping with the design of Seuret et al. [20] equipped with a configuration selection strategy [165] and two different workload mapping policies proposed by Coskun et al. [124] and Sabry et al. [141]. The latter is aimed at inter-layer liquid-cooled MPSoCs.

2.6.4.1 Thermal Hot Spots and Spatial Gradients

Table 2.9 shows the thermal hot spots and spatial gradients, on average, achieved by my proposed approach against the state of the art, for different QoS requirements. When no QoS

Table 2.9 – Thermal hot spot and spatial gradients for different QoS requirements

Approach	QoS	Die		Package	
		θ_{max}	$\nabla\theta_{max}$	θ_{max}	$\nabla\theta_{max}$
Proposed	*1x*	78.3	0.90	52.1	0.24
	2x	72.2	1.03	49.0	0.24
	3x	68.4	1.25	46.3	0.28
[20]+[165]+[124]	*1x*	83.0	0.95	52.5	0.27
	2x	79.5	1.33	51.4	0.30
	3x	77.8	1.60	49.1	0.36
[20]+[165]+[141]	*1x*	83.0	0.95	52.5	0.27
	2x	80.5	1.8	50.4	0.32
	3x	79.1	2.3	49.1	0.43



(a) (b)

Figure 2.23 – Thermal map of the die obtained from a) proposed approach and, b) state of the art

degradation is allowed, all approaches run the workload with f_{max} and maximum number of available cores and threads. Consequently, the improvement in hot spot and spatial gradient reduction comes only from the thermosyphon design. This comparison highlights the importance of a workload-aware thermosyphon design. In particular, when 2x or 3x degradation from the reference QoS is allowed, my proposed thermal-aware workload mapping outperforms that of Coskun et al. [124] and further diminishes thermal hot spots and spatial gradients. The main improvement is obtained if 3x degradation is allowed, because my workload mapping strategy is able to map the workload based on the C-states for idle cores.

The key idea of the workload scheduling policy proposed by Sabry et al. [141] is to map the workload first to the cores closer to the liquid inlet. However, such a policy is not suitable when a two-phase thermosyphon is used. Although my discussion in Section 2.6.2.1 reveals the importance of inlet and outlet location with respect to the die floorplan, since the die and micro-channels are separated by means of the package and the heat spreader, the amount of heat removal at the inlet compared to the outlet does not motivate mapping the workload starting from those closer to the inlet. In particular, the 3^{rd} scenario of Figure 2.22 clearly

discourages such a mapping strategy. Hence, the work of Sabry et al. [141], on average, provides the worst results.

Finally, Figure 2.23 depicts the die thermal map obtained through my design and workload mapping policy versus state-of-the-art approaches. This figure shows one sample thermal map obtained under 2x QoS degradation. While this hot spot is 78.2°C for the state of the art, my work achieves 71.5°C.

2.6.4.2 Cooling Power

To achieve the same hot-spot temperature without my proposed thermal-aware workload mapping for the same water flow rate, a water temperature of 20°C is required, instead of 30°C. Moreover, the temperature difference between the inlet ($T_{in,w}$) and outlet ($T_{out,w}$) water for my approach is 6°C, and 11°C without my approach. The higher this difference is, the more increases the chiller burden. This is due to the fact that the chiller has to cool down the outlet water to the needed inlet water temperature. This implies that the proposed approach reduces cooling power up to 45%, assuming that the electrical power (W) required to change the temperature of L liter water ΔT K is:

$$P = \dot{V} \times \rho \times C_w \times \Delta T, \tag{2.20}$$

where $\dot{V}$ is volumetric flow rate (Liter/s), ρ represents density (Kg/Liter), and C_w shows water specific heat ($J.Kg^{-1}.K^{-1}$).

Moreover, this previous discussion is pessimistic as I consider that only the chiller is in charge of cooling down the $T_{out,w}$ and the outside air temperature cannot be used. Considering a room temperature of 25°C, the required $T_{in,w}$ in my case is higher. Hence, in real scenarios, the chiller would need to consume much less power to cool down the water (even close to zero). Nonetheless, working with $T_{in,w}$ = 20°C, which is lower than the room temperature, requires the chiller to significantly consume more power.

2.7 Summary

Performance, power consumption, and thermal profile constitute the first-order design objectives in run-time management of multiprocessor systems. Heuristics, among all conventional methodologies, are widely used in multi-objective management of multiprocessor systems thanks to their low complexity and execution overhead. Traditional DTM and DPM heuristics, however, do not address lifetime reliability optimization with respect to thermal stress, including temperature gradients in time and space, as well as thermal cycling. Moreover, these heuristics often discard the undeniable role of cooling systems in thermal management of high-performance power-hungry processors. In particular, the most recent two-phase liquid cooling, thermosyphon, had neither been studied nor considered in DTM literature.

Therefore, in this chapter of my thesis, I have first proposed a thermal stress-aware power and thermal management framework (TheSPoT) for multiprocessor systems. TheSPoT had the objective of increasing the performance while considering the power constraint as well as thermal stress constraints including the spatial temperature gradient, temporal temperature gradient, and thermal cycles. TheSPoT provides a multi-level DTM approach through consolidation, deconsolidation and DVFS. I have considered a heuristic consolidation/deconsolidation scheme, while I have proposed two different schemes for DVFS: a heuristic and an optimal one. Comparison to a state-of-the-art heuristic DTM [88] revealed that lifetime reliability, in terms of MTTF, could improve by 47% through TheSPoT framework. In addition, I have shown that TheSPoT is able to outperform the conventional DTM approaches even more in presence of higher workload variations. Moreover, I have assessed the scalability of TheSPoT by simulating a multiprocessor system with different number of cores. The results showed that heuristic TheSPoT scales well with number of cores with only 4% performance overhead considering a 100ms interval for DVFS, whereas the run-time overhead of the optimal TheSPoT increases super-linearly with the number of cores, making it impractical. In fact, TheSPoT with the optimal DVFS is impractical when the number of cores increases since for a 16-core MPSoC its runtime overhead increases to approximately 80ms, showing 80% overhead. With even larger decision intervals, such as 1 second, the runtime overhead of the optimal approach is still too large to be applicable for real-life application with hard deadlines. Even worse, this overhead, as aforementioned, scales super-linearly with the number of cores. Since today's servers usually have more number of cores (32, 64, 128, etc.) than the maximum considered in this chapter (i.e., 16), the optimal solution is clearly not applicable. On the contrary, the proposed rule-based heuristic solution is simple and fast enough to make it a promising approach for many-core systems.

Second, I have studied and assessed a recently manufactured prototype of a micro-scale thermosyphon to enhance thermal control of multiprocessor systems. In particular, my study revealed that despite its higher efficiency compared to other single-phase and two-phase liquid cooling systems, micro-scale thermosyphon requires customized design with respect to the workload and processor. Furthermore, such a design must be accompanied by adequate thermal-aware workload mapping strategies and run-time management. Thus, I have proposed a heuristic workload- and cooling-aware DTM that together with an optimally-designed thermosyphon could reduce thermal hot spots and spatial gradients by up to $10^{\circ}C$, and 45%, respectively, while requiring at least 45% less cooling power for the chiller, compared to state-of-the-art DTM and thermosyphon solutions.

Although in this chapter I addressed new challenges in multiprocessor systems through heuristics, all those problems contained conventional runtime design space. In such problems, the intuition- and knowledge based heuristics were successfully applicable. However, in presence of much larger design space, especially when irregular patterns exist in workloads, or a more general solution is desirable for big data problems, heuristics fall short. In these scenarios, machine learning is a more promising solution and is applied in various problems [166–171].

3 Machine Learning for Runtime Management of Time-Variant Workloads

3.1 Introduction

Multiprocessor systems play a major role in achieving the required performance through parallel processing. However, the increased number of processors requires more complex software and hardware, which ultimately pose additional challenges for multi-objective management of these systems. In particular, while DVFS makes DPM and performance control feasible, numerous frequencies available per-core in modern processors considerably enlarge the design space. Moreover, with the increased number of cores, it is more challenging to find the number of optimal application threads and cores to provide power-efficient performance optimization.

Traditionally, heuristics have been used for multi-objective run-time management of multiprocessor systems thanks to their simplicity and low-overhead execution. However, heuristics are less sensitive to dynamic problems and fall short when facing large design space. With modern multiprocessor systems, the design space already has become too large for heuristics to handle it successfully. The shift towards new application models and services, such as real-time streaming, contributes to the existing challenge in heuristic multi-objective management. In particular, the workload variation posed by these applications does not simply come from different stages and functions with various characteristics. Rather, the input plays a more significant role in workload variation. This variation usually does not present any particular pattern and, thus, time-predictability becomes more challenging than ever [172].

In contrast to heuristics, Machine Learning (ML) is more flexible in coping with large and dynamic design spaces. In fact, supervised and unsupervised learning through a vast number of different algorithms, such as linear regression, logistic regression, K-Means, K-Nearest Neighbor (KNN), Principal Component Analysis (PCA), Random Forest, Support Vector Machine, etc., bring about a powerful tool to deal with multi-objective management of multiprocessor systems, especially, when large and seemingly patternless workloads accompany numerous available design parameters.

In order to tackle this multi-objective management of multiprocessor systems, however, ML requires informative data from both application (workload) and the underlying platform. Modern processors are equipped with hardware performance counters that enable non-intrusive monitoring of the processors [173], providing important information about the performance and workload behavior.

Hence, in this chapter, I address multi-objective management of multiprocessor systems for high-variation workloads through an ML-based framework, using hardware events. My proposed framework is able to predict the processor workload and estimate the expected performance for different system-level parameters, including operating frequencies and number of active threads and cores. To assess the efficacy of the proposed ML-based framework in dealing with large workload variation, I consider real-time HEVC encoder with various inputs. In addition, to conduct a fair comparison, I propose an application-specific workload-aware heuristic to determine the number of running threads and operating frequency of the active cores for HEVC encoder.

3.2 Case-Study Application: High Efficiency Video Coding (HEVC)

Video streaming services are expected to account for 80% of global traffic by 2019 [174], with services such as Netflix and YouTube accounting for over 50% of downstream traffic [175]. Due to the great variety of devices accessing media content as well as the users' demand for higher-quality videos, encoding has become a key application in current High Performance Computing (HPC). To satisfy the emerging large video resolutions, the High Efficiency Video Coding (HEVC) standard provides twice the compression as of its predecessors [176] while maintaining the same video quality, at the price of increasing the encoder complexity by several times [177, 178]. This increased computational complexity makes real-time power-efficient video encoding very challenging. Moreover, video content changes within a video imply workload variations which requires adaptive runtime management.

In this section, after a brief review on the HEVC standard, the reference software and my study setup, I first scrutinize the impact of contents of videos with respect to various metrics, including video quality (PSNR, i.e, Peak Signal-to-Noise Ratio), video compression (bitrate), performance (in terms of encoding time), power consumption, and temperature of target processor, in addition to several hardware event counts concerning with LLC and L2 accesses. Then, I overview two different methods for parallel processing of HEVC Encoders.

3.2.1 HEVC Standard, Reference Software, and Study Setup

Figure 3.1 shows a simplified HEVC encoder block diagram. The raw video frames, as inputs to the encoder, first split into Coding Units (CUs) of the size decided by the encoder, such as 16×16, 32×32, and 64×64. There are two different predictions that can be used for coding these CUs. In intra-picture prediction, spatial locality of pixels is used to predict and code the

Table 3.1 – Test sequences

Test Sequence	Resolution	Frame Rate (Hz)	Frame Count	Target Bitrate
SVT04a	1280x720	50	500	4
OldTownCross	1280x720	50	500	4
Tennis	1920x1080	24	240	4
Cactus	1920x1080	50	500	6
Calendar	1920x1080	50	500	6
BQTerrace	1920x1080	60	600	6
OldTownCross_HighRes	3840x2160	50	500	8

adjacent pixels within a frame. In inter-picture prediction, temporal locality is used such that an already coded frame (or a part of a frame) is used to further encode an incoming frame. To realize this entity, HEVC encoder has a builtin decoder composed of deblocking filter, Sample Adaptive Offset (SAO) filter, and a buffer. Transformation and Quantization blocks then provide coding of the prediction residuals, and uniform reconstruction quantization (URQ), receptively. Finally, Context Adaptive Binary Arithmetic Coding (CABAC) is used for entropy coding [178].

The reference software for HEVC is called HM (HEVC Test Model) [179]. This reference implementation provides a baseline for the researchers to check the correctness of their HEVC software implementation, but lacks any sort of optimizations in terms of performance targeting specific architectures (e.g., vector instruction support). However, a number of high-performance HEVC implementations are available, such as Kvazaar [180] and x265[1].

For the following study, I consider the reference software and use the default *Main Intra* configuration for intra-picture prediction. I use several well-known test sequences shown in Table 3.1 for this study, covering a wide variety of frame-to-frame motion, and texture within a frame. While *Cactus* and *BQTerrace* are among the test sequences introduced by JCT-VC [181], the rest are being frequently used in benchmarking video coding applications [182–184]. Moreover, I perform the experiments on an Intel S2600GZ server running CentOS 6.5. The server includes a 6-core SandyBridge-EP processor. The server comes with 32KB instruction and data L1, 256KB private L2, and a 15MB shared L3. I use Intel's Running Average Power Limit (RAPL) to collect power measurements of CPU and DRAM, while Intelligent Platform Management Interface (IPMI) is used to gather CPU temperature sensor measurements, based on the same methodology used by Salinas-Hilburg et al. [128].

3.2.2 Content and Workload Variation

Besides the inherent and exclusive features of each video type, such as frame resolution, bit depth, etc., the contents of a video play a major role in the obtained encoding time, quality, compression, power consumption, and even peak temperature, resulted from a specific set of encoding parameters.

Figure 3.2 shows the encoding time, bitrate, and PSNR obtained when encoding seven different

[1] http://x265.org/

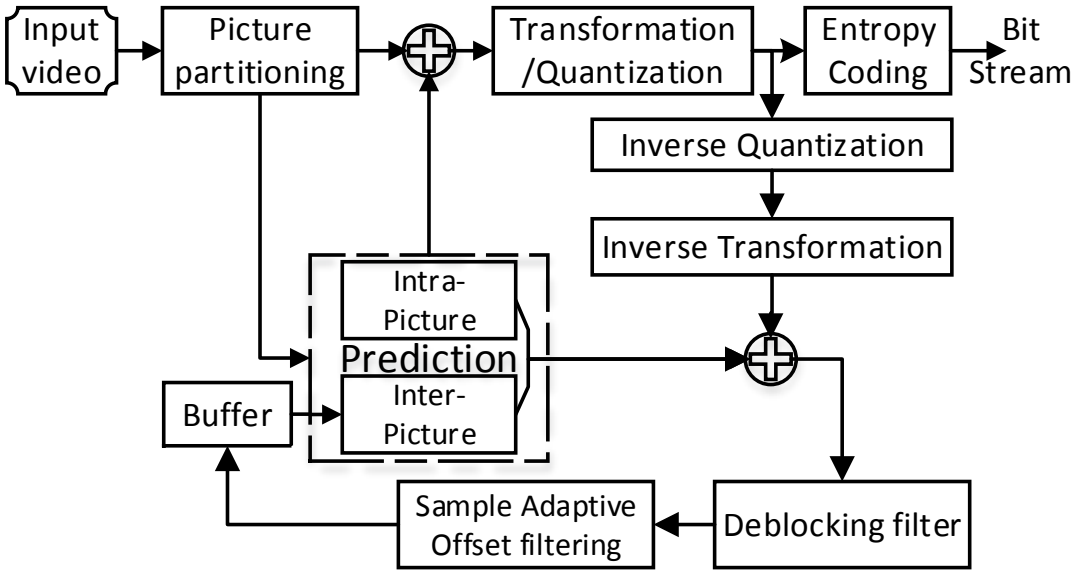


Figure 3.1 – Simplified HEVC encoder block diagram

8-bit test sequences shown in Table 3.1. As shown in Figure 3.2, the obtained metrics not only do differ considerably from one video resolution to another (*OldTownCross* vs. *OldTownCross_HighRes*), but they also differ vastly from one video to another of the same resolution (Cactus vs. Calendar), and even, within a video (all videos). For instance, the output bitrate of *BGTerrace* changes by 50% within only 240 frames. Also, the output PSNR of *Tennis* changes by 6dB in only 10 seconds due to the content variations. These significant content variations, however, do not change the encoding time as noticeably as they do the output PSNR and bitrate. The reason lies in the fact that the current reference model of HEVC [179] is not optimized for performance. Thus, a considerable amount of the execution time is spent on loading frames, coding unit preparations, etc., rather than on the encoding task. Yet, these content variations account for 5% to 10% of the variations in execution time. All in all, in a multi-stream video coding platform, the underlying processor hosts different videos with different contents which implies highly variable workloads.

Such variations in video contents impact the encoding time directly through affecting the memory sub-systems. In fact, while the CPU is fully utilized throughout the video encoding (tested on an Intel S2600GZ server with a E5-2620 CPU), the number of accesses/misses to/from L2 cache and Last Level Cache (LLC) change vastly as the contents vary from one frame to another. This behavior is consistent across different platforms (Intel Xeon X5650 and E5-2620, and AMD Opteron 6272 and 6300). Figure 3.3 shows the number of accesses to L2 and LLC, and misses from LLC for test sequence *Tennis* every second collected by Oprofile [185]. Comparing the number of accesses to L2 and LLC with the encoding time per frame for the *Tennis* sequence clearly shows the importance of video contents on memory hierarchy and, hence, on the encoding time as the trend thoroughly coincides with this metric (the same trend is observed for misses from L2). As Figure 3.3a and Figure 3.3b show, there is a correlation

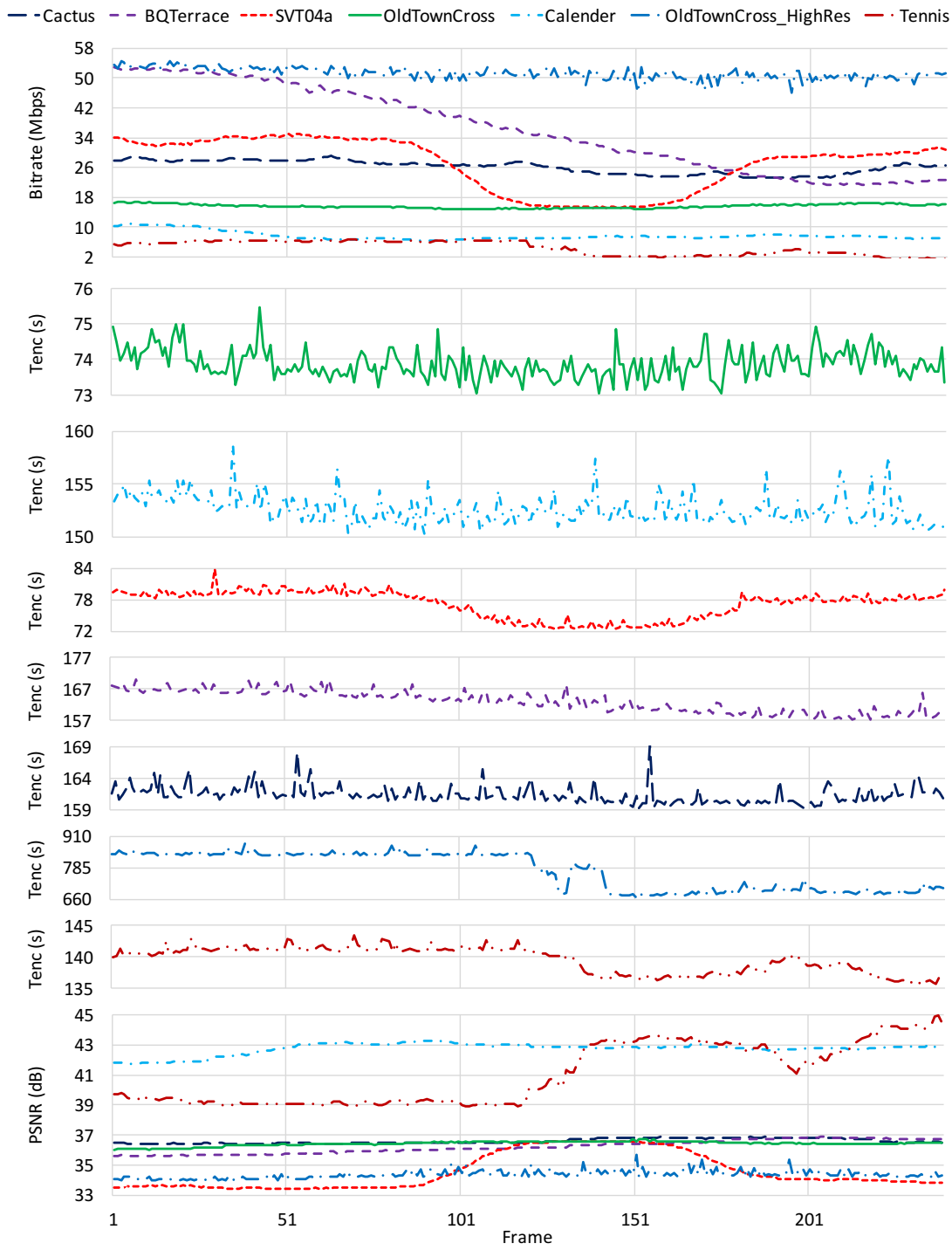


Figure 3.2 – Per-frame bitrate, encoding time (T_{enc}), and PSNR for seven test sequences with *Main Intra* configuration

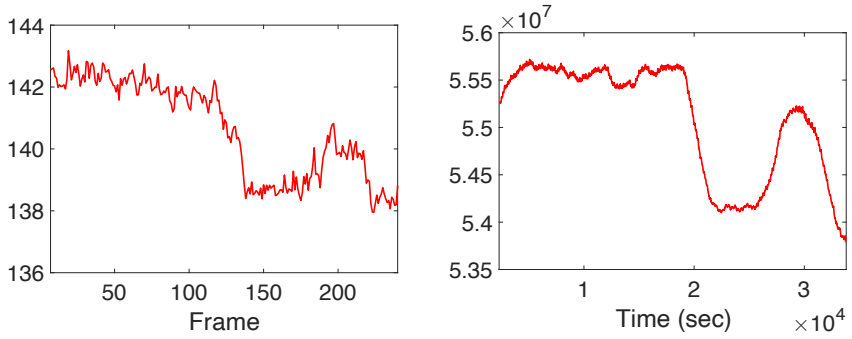
between the number of L2 accesses and encoding time. In this sense, when the number of L2 (or LLC) accesses is small, lower encoding time/frame is observed. Although such a pattern

seems to be absent in the trend of LLC misses in Figure 3.3c, this figure contains important information. In fact, LLC misses consist of two traces. The first one (bottom left) includes the spikes (in the range of $1 \sim 2 \times 10^6$) which represent misses due to loading a new frame from the main memory. The number of these spikes are approximately 240, i.e., the number of frames in the *Tennis* sequence. The second trace (bottom right) appears to be around 2×10^4 and follows the same variations as that of the encoding time/frame. The similarity between content variations (reflected in encoding time) and memory sub-system events demonstrates the importance of frame-by-frame power, performance, and thermal management. Figure 3.4 shows the power and temperature variations with respect to content variations in the input test sequence *Tennis*. Again, the power and thermal variations are aligned with the content variations.

3.2.3 Workload Parallelization for Multimedia Applications

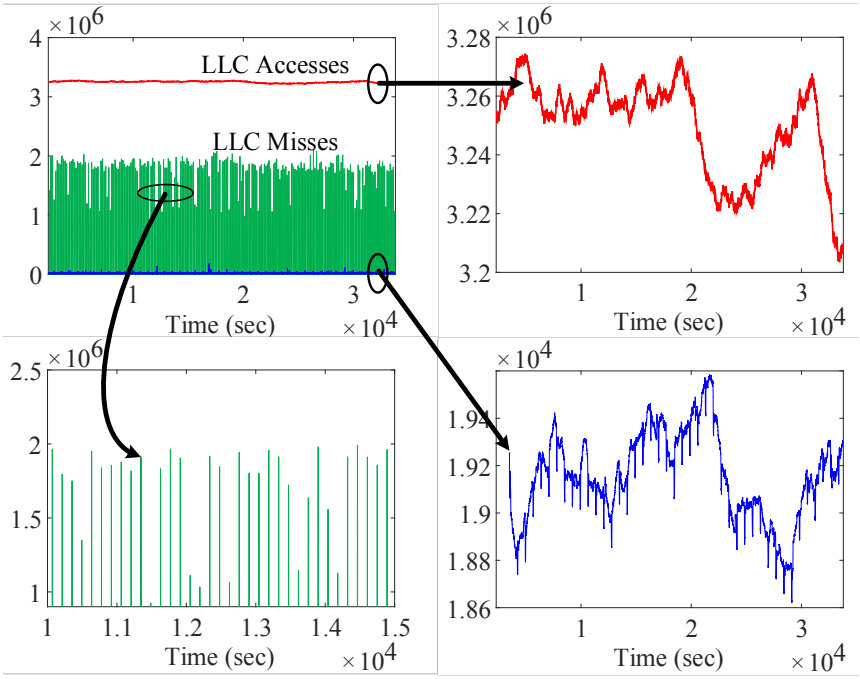
Several works have targeted the parallelization of transcoding for multimedia applications. Indeed, video frames can be clustered as group of pictures (GOPs) and then be independently processed providing workload parallelization [186]. At frame-level, workload parallelization is enabled through two different schemes by HEVC standard: Wavefront Parallel Processing (WPP) [187] and tiling. While wavefront dependencies prevent all partitions from being processed concurrently, tiles can be regarded as independent threads providing more parallelization and are leveraged by a few works [188, 189]. In tiling, the encoder can either rely on initial uniform tiles, or can dynamically change the tile formation through the nonuniform tiling property. Unlike WPP, high degrees of parallelization through tiling may ultimately lead to quality and compression degradation since the spatial localities of the neighboring pixels in two adjacent tiles cannot be used for more efficient coding.

Figure 3.5 depicts the impact of parallelization on power consumption, throughput, video quality and compression along with different values of Quantization Parameter (QP). As shown in Figure 3.5a, as the number of parallel threads increases, higher throughput can be achieved at the cost of more power consumption. However, the increase of frames per second, as the throughput metric, finally saturates, meaning that with higher number of threads there maybe no specific task assigned to a thread, thus, it does not help increasing the throughput. Figure 3.5b shows how parallelization affects PSNR and bitrate. This figure indicates that although parallelization does not change the output PSNR, the output video requires higher bandwidth due to less compression provided. The reason lies in the fact that by parallelization, spatial locality between some adjacent pixels within a frame are discarded. Since HM [179] does not support parallelization, for Figure 3.5 I use Kvazaar HEVC encoder.



(a) Encoding time/frame (sec)

(b) Number of L2 Accesses per second



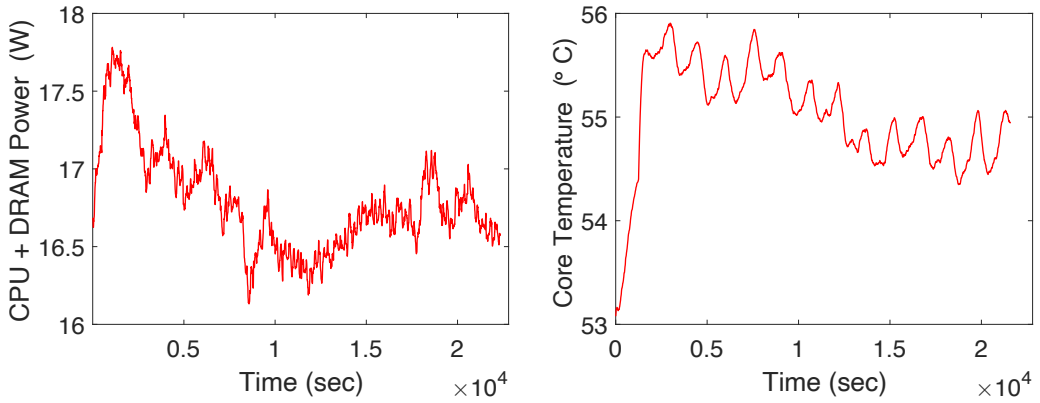
(c) Number of LLC Accesses and Misses per second

Figure 3.3 – Number of accesses to L2, accesses to LLC, misses from LLC and encoding time/frame for *Tennis*

3.3 Literature Review

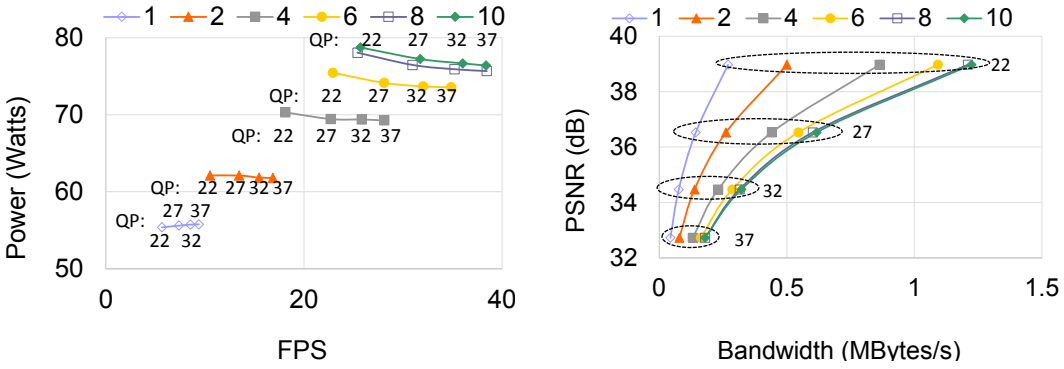
3.3.1 Machine Learning for Workload and Performance Prediction

Workload prediction methodologies are quite rich in literature, especially for cloud-based applications [167–170]. However, since these applications usually show workload variations on an hourly, daily, or even weekly basis, many of these existing works use ARIMA model and its derivatives [190], which is not necessarily compatible with highly time-varying non-cloud applications.



(a) Power consumption (b) Core Temperature

Figure 3.4 – Content-based power and temperature variation



(a) (b)

Figure 3.5 – RD-curves, power, and throughput with respect to number of threads: 1, 2, 4, 6, 8, and 10 and QP values: 22, 27, 32, and 37 while encoding a 1080p-video at 3.2GHz using Kvazaar with the *ultrafast* configuration.

On the contrary, machine learning models, such as clustering, classification, and regression have been widely used to predict the future workload for different purposes, from intelligent vehicles [191] to video decoding [192]. In this context, Jung and Pedram [109] leverage supervised learning for performance prediction and use it in a power management scheme. Wu et al. [193] train a Neural Network (NN) to predict GPU performance under different workloads. Thermal prediction and lifetime reliability estimation are addressed through NNs by Jayaseelan and Mitra [194] and Yamamoto and Ababei [195], respectively. Reza et al. [196] predict the power consumption and thermal profile of Network-on-Chips through NNs. Linear Regression is used by Inchoon Yeo et al. [197] and Li et al. [198] for predicting the temperature and execution time, respectively. Workload classification is addressed through Multinomial Logistic Regression by Das et al. [199]. Akay and Abasıkeleş [200] develop a Support Vector Regression (SVR) model for predicting the performance measures (such as average network latency) of multiprocessor systems with distributed shared memory. A Tree-based design space

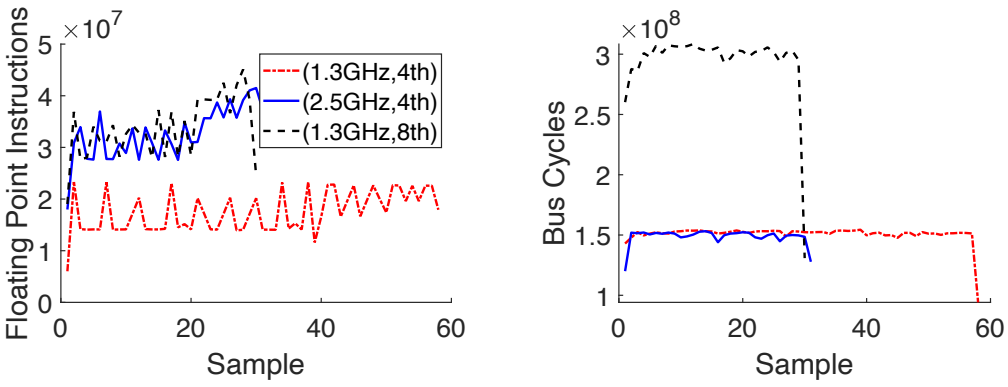
exploration is proposed by Sinaei et al. [201] for run-time resource allocation on MPSoCs. Despite such a variety of machine learning applications, workload prediction and performance estimation of highly time-varying applications on multiprocessor systems with respect to different system-level parameters has not been adequately addressed.

3.3.2 Hardware Event-Based Management of Multiprocessor Systems

The correlation of hardware events with performance and power consumption of the application has been traditionally used in literature for performance and power analysis, prediction, and tuning [202]. The first step for this purpose is to extract the most relevant and accurate information from these counters. In this context, CounterMiner [203] proposes a machine learning-based methodology to improve the accuracy and efficiency of using hardware events. Li et al. [204] use linear regression and SVM for thread characterization. However, no run-time adaptation of system-level parameters is provided.

Although performance counters have been used for different application domains such as hardware security improvement [205] and malware detection [206], the most common one is power and performance prediction and management [207]. For instance, Borghesi et al. [208] leverage hardware events as a part of the input to their predictive model for power consumption of HPC systems. This work, however, does not address power and performance estimation for different system configurations.

Nonetheless, there are a few works that use hardware events for power and performance estimation across different system configurations. In particular, Singh et al. [209] develop a power estimation and thread scheduling scheme using a piece-wise model based on multiple linear regression. Nonetheless, this work is limited to thread scheduling as their only design parameter. Moreover, the scalability of the proposed strategy remains in question due to the use of linear regression [193]. Besides, Curtis-Maury et al. [210] propose an online strategy for power and performance management of multi-threaded applications using hardware event-based prediction. In their work, however, operating frequency as one of the main design space scaling factors has been ignored. More importantly, it is based on linear regression which is known to be limited in modeling large design spaces and runtime hardware changes. Finally, in contrast to the work of Singh et al. [209] and Curtis-Maury et al. [210], Wu et al. [193] provide low-overhead scalable solution by training a neural network with performance counter values to predict power consumption and performance of different applications at different GPU configurations. Authors, however, train their neural network with the overall behavior of complete input kernels rather than with fine-grained input samples. Such an approach can lead to wrong decisions for dynamic configuration selection, as it ignores rapid workload variations within a kernel, which is very typical in recent time-varying applications, such as video streaming.



(a) (b)

Figure 3.6 – Number of (a) floating point instructions and, (b) bus cycles, every 400 ms, under 3 system configurations: (frequency (GHz), number of threads)

3.4 Proposed ML-Based Framework for Power and Performance Management

In this section, I detail the proposed ML-based framework for workload prediction and throughput estimation highly variant workloads under different system configurations, in terms of operating frequency and number of threads, given a set of performance counters as the training dataset. The proposed ML framework incorporates classical ML algorithms of clustering, classification, and regression.

3.4.1 Problem Definition

With increasing the number of available processors in multiprocessor systems along with wider range of operating frequencies the design space for power and performance management has become too large. In order to tackle such a large design space, modern processors are equipped with hardware performance counters that enable non-intrusive performance monitoring of the processors [173]. Indeed, these counters can provide important information about workload behavior, which can be used in workload prediction. This workload prediction, if accomplished accurately enough, can be effectively deployed for power management and performance control of time-varying applications. To this end, hardware event-based simulators and analytic models have been traditionally used for workload assessment and prediction. However, due to the need for fast and dynamic changes of system configurations for efficient proactive power and performance management, these approaches are not feasible in practice [193].

Moreover, in multi-core platforms, information extracted from the performance counters does not purely indicate the workload behavior, as the collected data are also affected by the specific system configuration (i.e., the number of processors and the operating frequency). Therefore,

workload prediction is not straightforward since with any change in the system configuration, performance counters may imply a new workload for the same input of the application, even though the application as well as its input data remain the same. As an example, Figure 3.6 shows the trace of two performance counters while running a time-varying application (from the beginning to the end) under three different configurations on a multi-core server. For this specific application, the number of *Bus Cycles* are mainly affected by the number of threads rather than the core frequency. Furthermore, as expected, system configurations of (2.5,4) and (1.3,8) can provide almost the same average performance, with the latter having twice *Bus Cycles*. Therefore, in this case, both measures are required to properly adapt the system configuration.

Nonetheless, when changing the system configuration to the other, the already-gathered hardware events cannot be directly used to predict the next values through conventional workload prediction techniques, such as Auto-Regressive Integrated Moving Average (ARIMA) model and its derivatives [211]. In contrast, machine learning (ML)-based approaches represent a promising solution, as they can directly learn from the data rather than counting on rule-based programming. More importantly, ML-based approaches are known to provide scalable solutions for large design spaces with low-overhead inference time, which makes them appropriate candidates for power and performance management of multi-core platforms.

3.4.2 Hardware Event-Based ML Framework

Figure 3.7 shows an overview of my proposed ML-based framework. The goal of this framework is to estimate the future throughput of an application with iterative structure (as shown in Figure 3.8) for different system configurations (i.e., core frequency and number of cores), given a set of hardware performance counters ($PC_j(t)$) at time t, obtained from a particular system configuration (j). Since at time t, the only available information is $PC_j(t)$, it is not straightforward to estimate the future throughput for all other configurations directly. Therefore, first, $PC_j(t)$ is used to specify the current workload ($WL_j(t)$) of the application through clustering and classification algorithms. This workload accounts for input and system configuration. Then, $WL_j(t)$, in conjunction with the previous workload of the application under the same configuration ($WL_j(t-1)$), is used to predict the next workload for all available system configurations, $Config_k(t+1)$. Next, the predicted workload ($WL'_k(t+1)$) is employed to estimate the future throughput of the application for each configuration.

When using performance counters, the first step is to determine the most relevant counters that carry useful information about system and application. Therefore, I first explain how to choose these performance counters. Second, I explain in detail the steps I take for the per-configuration throughput estimation depicted in Figure 3.7.

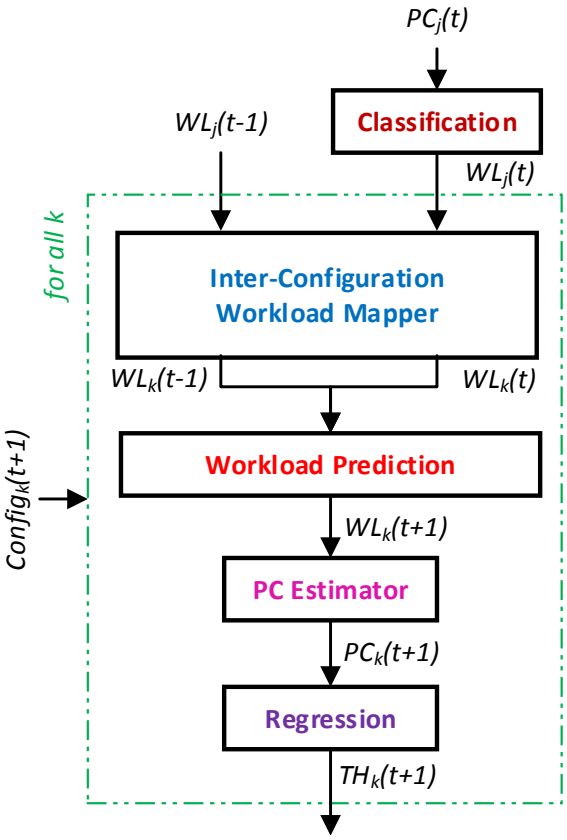


Figure 3.7 – Proposed ML-based framework

3.4.3 Counter Selection

Although there can be hundreds of counters available on a modern machine [212] (e.g., more than 500 in an Intel Xeon CPU E5-2667), the number of counters that can be simultaneously collected without time multiplexing are limited. Also, precision of the measured counters degrades when more counters are to be read. Moreover, using more counters as the system and application features for estimating the future throughput does not necessarily improve the accuracy. Finally, using redundant features may ultimately lead to more runtime overhead due to the increased complexity. Therefore, it is vital to select counters that carry the most relevant information regarding the target system and application behavior.

For each application in control, starting from the main hardware counters introduced in the literature [210], I use the Pearson Correlation [213] to determine those that contain the most relevant information according to the throughput. Although there are a lot of different approaches for feature reduction in literature, such as PCA, I found low-overhead Pearson Correlation sufficient for my purpose. The selected counters are the ones with larger correlation coefficient with respect to the throughput, which are less correlated with each other to avoid redundant information.

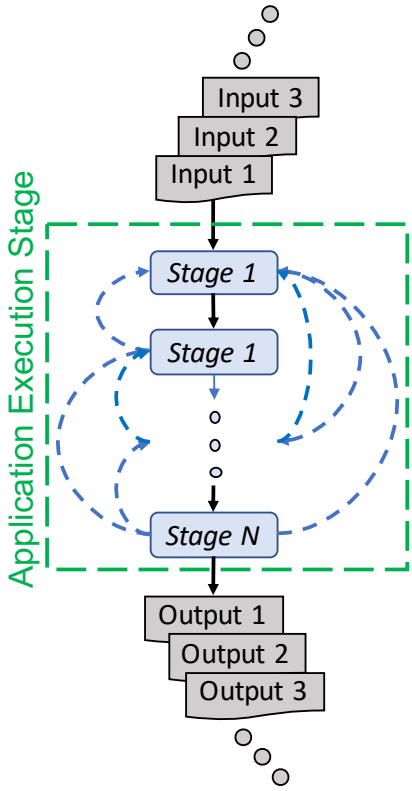


Figure 3.8 – Example of an application with iterative structure

3.4.4 Workload Clustering and Classification

For a single application, the values of the performance counters and the application throughput vary within different system configurations. Moreover, for the same system configuration, these values can change due to the workload variations at different stages of the application. Since with the same application and input size, each system configuration may lead to a different set of performance counters and throughput for a particular stage of the whole application, the workload type under process should be studied per configuration. Therefore, I propose to qualify WL_k based on PC_k for each $Config_k$.

With a constant performance counter sampling rate, a different number of workload types can be observed for each configuration. For configurations with higher frequency and larger number of cores, the sampling rate can be comparable to the execution time of one iteration of the application, while with lower frequency and smaller number of cores, it can be much higher than the execution time. Hence, for the latter, $PC(t)$ may indicate the difference between different stages of the application while for the former, $PC(t)$ may only show the difference between different inputs.

Imagine a video encoding application that outputs one frame per second under a particular configuration. In such a case, with a sampling rate of 10 Hz, 10 sets of performance counters exist for a single frame. On the contrary, if with a different configuration the application

outputs 10 frames per second, there would be one set of performance counter samples per frame. Thus, for the latter, the difference between performance counter samples are mostly due to different input data (here, frames) processed at each iteration of the complete application run.

In order to qualify the workload type per configuration, I propose to use the $Kmeans++$ clustering algorithm [214]. I consider the Silhouette [215] criterion to find the optimal number of clusters per configuration. Thereafter, considering the Random Forest (RF) [216] with 50 weak learners as the classification algorithm, I train an ensemble of bagged classification trees using the performance counter samples as the input features and the workload type (WL) obtained from the clustering as the output labels. While these two phases are performed offline, the obtained classification model is employed at runtime to qualify $WL_j(t)$ from $PC_j(t)$. RF suits well my purpose since I need to deal with a multi-class classification problem rather than a binary one. Moreover, RF is suitable when coping with a mixture of numerical and categorical features which necessitates using the features (performance counters) on their own various scales.

3.4.5 Inter-Configuration Workload Matching and Prediction

Once $WL_j(t)$ is recognized from $PC_j(t)$, for each available configuration ($Config_k(t+1)$) the next workload should be predicted. I propose to create a discrete-time, finite-state Markov chain [217] from workload transitions for each configuration using the same training data as in the workload classification. Thus, the next workload can be predicted considering the probability of transitioning to a new workload according to an already-observed chain of transitions. During runtime, I keep updating the transition probabilities to enhance the next workload prediction within a single configuration. The length of this chain depends on (i) the nature of the application, (ii) the frequency of the workload changes within a certain configuration, and (iii) the performance counter sampling frequency. For my purpose, I found 10 Hz sampling rate and a chain length of two sufficient to account for workload variations without adding extra load to the system.

The $Kmeans++$ algorithm finds the optimal centroid seeds by first choosing seeds from a random observation of the data set. Consequently, as I run the algorithm for each configuration, the outcome clusters of one configuration are not in the same order as the clusters obtained for another configuration. For instance, the first cluster of the j^{th} configuration may correspond to any workload type in the k^{th} configuration, where $j \neq k$. Moreover, as aforementioned, there can be different numbers of clusters (i.e., workload labels) for each configuration. Therefore, to predict $WL_k(t+1)$ for $Config_k(t+1)$ using the Markov chain, I need to know about its current workload $WL_k(t)$. Since at runtime, only $WL_j(t)$ is known, where $j \neq k$, it is necessary to define the translation to the workload labels of other configurations. Thus, in order to find the best matching workload in configuration k with N_{wl_k} workload types

(number of labels), I use the squared Euclidean distance metric (D_{Euc^2}), as follows:

$$wl_k^* \leftarrow \underset{wl_k^{'}}{\textbf{argmin}}(D_{Euc^2}(PC_{centroid}(WL_j(t)), PC_{centroid}(wl_k^{'})) \\ wl_k^{'} = \{1, ..., N_{wl_k}\} \tag{3.1}$$

where wl_k^* is the workload type of configuration k which is well matched to the current workload of configuration j, $WL_j(t)$. In this formulation, $PC_{centroid}(WL_j(t))$ shows the performance counter values related to cluster centroid of the workload at $Config_j(t)$, and $wl_k^{'}$ represents all known workloads for configuration k. I propose to use Euclidean distance metric as it is originally used by $Kmeans++$ algorithm to find the workload clusters. Finally, I find the inter-configuration workload matches offline, and refer to them as a look-up table during runtime.

3.4.6 Performance Counter Estimator and Regression Model

Once the next workload for each configuration is predicted ($WL_k^{'}(t+1)$), the future throughput can be estimated through a regression model. As discussed earlier, each workload label corresponds to a cluster centroid composed of a set of performance counters. As shown in Section 3.4.3, I can find a set of performance counters correlated with the throughput with smaller correlation coefficients among them. These counters are suitable candidates to provide a regression model to estimate the throughput. Hence, I leverage RF [216] with 50 weak learners for the regression algorithm, as it is able to sufficiently handle non-linear dependencies. I train an ensemble of bagged regression trees with this set of performance counters as the input features, and the corresponding throughput, as the response variable.

Nevertheless, in the real-time execution, only a unique set of performance counter values can be extracted from $WL_k^{'}(t+1)$. Using these values into the trained regression model may cause a large prediction error, as the actual performance counter values may not be close enough to the centroids. Therefore, in order to compensate this error, I propose to use the ratio of actual performance counter values at time t over the estimated ones at time t by the centroids as a scaling factor for $PC_k(t+1)$, as follows:

$$PC_k^{'}(t+1) = PC_{centroid}(WL_k(t+1)) \odot \big(PC_j(t) \oslash PC_{centroid}(WL_j(t))\big), \tag{3.2}$$

where $PC_{centroid}(WL_k(t+1))$ is the performance counter values of the centroid that corresponds to the workload at time $t+1$ for the k^{th} configuration, and $\odot$ and $\oslash$ represent the element-wise product and division operations, respectively. Finally, the estimated performance counters, $PC_k^{'}(t+1)$, can be used in the trained regression model to predict the next throughput, $TH_k(t+1)$, for all available future configurations, $Config_k(t+1)$.

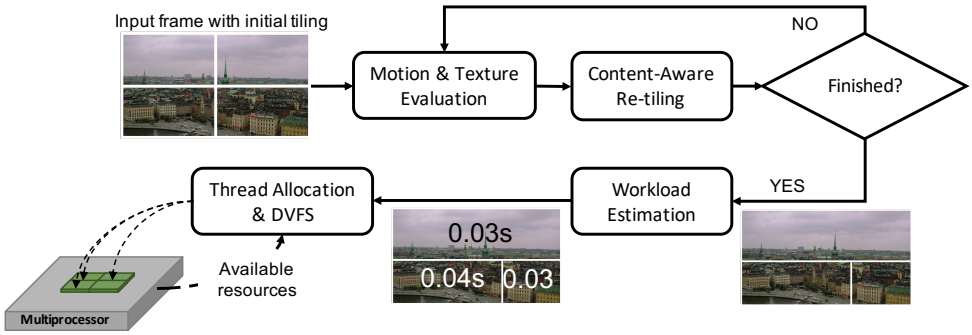


Figure 3.9 – Proposed heuristic workload estimation, thread allocation, and DVFS for HEVC encoding

3.5 Proposed Heuristic Workload-Aware Management for HEVC Encoders

In this section, I address power-efficient HEVC encoding through heuristics to provide a fair comparison between ML and heuristics in Section 3.7. The main objective of heuristic approach is to maximize the throughput while minimizing the power consumption and meeting the throughput constraints denoted as FPS.

HEVC is equipped with powerful tools that enable multi-threaded processing in different levels, from splitting a single frame into multiple threads, to processing several different frames at the same time. Such features make multiprocessor systems promising to achieve real-time video encoding through workload parallelization. In this section, I consider a frame rate of 24 FPS as the goal throughput, which is the most common one in the state of the art. However, the functionality of my proposed heuristic does not depend on the target throughput. This requirement means that every $\frac{1}{24} \approx 0.04$ seconds, one frame should be encoded.

Figure 3.9 illustrates the proposed content-aware heuristic approach for enabling real-time HEVC encoding. For each frame in a video the motion and texture are evaluated (Subsection 3.5.1) for every initial tile (from the last processed frame, or simply a uniform tiling if no frame has been processed yet). Based on the knowledge of the motion and texture of different areas of the frame, I perform the re-tiling of the frame (Subsection 3.5.2). The predefined minimum tile size and the maximum number of tiles within a frame ensure fast ending of this phase. The workload corresponding to each tile and its encoding parameters is estimated via a look-up table (LUT) which is dynamically updated throughout the encoding process (Subsection 3.5.3.1). Finally, based on the estimated workload and availability of the resources (i.e., available processors and operating frequencies of the platform) I allocate each tile to an available resource and set the operating frequency to enable real-time energy efficient video encoding (Subsection 3.5.3.2).

3.5.1 Motion and Texture Evaluation

The diversity in luma samples (i.e., achromatic portion of the image) in a tile as well as the amount of motion affect the encoding complexity, thus, the run-time execution. Therefore, efficiently tiling based on its contents helps achieving more efficient thread allocation and scheduling.

The first step for efficient tiling is to quantify the motion and texture of different partitions of the frame. Hence, starting from the latest tiling of the previous frame, I evaluate the diversity of the texture and the presence of motion. Since this evaluation must be fast enough to avoid any computational overhead, I use the coefficient of variation (CV) defined as the ratio of the standard deviation to the mean. Then, I classify the tile based on a predefined threshold of the texture, as follows:

$$T = \begin{cases} low & CV \leq T_{th,l} \\ medium & T_{th,l} < CV \leq T_{th,h} \\ high & CV > T_{th,h} \end{cases} \tag{3.3}$$

In order to obtain a low-overhead measure of the motion in a tile, I propose a pixel-to-pixel comparison of a limited number of pixels including four corners, the center, and the one with the maximum value. In particular, these pixels within each current tile are compared to the corresponding pixels from the subsequent frame, as follows:

$$M = \alpha \sum_{i=1}^{4} x_i + \beta c + \gamma m, \tag{3.4}$$

where x_i, c, and m are booleans for pixel comparisons respectively at the 4 corners, center and maximum point. When pixels are equal, booleans are zero. In this formulation, α, β, and γ denote the importance of the comparison for different coordinates of the tile.I experimentally choose 1, 3, and 3 for α, β, and γ, respectively. Finally, I define the motion threshold ($M_{th} = 3$) based on which a tile is regarded as high- or low-motion, as follows:

$$Motion = \begin{cases} low & M < M_{th} \\ high & M \geq M_{th} \end{cases} \tag{3.5}$$

3.5.2 Content-Aware Re-tiling

Based on the existing texture and motion in different parts of a frame, I split it into different number of tiles. I propose to start re-tiling from the initial tile in the top left of the frame. If the motion and texture of the tile is *low*, I increase the tile size by 25% more pixels first in the width and then in the height. This value was experimentally found and represents a trade-off between optimal tile size and the time it takes to find it. This procedure continues until the texture or the motion is not low anymore, and I keep the latest tile coordinates. Due to these

new coordinates, the coordinates of one or more tiles can change accordingly. Then, if there is still at least one tile to the right of the first one, the same procedure is applied to find the new coordinates of this tile. If there is no tile left at the right, the next first tile at the left of the frame is evaluated according to the same procedure. Figure 3.9 shows that starting with a frame uniformly split to four tiles, the proposed content-aware re-tiling outputs three tiles within the frame.

3.5.3 Workload Estimation, Thread Allocation and DVFS

3.5.3.1 Workload Estimation

In order to perform the best thread allocation and DVFS to maximize the number of users that can be sustained by the target multiprocessor system, while meeting the required framerate and encoding efficiency, I use an LUT-based approach. In fact, the LUT-based approach is very suitable for this context due to the nature of the proposed re-tiling approach, which includes a limited number of different attainable tile structures within a frame.

In my proposed framework, I store the histogram of the CPU time in the LUT and keep updating it throughout the whole video encoding. I use the stored histograms to estimate the workload for robust thread allocation and DVFS.

3.5.3.2 Thread Allocation and DVFS

To reduce the execution time of the proposed content-aware re-tiling, I consider GOP of size 8 and apply the re-tiling and thread allocation strategies only once for a GOP. However, DVFS on modern multiprocessor systems is sufficiently low-overhead and can be applied on a per-frame basis.

To increase the parallelism of the encoding, I apply the GOP-level parallelization besides the tile-level one. In this scenario, multiple GOPs (N_g) of the same video are released at the same time. Therefore, one instance of re-tiling is applied to the first frame of each GOP at the same time.

Algorithm 3.1 shows the proposed heuristic DVFS and thread allocation, where N_g is the total number of GOPs within a video, N_c shows the total number of cores, N^i_{th} is the total number of threads in the first frame of the i^{th} GOP, $\boldsymbol{T^i_f}$ represents the estimated CPU time for all threads of the first frame of the i^{th} GOP, and $\boldsymbol{F}$ is the set of available frequencies on the target multiprocessor system. I first determine the minimum number of cores needed for each GOP (N^i_{core}) based on its threads CPU time (Line 1) given the maximum frequency level ($T^i_{f_{max}}$) at each time slot (i.e., $1/FPS$). Then, I select G GOPs until it reaches the total number of cores (N_c). Afterwards, I try to allocate all the threads of the selected GOPs (G) to the cores, while the cores CPU time used by the threads does not exceed the time slot period ($1/FPS$).

Algorithm 3.1: Thread allocation and DVFS

Input : $N_g, N_c, \boldsymbol{N_{thr}} = \{N^1_{thr}, ..., N^{N_g}_{thr}\}, \boldsymbol{T^i_f} = \{T^i_{f,1}, ..., T^i_{f,N^i_{thr}}\}, \boldsymbol{F} = \{f_1, ..., f_{max}\}, FPS$

Output : Serving maximum number of frames

1 $N^i_{core} \leftarrow (\Sigma_{j=1}^{N^i_{thr}} T^i_{fmax,j}).FPS, \quad i \in \{1,2,...,N_u\}$

2 $G \leftarrow$ Find the maximum of g GOPs w.r.t. $\Sigma_{k=1}^{g} N^k_{core} \leq N_c$

3 **for** $i = 1 : N^U_{thr}$ **do** //Thread allocation

4 **for** $k = 1 : N^g_{core}$ **do**

5 **if** $max_k(load_k) > \frac{1}{FPS}$ **then**

6 $Cap \leftarrow \frac{1}{FPS}$

7 **else**

8 $Cap \leftarrow max_k(Load_k)$

9 $Dist^j_k \leftarrow \|Cap - (Load_k + T^{g_j}_{fmax,j})\|$

10 $ID_c \leftarrow$ Find Core with $min_k(Dist^j)$

11 $Load_{ID_C} \leftarrow Load_{ID_C} + T^{g_j}_{fmax,j}$

12 **for** $k = 1 : N_c$ **do** //DVFS for energy efficiency

13 **if** $Load_k \leq \frac{1}{FPS}$ **then**

14 Set $min(F)$ to core k for slack time $(\frac{1}{FPS} - Load_k)$

15 $Load_k \leftarrow 0$

16 **else**

17 Set f_{max} for the whole $\frac{1}{FPS}$

18 $Load_k \leftarrow Load_k - \frac{1}{FPS}$

For this purpose, I first select one thread from the pool of selected GOPs and try to find the best core for it (Lines 3-12). N^G_{thr} denotes the total number of threads available for G. Then, I determine a dynamic cap (Cap) finding the maximum load over all cores (i.e, CPU time already allocated to threads, which cannot be above $1/FPS$ (Lines 5-8). Based on this cap, I compute the euclidean distance (Line 9) between the cap and the load of the k^{th} core ($Load_k$) with the consideration of allocating the thread to this core ($Dist^j_k$). Finally, I allocate the j^{th} thread to the core whose CPU time nearly reaches the cap (i.e., $min(Dist^j)$), and update its load (Lines 10 and 11). This shows that this thread can make the CPU time of the target core more utilized than the other cores.

After allocation, for energy efficiency (Lines 12-18), I check the load of each core whether it has the slack time (idle time). If it does, I set the minimum frequency ($min(F)$) to the core for the rest of the time. Otherwise, I keep the maximum frequency and shift the remaining load (CPU time) to the next interval.

Algorithm 3.2: Power Minimization

Input : $TH^{'}(Config_k(t+1))$
Output: $Config_{opt}(t+1)$

1 $\boldsymbol{Config_{candidate}} \leftarrow$ **Find**$(Config_k(t+1))$
2 $\quad s.t. \quad TH^{'}(Config_k(t+1)) \geq TH_{const}$
3 $Config_{opt}(t+1) \leftarrow \underset{Config_{k'}}{\textbf{argmin}}(P_{RF}(Config_{k'}),$
$\quad Config_{k'} \in \boldsymbol{Config_{candidate}}$

3.6 Experimental Setup

I perform my experiments on a server equipped with two Intel Xeon E5 v4 CPUs providing a total number of 32 threads. The server is able to perform DVFS with 16 available frequencies ranging from 1.2 GHZ to 3.6 GHz. Therefore, there are $32 \times 16 = 512$ different system configurations available for running each application. I use the Linux Perf tool [212] to collect the performance counters data. In order to provide accurate data about the application behavior, I use Docker containers to isolate the application from any possible background tasks. As a case study, I consider Kvazaar [180] HEVC encoder.

I compare the proposed ML-based framework with 3 different solutions: proposed heuristic, a neural network-based approach [193], and a load balancing method for video applications [189]. The proposed heuristic and the work of Khan et al. [189] directly include power minimization, whereas the proposed ML-based framework and the work of Wu et al. [193] provide performance and power estimation. In order to evaluate the accuracy of these estimates, I consider a power minimization problem in which the system configuration changes dynamically to satisfy the required throughput while providing the minimum power consumption. For this purpose, I provide a general power model. In particular, I run PARSEC 3.0 benchmark suit [162] with native input size under all available configurations and measure the average power consumption of package and DRAM. Then, I train an ensemble of bagged regression trees of RF with 50 weak learners. I consider the system configurations as the input features and the measured power consumption as the response variable. I test my trained ensemble with measured power consumption of HEVC encoder under different configurations. My model provides an average error of 5.3% with standard deviation of 3.9%.

Using the obtained power model, I apply Algorithm 3.2 to the output of my proposed framework and that of Wu et al. [193]. The input to this algorithm is the estimated throughput, $TH^{'}(Config_k(t+1))$, for all available configurations. Here, the goal is to find a configuration by which the minimum power consumption can be achieved, while satisfying the throughout constraint, TH_{const}. In this algorithm, $\boldsymbol{Config_{candidate}}$ shows a vector of all configurations which are predicted to achieve a throughput larger than the constraint (Lines 1 and 2). I use the obtained power model from the RF algorithm (P_{RF}) to estimate the power consumption of each candidate configuration. Finally, the one that achieves the minimum power consumption is used to run the application for the next time slot (Line 3).

Table 3.2 – Pearson correlation matrix of different performance counters and HEVC encoder throughput (FPS)

	branch miss	bus cycle	L2 miss	floating point instr.	total Instr.	FPS
branch miss	1.0	0.0	0.0	0.6	0.0	0.1
bus cycle	0.0	1.0	0.0	0.7	0.9	0.8
L2 miss	0.0	0.0	1.0	0.0	0.0	0.0
floating point instr.	0.6	0.7	0.0	1.0	0.8	0.8
total Instr.	0.0	0.9	0.0	0.8	1.0	0.9
FPS	0.1	0.8	0.0	0.8	0.9	1.0

In case of the ML-based framework, depending on the target frame rate (24 FPS, in my case), some of the system configurations can never satisfy the real-time HEVC encoder requirements, while some others may result in excessive throughput which increases power consumption and require larger buffers. Furthermore, if trained with all spare configurations that can never provide a satisfactory power consumption with the target throughput, the accuracy of the workload prediction and throughput estimation framework can degrade considerably. Hence, for my target case-study application, based on my observation when collecting training data for my framework, I narrow down the available configurations to 192. This selection is obtained by assuming two threshold values for the output framerate. In particular, I considered configurations that are able to reach at least 18 FPS once in the whole execution and do not provide an average framerate of larger than 35 FPS. I experimentally found this boundary around the target framerate (24 FPS) sufficient to account for any content variation within a video and between different videos.

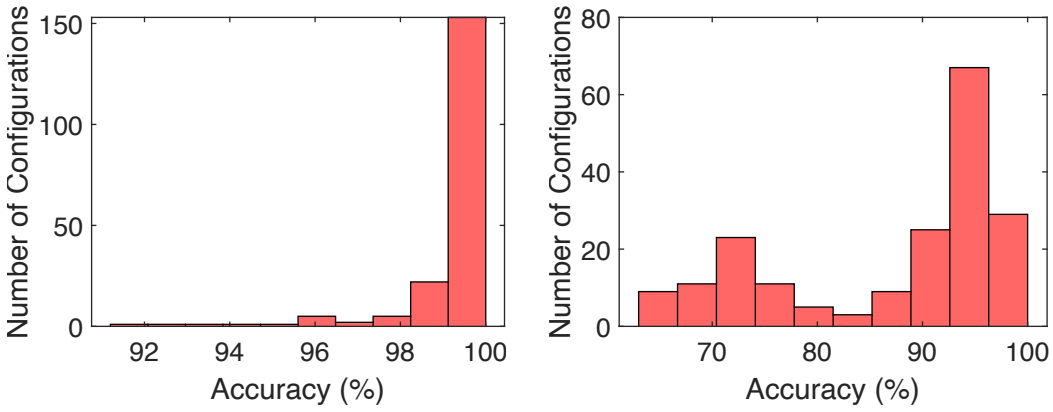
3.7 Experimental Results and Discussion

In this section, I first evaluate the accuracy and efficiency of each block in the proposed ML-based framework with respect to the case-study application. Afterwards, I provide a comparison with NN-based approach [193] to assess the throughput estimation accuracy when dealing with a power minimization problem under QoS constraint. Finally, power consumption and throughput of my ML-based approach is compared with two other heuristics.

3.7.1 Performance and Accuracy of Proposed ML Framework

3.7.1.1 Counter Selection

Table 3.2 lists the Pearson correlation matrix with one digit precision for the main counters and the throughput, FPS. On one hand, the total number of retired instructions (*total Instr.*) has the highest correlation with the output FPS. The next two hardware events with the largest correlation coefficient with respect to the throughput are, respectively, floating point arithmetic retired instructions (*floating point Instr.*) and bus cycles. On the other hand, *total Instr.* is highly correlated with the other two events, while *bus cycles* and *floating point Instr.* are less correlated. Hence, I consider only *floating point Instr.* and *bus cycles* which can well model the FPS through regression. On the contrary, for the workload clustering and classification, I



(a) Classification (b) Prediction

Figure 3.10 – Accuracy of the proposed (a) classification and (b) prediction

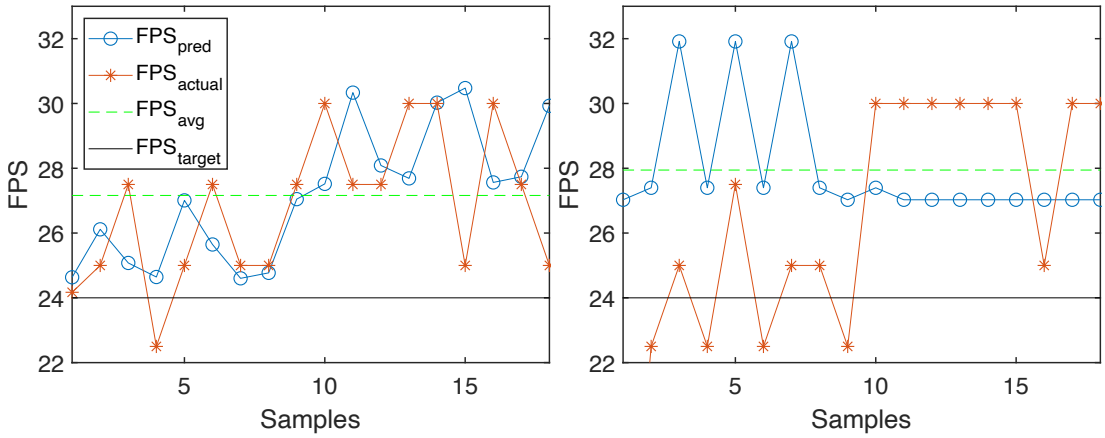
consider all the listed counters, except for *total Instr.*, as each of the rest can demonstrate a particular aspect of the workload.

3.7.1.2 Per-Configuration Workload Clustering and Classification

As discussed in Section 3.4.4, for each configuration, a different optimal number of clusters (workload labels) can be found. For my case-study application, I found 128 configurations with 2, 59 configurations with 3, and 5 configurations with 4 workload labels. My study shows that, within those configurations providing higher throughput more workload types can be found. Subsequently, Figure 3.10a shows the histogram of the workload classification accuracy for the system configuration based on the observed event counters. On average, my classification achieves an accuracy of higher than 99%.

3.7.1.3 Per-Configuration Workload Prediction

Figure 3.10b shows the histogram of the workload prediction accuracy for the system configuration. While the average accuracy is 87%, lower accuracy values (e.g., around 70%) are mainly due to the fact that for some configurations with very large throughput, there is a rapid traverse from one content to a completely different one (and hence, different workload). Nonetheless, in practice, since these particular configurations often provide very large unnecessary throughput and, thus, higher power consumption, they are not proper candidates when solving the power minimization problem. Hence, as described in Section 3.7.2, the overall results are only slightly affected.



(a) Proposed ML framework

(b) State of the art [193]

Figure 3.11 – Throughput prediction and actual throughput

3.7.1.4 Throughput Regression

The regression accuracy is computed across all configurations in terms of Root Mean Square Error (RMSE). In my case of study, the RMSE is 0.54 FPS.

3.7.2 Throughput Estimation Accuracy and Evaluation of Power Minimization

Figure 3.11a shows the throughput estimation obtained from my ML framework, the actual, average, and the target throughput for a particular runtime window. Figure 3.11b shows the same plots as in Figure 3.11a for the same test video, but obtained from SoA[193]. As shown in these figures, my framework is able to provide more accurate throughput estimation. For the test case shown in these figures, my framework is able to predict the output framerate with only 6.8% error, while this error for the work of Wu et al. [193] is 20.5%. Furthermore, in the window shown, my framework violates the target throughput only once, while SoA [193] shows four QoS violations.

As shown in Figure 3.11a, my framework is able to provide very accurate estimates for some samples, while for some others the estimation error is larger (e.g., 25% as for sample 15). The main source of such an error is the *Workload Prediction* rather than the *Workload Classification*, since as shown in Section 3.7.1.2, the latter is highly accurate. In particular, such a large error can occur if the selected configuration for the next time slot is the one for which the average workload prediction accuracy is not sufficiently high. For this specific sample, the selected configuration is ($1.5GHz$, 10 threads) with a workload prediction accuracy of 70%, on average.

In addition, I perform a K-fold cross-validation analysis. In contrast to my proposed framework, the work of [193] is more sensitive to the training data. The reason lies in the fact that Wu et al. [193] train the neural network with the average throughput of each configuration. Consequently, it performs more accurately if the test data (i.e., video in my case) comes with

Table 3.3 – Average throughput estimation error and QoS violations

	Min.(%)	Max.(%)	Mean(%)	STD(%)	#QoS Violations
Proposed ML	0.0	26.1	7.5	6.1	18
SoA [193]	14.4	30.0	20.1	5.5	61

Table 3.4 – Comparison to the state-of-the-arts (SoA)

Method	Avg. FPS/Stream	#Violation/Stream	Normalized Power
Proposed ML	24.8	18	0.67
SoA (NN)[193]	25.8	61	0.79
Proposed Heuristic	25.5	73	0.76
SoA (LB)[189]	27.4	81	1

almost the same variation range compared to the training data. However, this is not the case for streaming and other highly time-varying applications. Table 3.3 shows the average throughput estimation error and QoS violations. On average, for different folds considered in my sensitivity analysis, my framework reduces the violations of the target throughput by 3.4x compared to SoA [193].

3.7.3 Comparison to Heuristics

Table 3.4 shows the per-stream average throughput, average number of violations, and the normalized power consumption (with respect to the Load Balancing method) for the proposed ML and heuristics approaches, State-of-the-Arts (SoA), including the neural Network (NN) [193] and heuristic Load Balancing (LB) [189].

As shown in the table, all the methods compared can achieve an average throughput more than the QoS constraint, i.e., 24 FPS. However, since one of the main goals is to minimize power consumption, spare throughput is not of much help as it adversely increases the power consumption unnecessarily. Although this spare throughput can be buffered, large buffer sizes are required which can adversely affect the power consumption. Also, in more practical cases, such as a real-time multi-stream scenario, it is necessary not to waste resources available to other videos by encoding more than the required throughput for a particular video. In this regard, LB [189] has the least control over the HEVC encoder and the throughput, thus, it increases the power consumption by outputting more frames, on average, yet violating the minimum required FPS, 24, more frequently than any other methods shown in Table 3.4. My proposed heuristic, on the other hand, can significantly reduce the power consumption compared to LB [189] thanks to the application-specific content-aware multi-threading approach. Although this approach outperforms the NN [193] with respect to power-efficiency, it cannot provide as few QoS violations. In fact, since the proposed re-tiling is performed only for the first frame of the GOP (every 8 frame), abrupt content variations within a GOP may cause several QoS violations.

Finally, my proposed ML framework is able to further enhance power consumption compared

to both the proposed heuristic and NN [193]. This additional power saving is due to more accurate throughput estimation obtained by ML. In fact, as shown in Figure 3.11b, there are several points where NN [193] underestimates the throughput and has to select a configuration with either higher frequency or larger number of threads than required, which leads to higher power consumption. For instance, from Sample 11 to Sample 15, NN [193] chooses 1.6 GHz as the operating frequency of 10 threads, and is unable to find any other configuration to satisfy the QoS with a lower power consumption. Nevertheless, 1.5 GHz and 9 threads as system configuration could also satisfy the QoS requirement with less power consumption. Unfortunately, NN [193] discards this configuration because it mispredicts the corresponding throughput to be less than 24 FPS (see Line 1 in Algorithm 3.2).

3.8 Summary

Multi-objective management of multiprocessor systems in presence of workload variation requires more novel solutions than the existing popular heuristics. Indeed, heuristics, which are mostly based on the designer's knowledge and intuition of a particular application, are not flexible enough to address the application requirements with highly time-variant workloads. Especially, for today's trending applications and services, such as video streaming, where the workload variations do not simply come from different functions within the application, but rather depend on the rapid changes in the input, heuristics are insufficient for power and performance management.

In contrast to heuristics, machine learning (ML) is more capable of dealing with dynamic environment, such as highly time-variant applications. ML algorithms, however, require informative observations from the environment. Modern multiprocessor systems expose several performance counters collecting hundreds of hardware events. These events can provide very useful and low-overhead insight into the system and application behavior.

In this chapter of my thesis, I have first assessed the content-based workload variation of HEVC encoders, as a case study of highly time-variant applications. Then, I have addressed workload prediction and throughput estimation under different system-level parameters, including number of processing cores and operating frequency. In particular, I have leveraged supervised and unsupervised learning to interpret performance counter values available on modern multiprocessor systems. I have compared the accuracy of workload prediction and throughput estimation provided by the proposed ML solution with a neural network based approach that leverages the same hardware events. The results have demonstrated that for highly time-variant workloads, such as HEVC encoding, state-of-the-art fails to adapt the system-level parameters such that the computation demand of the QoS-sensitive application is thoroughly satisfied. In this context, my ML framework could reduce the QoS violations by at least 3.4x, while decreasing power consumption by 15% due to its more accurate predictions than state-of-the-art approaches.

Moreover, I have compared my ML framework with two heuristics. The first one was a load

balancing approach for multimedia applications from state of the arts, while for the second one, I have proposed and developed an application-specific content-aware heuristic power and performance management for HEVC encoders. Although both of these heuristics could satisfy the average throughput constraint, they suffered from insufficient resources allocated to the active threads, or wasting the resources, both as a consequence of rapid changes in the workload. Therefore, ML framework decreased the power consumption of the target multi-core server by 33% and 12% compared to the load balancing and proposed heuristic approaches, respectively, while enhancing the QoS violations by 4.5x and 4.0x, respectively.

Despite the fact that the proposed heuristic outperformed the conventional heuristics, it could not bring about the same power efficiency as of the proposed ML framework. More importantly, these heuristics, including the one proposed in this chapter, are mostly formed based on prior knowledge or intuitions, are usually application-specific, and are inflexible in presence of rapid workload variations. On the contrary, the proposed ML framework suits any time-variant application, achieving a robust solution for power and performance management of multiprocessor systems.

4 Reinforcement Learning for Run-time Management and Design Space Search

4.1 Introduction

Due to the lack of flexibility of the conventional approaches, such as heuristics, in confronting large and dynamic design spaces, a few of important aspects of multi-objective management of multiprocessor systems have been hardly addressed. In this context, despite the important role of adaptive cooling in lifetime reliability of multiprocessor systems, heuristic and optimal DTM methods avoid enlarging and complicating the design space by not adding adaptive cooling parameters, such as fan speed [131]. Although in most platforms fan speed can be dynamically adjusted for more efficient heat removal, many DTM approaches do not consider it, or simply rely on the default settings handled by the OS [218], which could be sub-optimal since it does not simultaneously consider all available runtime parameters. Once these new control knobs are added to the common system-level parameters, such as DVFS, novel approaches are needed, especially in the presence of workload variations as in the video streaming applications. Moreover, since the fan power may account for more than 20% of the power consumption in a typical server [219], it is vital to consider its power consumption in a DPM scheme. All these aspects add to the complexity of multi-objective runtime management of multiprocessor systems.

Task allocation is a very well-known problem in DPM and DTM of multiprocessor systems. Optimal task allocation at runtime, however, is NP-hard [75], thus, has been addressed through low-overhead heuristics, rather than sophisticated time-consuming optimization methodologies. Nevertheless, conventional heuristics require the designer to master the target application and the underlying system to some extent. Prior knowledge regarding an application yet is not always easily attainable. In many applications, such as multimedia streaming, abrupt workload variations due to unforeseen input change may radically alter any planned-ahead scheme for (near-)optimal task allocation. Moreover, there are application, such as HEVC video streaming, that provide a number of internal parameters through which the complexity of the application and, thus, the workload, can change dramatically, based on the users' demands and requirements. Furthermore, in video streaming services, multiple streams may

be run concurrently at the same server. This scenario also adds to the workload variations that already exist due to the rapid changes of a single video contents. Since heuristics are not flexible enough to deal with such a dynamic problem, deploying new methodologies is vital.

Moreover, nowadays, new trending applications and services, such as HEVC streaming and Deep Learning (DL) expose a large number of internal parameters that need to be adjusted dynamically at runtime or set at design time. On one hand, HEVC encoding urges the designers to dynamically tune several wide-ranging parameters at runtime to maintain video quality along with other ordinary design objectives and constraints. Adding these application-level parameters to the system-level ones such as DVFS, task allocation, cooling, etc., leaves the designer with numerous choices most of which do not suit particular objectives and constraints posed by the application and the underlying system. On the other hand, Convolutional Neural Networks (CNNs), as a member of DL, include tens of layers each with several parameters (a.k.a hyperparameters) that extensively impact not only the model accuracy and training time, but also the inference time (performance) and energy consumption of multiprocessor systems. Hence, CNN designers should exhaustively search for the best hyperparameters through which both application- and system-level objectives and constraints are satisfied. Such an extremely large design space, thus, necessitates more novel approaches rather than the intuition-based heuristics.

On the contrary to heuristics, Machine Learning (ML) is more flexible in coping with large-scale and dynamic problems. However, direct runtime management of multiprocessor systems cannot be accomplished through classical machine learning algorithms. In fact, this is not a clustering, classification, or prediction problem, but rather a scheme that enables direct interaction with the environment and learning all its dynamism is required. Reinforcement Learning (RL) provides this opportunity by directly interacting with the environment and continuously learning the credit of each design parameter in different situations with respect to the objectives and constraints. In particular, RL includes model-free algorithms which do not require any prior knowledge and intuition about the problem. In fact, RL facilitates automation of runtime management and design space search for complicated problems such as multi-objective management of multiprocessor systems.

In this chapter of my thesis, I leverage RL for multi-objective runtime management of multiprocessor systems and design space search for new trending applications and services. In particular, first, I provide a comprehensive study of HEVC encoder and Deep CNNs (DCNNs) with respect to their application-level parameters. Then, after overviewing the background concepts of RL, I apply Q-Learning, a model-free algorithm of RL, to various multi-objective problems, as follows:

- I propose an RL-based framework to incorporate adaptive fan speed control to other traditional runtime parameters, such as DVFS and thread allocation on multiprocessor systems. The proposed framework aims at maximizing the performance and minimizing the cooling power, while maintaining the peak temperature below a thermal threshold.

- I address multi-user HEVC streaming on heterogeneous multiprocessor systems. The proposed approach maximizes the number of streams encoded simultaneously (throughput) through DVFS and adaptive stream allocation with respect to different video contents and HEVC preset parameters.
- I use RL to enable joint optimization of application- and system-level parameters for multi-objective runtime management of multiprocessor systems. The proposed approach is able to dynamically adjust several internal HEVC encoder parameters along with DVFS on a per-core basis. I consider encoding time (performance), video quality and compression (QoS), power consumption, and peak temperature as the design objectives and constraints.
- To achieve a fast, yet accurate exploration when RL faces extremely large design spaces, I propose a novel multi-agent RL (MARL) framework. Through this framework the joint design space of HEVC encoder and the target multiprocessor system are split to multiple smaller spaces. Each RL agent independently explores a specific assigned design space, while exploiting its experience cooperatively with other agents.
- Finally, I propose a MARL framework for hyperparameter optimization of DCNNs at design time through which it maintains the state-of-the-art accuracy while improving the training and inference time.

4.2 Case-Study Applications and Design Space

In this section, I provide a comprehensive study of two trending applications, which expose adjustable design- and run-time parameters. In particular, I assess the impact of these parameters on QoS of the application, as well as the main design objectives of multiprocessor systems, such as performance, power consumption, and temperature.

4.2.1 HEVC Encoder: Run-time Parameters

In 2015, real-time entertainment already accounted for more than 74% of downstream network traffic in North America, with streaming services, including Netflix, YouTube, and Amazon Video, accounting for 57% of the global share [22]. Moreover, video streaming services continue to grow, and users are shifting towards the use of emerging video technologies, such as 4K video resolution; thus North America is expected to be the first region surpassing the 80% downstream streaming traffic threshold by the end of 2020 [22]. As a result of the network pressure posed by video streaming services, a shift to next generation video encoding standards, such as High Efficiency Video Coding (HEVC), is vital. HEVC provides twice the compression as of its predecessor, while keeping the same video quality [220]. However, such a considerable reduction in bandwidth requirement is accompanied by a 10x higher computational complexity. This fact poses challenges of time and energy consumption on the video providers' servers.

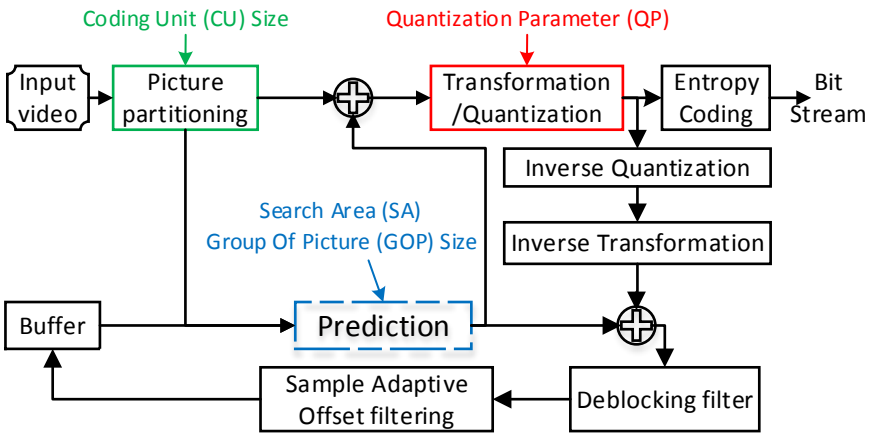


Figure 4.1 – HEVC encoder block diagram and main configuration parameters

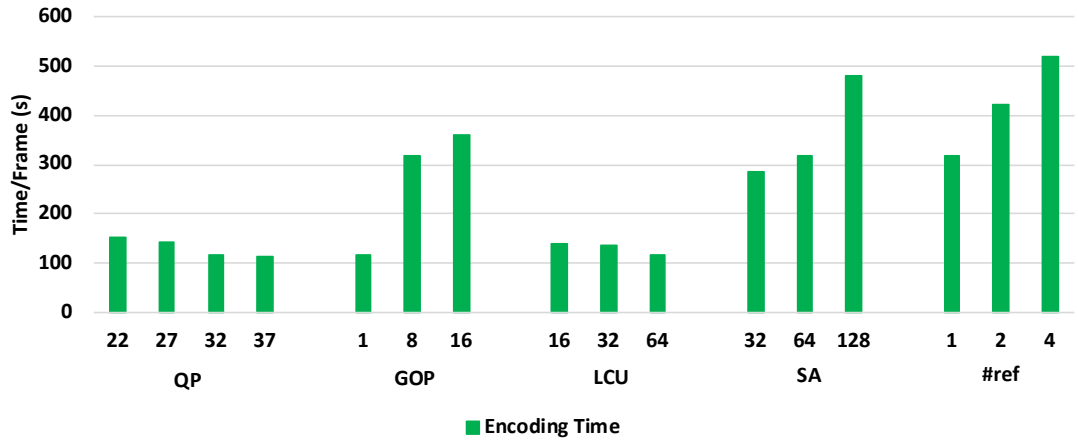
Table 4.1 – Application and system parameters, and corresponding selected values

QP	22	27	32	37
Search Area (SA)	128	64	32	
LCU size	64	32	16	
GOP size	16	8	1	
# reference frames	4	2	1	

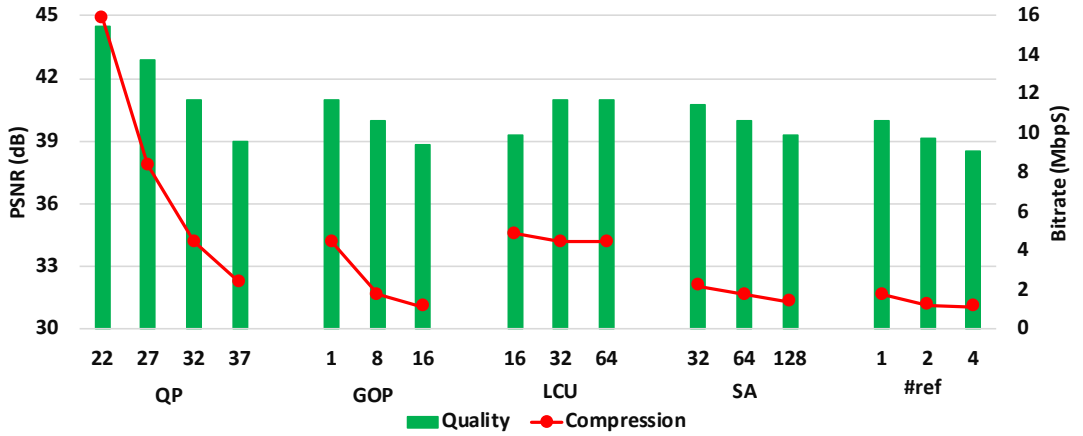
HEVC also brings more flexibility in terms of encoding parameters, which also makes selecting the most appropriate encoding configuration more challenging.

HEVC encoder provides designers with more than 100 of parameters, most of which tunable at run-time to facilitate trade-offs between the encoding complexity, QoS, power consumption, and performance. In this study, however, I consider those parameters that can significantly affect either of the following metrics: video quality, video compression, performance (encoding time or throughput), power consumption, and thermal profile. This study is different from the one presented in Section 3.2, as this one is focused on the impact of different HEVC internal parameters rather than the impact of the video content variation.

Figure 4.1 illustrates a simplified HEVC encoder block diagram. Each block contains several parameters to configure the encoder (i.e., configuration parameters). Prior to all the blocks, the largest CU (LCU) size is specified. Search area (SA), prediction mode, GOP (group of picture) size, and reference frames are used in the prediction block. Also, the quantization parameter (QP) is used to control the level of quantization. All these parameters can be dynamically tuned frame-by-frame, except for the GOP size that can only be changed every several frames. Finally, when $GOP = 1$, frames are considered as I-frame (i.e., intra-picture prediction only), while when $GOP \neq 1$, only the first frame is considered as I-frame and the rest in the GOP structure are B-frames (i.e., inter-picture prediction is also applied). Number of reference frames denotes the number of frames kept in the internal decoder for motion search and is always smaller than the number of pictures in GOP.



(a)

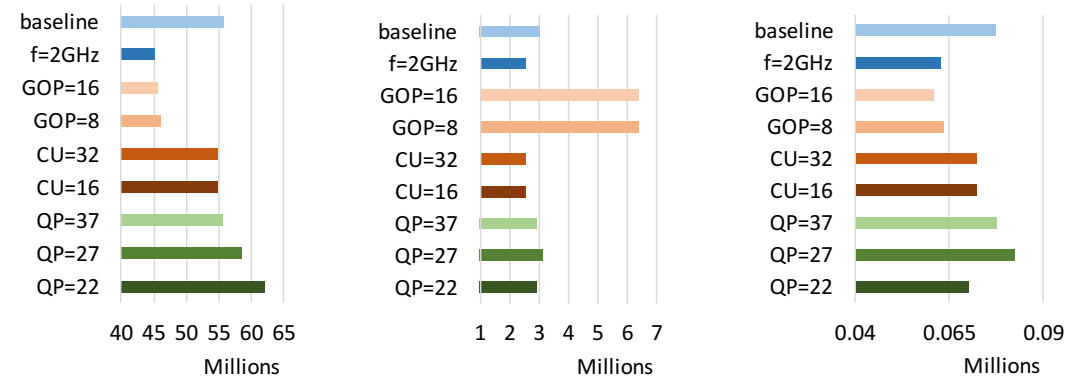


(b)

Figure 4.2 – Impact of different encoding parameters on (a) encoding time, and (b) PSNR and bitrate, for the test sequence *Tennis*

Table 4.1 lists these parameters and their corresponding values I consider in this study. Some of these parameters can take a wider range of those shown in Table 4.1. For instance, in HEVC standard, QP can theoretically take any values from 1 to 51. However, I limit this range based on my observation on the output video quality and compression, as well as according to the guideline provided by Joint Collaborative Team on Video Coding (JCT-VC) [181]. The idea behind constraining some of these encoding parameters is to avoid irrelevant values that only add extra complexity without any gain with respect to the design objectives. Figure 4.2 shows, on average, how different parameters affect encoding time, PSNR and bitrate.

Similar to the video contents (discussed in Section 3.2.2), encoding parameters affect the memory sub-systems considerably, resulting in significant change in encoding time, power consumption, and temperature. Figure 4.3 shows how different encoding configurations in addition to the operating frequency change the average number of L2 and LLC accesses, and



(a) L2 Accesses (b) LLC Accesses (c) LLC Misses

Figure 4.3 – Average number of accesses to L2, accesses to LLC, and misses from LLC every second

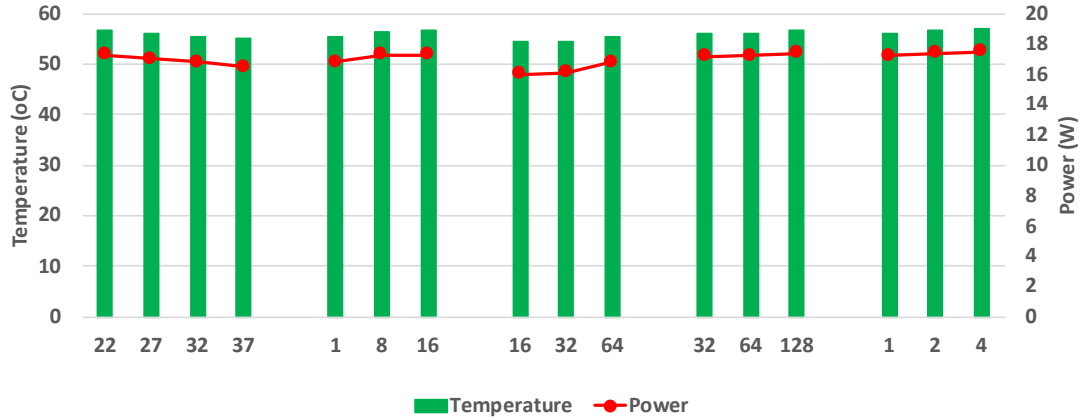


Figure 4.4 – Impact of application parameters on CPU power and temperature for *Tennis* running on one core

LLC misses for test sequence *Tennis*. For this figure I use the default *main Intra* configuration (QP=32, GOP=1, CU=64) with the maximum frequency (2.4 GHz) as the *baseline*. Then, I provide the number of memory events by changing these system- and application-level parameters. Because these metrics are correlated with encoding time, as previously shown, improvements on encoding time at the CPU level will also have a beneficial impact on the power consumption of the memory subsystem, due to the reduced accesses to memory. Such observations imply the significance of application-level parameters in encoding efficiency and time, ultimately affecting power and temperature of the chip. Figure 4.4 shows how power and temperature are affected by HEVC encoding parameters.

Finally, while CPU frequency does not affect the encoding efficiency, it plays a major role in encoding time, power consumption, and peak temperature, as shown in Figure 4.5. Therefore, frequency has to be considered as a major runtime parameter along with all application

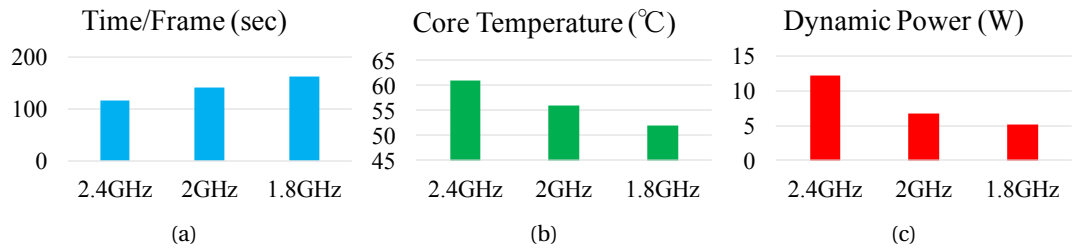


Figure 4.5 – Impact of frequency on encoding time, power and temperature, for the test sequence *Tennis* running on one core

parameters.

The undeniable complexity of HEVC, together with the increase of video streaming users, poses an important challenge for power- and thermal-aware resource allocation and management of these applications when running on multi-core servers. In particular, users daily upload more than 65 years of contents to YouTube servers [221]. In order to satisfy the massive streaming to a wide variety of personal devices, video providers need to decode and encode the video into several formats (a process named *transcoding* [222]). This poses a high computational burden on the providers' server facilities, leading to the need of developing runtime quality-aware power and thermal management for multi-core servers. Current research in this area is mostly focused on the software optimization of one or several blocks of the encoding algorithm. However, to address the challenge of power and thermal management for HEVC, application-level configuration and system-level parameters need to be jointly integrated on top of algorithmic optimization. In addition, when dealing with multiple encoding requests on a multi-core server, a proper video (i.e., workload) assignment strategy is vital for reducing the thermal hot spots while maintaining the desirable encoding time. Indeed, the increased complexity of HEVC requires high-performance architectures and induces more frequent hot spots. In this chapter, I address these issues through RL.

4.2.2 Convolutional Neural Networks (CNNs)

CNNs encompass a large range of different architectures and layer depths. While the true history of DCNNs started from AlexNet [223] with only 6 layers, nowadays there are many deeper CNNs available, such as VGG [224] with up to 19 layers, and ResNet [225] with up to 1022 layers [226]. On one hand, these CNNs are composed of several types of layers, such as *convolution*, *pooling*, *fully connected*, *softmax*, etc. During the training process, there may be millions of internal parameters (e.g., weights) within the CNN that are optimized with respect to a particular loss or accuracy metric via an optimizer, such as Adam [227] and Stochastic Gradient Descent (SGD) [228]. On the other hand, each layer has a couple of so-called hyperparameters, that unlike these internal parameters, are not or cannot be optimized during the conventional training procedure. Researchers and CNN designers traditionally trust rule-of-thumb approaches followed by grid search or random search. While even with

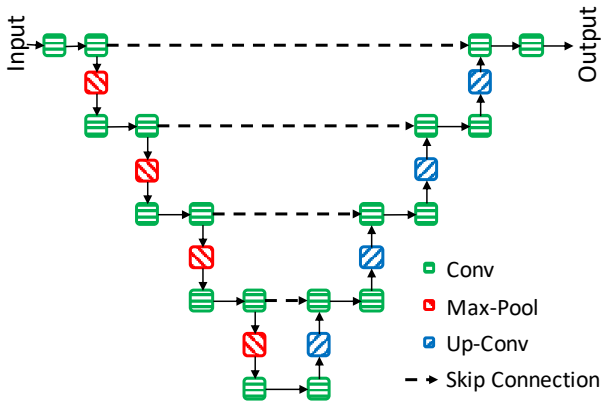


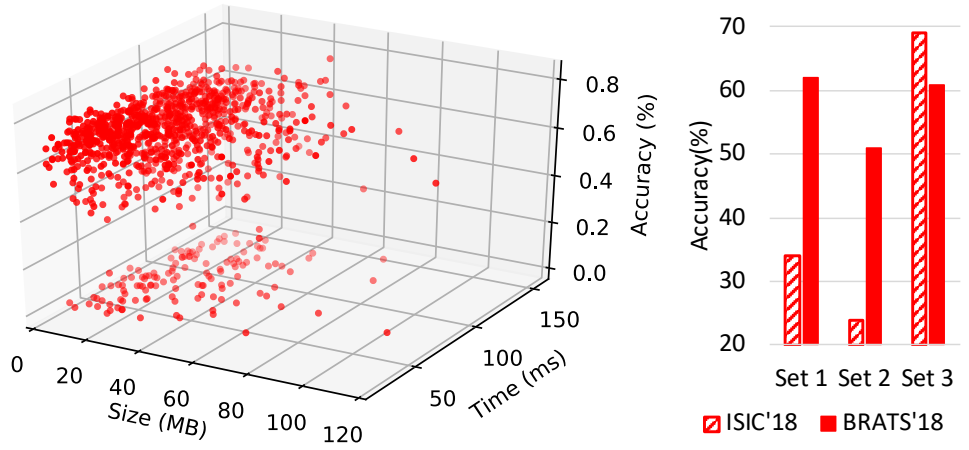
Figure 4.6 – A simplified U-Net architecture

small number of layers grid search seems impractical, with deeper CNNs random search also becomes too time-consuming and inefficient.

Figure 4.6 shows a U-Net architecture profoundly used in semantic segmentation of biomedical images [229]. For clarity, I consider only the number of kernels and kernel size of convolution layers (including *Up-Conv* layers, a.k.a *Deconvolution*) as the hyperparameters. Moreover, I limit the design space of hyperparameter as follows. Number of kernels can be any value from $\{16, 32, 64, 128, 256, 512, 1024\}$ and kernels are square-like where width and height are equal and can be any value from $\{3, 5, 7\}$. Therefore, according to the fundamental counting principle, there are 7×3 different choices for each convolution layer. Finally, since there are 22 convolution layers in the U-Net shown in Figure 4.6, where each is designed independently, 21^{22} different choices are available when designing such a CNN. Throughout this work, I refer to any combination of different hyperparameters of different layers as a hyperparameter set.

In addition, Figure 4.7a shows model size, training time per batch, and accuracy (with respect to Intersection over Union metric) for 1000 different hyperparameter sets used to train the U-Net described in Figure 4.6 for a limited number of epochs on an NVIDIA V100 GPU. As shown in Figure 4.7a, there are many sets of hyperparameters for which the accuracy, as the most important metric in designing DCNN, remains close to 0. On the other hand, although for several hyperparameter sets the U-Net converges to higher accuracy, the training time and model size, as the other two important metrics, change considerably depending on the hyperparameters. Obviously, smaller models with shorter training/inference time and higher accuracy are the most desirable of all. However, finding such a hyperparameter set that satisfies all constraints and objectives is extremely challenging with such a huge design space of 21^{22} different hyperparameter sets.

Finally, Figure 4.7b compares the accuracy of a U-Net trained for limited number of epochs with two different datasets, BraTS'18 [230–232] and ISIC'18 [233, 234] . As shown in the figure, with the same hyperparameter sets, the output accuracy is different due to different input data. This observation demonstrates that a successful hyperparameter optimization should



(a) (b)

Figure 4.7 – a) Model size, validation accuracy, and inference time for 1000 different sets of hyperparameters for BraTS'18 dataset, b) Accuracy with three different hyperparameter sets for BraTS'18 and ISIC'18 datasets

be data-driven. In this context, a successful hyperparameter optimization approach would be the one that considers input data characteristics and tunes the hyperparameters accordingly, since I require a particular DCNN perform well on the specific dataset for which it is trained.

Among all traditional approaches, RL is very efficient in dealing with very large design spaces and is known to provide such a data-driven solution, as it does not require any prior knowledge about the input data.

4.3 Reinforcement Learning: Background Concepts

Reinforcement Learning (RL) is a paradigm of learning process where a single agent or multiple agents learn overtime to behave optimally in a certain environment by continuously interacting with that environment. In fact, the agent experiences various situations in the environment and has to take an action to deal with this situation. Overtime, the agent learns to modify the actions with respect to each experienced situation such that an optimal behavior is attained at the end of the learning process. Thus, RL is appropriate for problem domains where reinforced information is available after a sequence of actions is performed in the environment. RL, in particular, is able to deal with environment-dependent problems through dynamic optimization programming [235]. RL also brings about a promising solution for Markovian Decision Processes (MDPs) where due to the large number of states dynamic programming becomes infeasible. Figure 4.8 shows a simple RL scenario. As shown in the figure, ***Environment*** can be anything with which the ***RL Agent*** needs to deal. In the case of multi-objective management of multiprocessor systems, the environment can be composed of the input data, application, and the underlying multiprocessor system. The agent interacts with the environment by taking an action among a set of available ***Actions***. This action, by

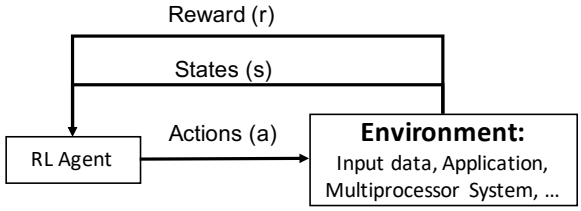


Figure 4.8 – A basic Reinforcement Learning scenario

the scope of interest of this thesis, can be any run-time or design-time parameter, such as DVFS, task allocation, application-level internal parameters, fan speed, etc. As a consequence of applying different actions to the environment, the agent observes a set of different ***States***, along with a reward signal or a function of multiple reward signals. The reward value, in the context of this thesis, represents the design objectives and constraints. The goal of the RL agent, thus, is to find actions to be taken in specific states so that the design objectives and constraints can be met. In the multi-objective management of multiprocessor systems, the reward may include power, performance, and thermal optimization, as well as the QoS.

4.3.1 Model-Based vs. Model-Free RL

In general, there are two types of RL: model-free, and model-based algorithms. In model-based RL, the agent builds an internal model of the transitions from one state to another and the immediate rewards of such transitions. Then, using this model, the agent decides which action should be taken in a particular state. In the contrary, in the model-free RL, the agent directly learns a state-action value or policy from dealing with the environment, without estimating or modeling it. Examples of model-free RL algorithms include Monte Carlo Control, SARSA, Q-learning, and Actor-Critic, where the agent relies on real samples from the environment rather than generating predictions of the next state and reward.

In many real-life problems, an accurate preexisting model is not available. Similarly, in multi-objective management of multiprocessor systems modeling the environment, i.e, how power, performance, temperature, and QoS change with respect to the input data, as well as the application- and system-level parameters is infeasible. Therefore, model-free RL is a more promising solution.

4.3.2 Single-Agent vs. Multi-Agent RL

RL traditionally refers to Single-Agent Reinforcement Learning (SARL) where there is one, and only one agent dealing with the environment. In contrast, Multi-Agent Reinforcement Learning (MARL) employs more than one agent in interaction with the environment to cope with more complicated problems. MARL is composed of multiple agents competitively or cooperatively coping with a particular problem. While in competitive MARL agents compete each other to maximize their own reward obtained from the environment, in cooperative

MARL, agents help each other to more efficiently solve a problem and, thus, obtain a higher shared reward. In particular, the latter is beneficial if the task given to an agent is very large or complex to handle. In this context, agents can be homogeneous, i.e., they have exactly the same features and responsibilities. In this version of MARL, agents explore the whole environment autonomously, but share their experience, e.g., the learned policies, to each other to come up with a more likely optimal policy. In contrast to homogeneous agents, cooperative MARL agents can have different features and, hence, different tasks can be assigned to them. These so-called heterogeneous agents can split the large and complicated task to provide faster, yet accurate exploration in the environment.

4.3.3 Q-Learning

For very large design spaces, where it is practically impossible to explore the whole space, Q-Learning (QL), as a model-free RL algorithm, is shown to outperform other model-free RL algorithms, in providing greedy solution with limited number of observations. Also, compared to other well-known RL algorithms, QL is able to interact with more sophisticated industrial applications [236]. The (single-agent) QL is composed of an agent able to take actions from a finite action set, **A**, and capable of observing (sensing) its current state from a finite state space, **S**. The agent is in charge of applying actions starting from an initial state and move to a new one. Applying particular actions in particular states is encouraged or discouraged based on a reward. The agent, then, maximizes this reward by storing a Q-value per state-action pair as $Q^{\pi}(s,a)$ to indicate the quality of applying action a in state s. Starting from a random policy for taking actions, the agent is ultimately able to follow a learned policy, π, which is a mapping from the state space to the action set. This map simply implies whether action a in state s is worth to apply. In other words, this value represents the most probable long-term reward, if the agent starts from state s, applies action a, and follows the policy π. The Q-values and the Q-table, which are usually initialized to zero, are updated as follows [235]:

$$Q_{t+1}(s_t,a_t) = Q_t(s_t,a_t) + \alpha(s_t,a_t) \times [R_{t+1} + \gamma \max Q_{t+1}(s_t,a) - Q_t(s_t,a_t)], \tag{4.1}$$

where $Q_t(s_t,a_t)$ and $Q_{t+1}(s_t,a_t)$ are, respectively, the current and updated Q-values corresponding to current action and state (a_t and s_t), R_{t+1} is the reward observed after a_t is applied at state s_t, $\alpha(s_t,a_t)$ determines the learning rate, and γ is the discount factor and controls the significance of the history of the Q-values against the recently obtained reward.

The learning rate defined in Q-learning depends on the state-action pair. In fact, through the learning rate definition, the agent ensures whether a specific state-action pair has been sufficiently observed. The learning rate is indirectly determining the next action (in the exploitation phase), but it only depends on the current state (s_t) and the current action (a_t). In stochastic environments where action a_t at state s_t does not always result in a particular next state s_{t+1}, the learning rate is very important to ensure a fast and flawless learning. If the learning rate is assumed constant and set to 1, the previous reinforced information is overridden every time the state-action pair of (s_t,a_t) is observed. If the learning rate is constant

and set to zero, there is no learning process. For fully deterministic environments, $\alpha(s_t, a_t) = 1$ provides optimal learning. In contrast, for stochastic environments, a decreasing-to-zero function for learning rate is able to provide optimal learning phase [237].

4.4 Literature Review

In the following, I first overview the state of the arts on RL for multi-objective runtime management of multiprocessor systems and design space search. Thereafter, I summarize recent works covering adaptive fan speed control, multimedia workload allocation, system-level HEVC encoder optimization, and neural architecture search and hyperparameter optimization.

4.4.1 RL for Runtime Management and Design Space Search

RL has been used for two different perspectives in the literature of multiprocessor systems. One group of existing works leverage RL as a design space search algorithm [238–240]. This application of RL has recently attracted a lot of attentions for Neural Architecture Search (NAS) and hyperparameter optimization. Nevertheless, the majority of RL-based methods have been used to automate runtime management of multiprocessor systems [125, 126, 81, 241–247]. In the latter, similar to the former, RL indeed explores a design space, however, the experience gained during the exploration will be directly applied at run time for power, performance, and thermal management. In this context, Das et al. [125] address thermal and lifetime optimization of multiprocessor systems through RL. An RL-based DTM approach is proposed by Khan and Rinner [241]. Hu et al. [242] leverage RL for frame rate control of multimedia applications. Virtual machine (VM) allocation in data centers is addressed by Pahlevan et al. [81] through RL. An RL-based DVFS scheme is presented by Chen et al. [243]. Shen et al. [244] address power management of multi-core systems by using RL. Viswanath et al. [12] adapt RL for real-time task scheduling. Finally, RL is used for performance optimization [245] and energy efficiency [246, 247]. Despite such a rich literature in using RL for runtime management of multiprocessor systems, most of these problems cannot be justified for RL and have very well-known heuristic alternatives. In fact, a problem is motivated to be solved through RL if the design space is too large or dynamic enough to be handled by conventional heuristics. Nonetheless, the aforementioned works do not address such problems, such as joint optimization of application- and system level parameters, especially for input dependent workloads.

4.4.2 DTM with Adaptive Fan Control

A large number DTM policies have been proposed in the literature [127]. However, many of them discard active cooling [18], or simply consider its power consumption in a power model [128], rather than leveraging adaptive control schemes to further improve the DTM efficiency.

Nevertheless, a few works consider fan speed as a control knob. In particular, an optimization framework is proposed by Dousti and Pedram [129] to find the optimal fan speed. Nonetheless, the framework has not been validated through real measurements and its applicability when the number of cores scales remains in question. TECfan [130] uses a look-up table created offline for different fan speeds and workloads. A similar work by Zapater et al. [131] addresses adaptive fan speed control based on look-up tables. However, the provided DTM policies are limited to the studied workloads and may not be applied to unseen workloads. Hanumaiah and Vrudhula [132] formulate the multiprocessor temperature as a convex function of fan speed and solve three optimization problems separately in different time scales to find the optimal task migration, DVFS, and fan speed. In this solution, however, the number of released tasks determines the active cores. Thus, the number of cores and performance could be non-optimal. Furthermore, solving a convex optimization problem is time-consuming and is not recommended for prompt decisions in runtime management of multiprocessor systems. Core temperatures are estimated using neural networks for preemptive fan control by Acun et al. [133]. In this work, authors consider very large intervals (in order of minutes) for workload variations such that each core temperature reaches its steady state. Unfortunately, for many applications, workload may vary in order of seconds or even less [248]. Chan et al. [135] define a convex optimization problem to find the optimal fan speed. This work, however, does not take into account other system settings, such as number of cores and core frequency. Finally, Kim et al. [136] propose a fan speed controller to guarantee server operation stability without directly considering optimization of cooling power.

4.4.3 Workload Allocation of Multimedia Applications

An aging-aware energy-efficient workload allocation for mobile multimedia platforms is proposed by Paterna et al. [249]. In this work, only three computational kernels of multimedia workloads have been considered as the test-case study. Nan et al. [250] propose a workload balancing approach for cloud-based multimedia applications. This approach is validated on a simulation framework and the test-case workloads do not represent the real-world multimedia applications. Similarly, VM (Virtual Machine) allocation of multimedia workloads has been addressed by several works [251, 252], however, none of them deal with content-based workload variations.

On the contrary, an MPEG-2 video streaming workload allocation for real-time decoding is proposed by Mendis et al. [253]. Nonetheless, this work does not take into account any constraints on the processing platform. Lee et al. [254] address video quality adaptation for power-constrained systems through workload allocation. However, this work considers the whole stream as the workload and variations within the video contents are discarded. Song et al. [255] address power management of video transcoding while considering system-level parameters such as DVFS, task allocation, and thread migration. Nevertheless, the proposed approach simply characterizes the workloads with respect to the requested video transcoding deadlines. Finally, Khan et al. [189] employ DVFS and thread allocation for HEVC encoding on

multiprocessor systems. However, in this work, workload only accounts for a single stream paralleled to multiple threads. In a multi-user environment this is not true though.

From the above discussion, there are several topics in workload allocation of multimedia applications that have been either not considered or only partially addressed by the literature. Many of these works [250–252] assume VMs of multimedia workloads and address VM allocation in data centers. The definition of workload variation in the existing works does not comprehensively represent content variations within a single video and among different videos. Finally, only a few works have considered the most recent video coding application.

4.4.4 HEVC Optimization and Runtime Management

Research in multimedia applications [256, 257] have been extensively discussed in the literature. In particular, there are several works providing power reduction and/or encoding time enhancement for multimedia workloads, most of which targeting the previous standards, such as H.264/AVC [258] (*e.g.*, [259, 260]). However, these works need modifications to conform with HEVC requirements due to its higher complexity.

Several implementations of HEVC standard exist, ranging from the non-real-time HM Test Model [179] as the reference software, to Kvazaar [261] and x265, both of which are able to provide real-time HEVC through thread-based parallel processing. While Kvazaar is equipped with parallel processing at 1) tile, 2) Wavefront Parallel Processing (WPP), and 3) picture level, the first one does not exist in x265.

HEVC optimization includes optimizations at encoder and decoder levels. Real-time HEVC decoder [262–265] has been already addressed mainly through hardware acceleration [266]. Power-aware streaming has been also accomplished by He et al. [267], targeted at mobile devices. Therefore, the recent research has focused more on the encoder as it is approximately 100 times more complex [177]. In this context, the majority of the work includes algorithmic optimization with various objectives such as power and complexity reduction, encoding time enhancement, etc.

Several works focus on the time reduction and performance improvement of HEVC encoders [268, 269]. One approach to increase the throughput of the encoder is taking advantage of the tile feature in the HEVC standard. In this context, Shafique et al. [188] and Khan et al. [270] split a frame into different number of tiles and gain speedups since each tile can be processed independently and, hence, regarded as a thread. Moreover, the HEVC encoder complexity is highly dependent on the depth levels of Coding Units (CUs) [176]. Thus, CU depth reduction has been addressed by recent works [271, 272], where it is shown that the reduction of the computational complexity of the encoder leads to a decreased energy or latency of the application. Besides, several works directly aim at power reduction of HEVC streaming by proper thread allocation [189], and memory bandwidth reduction [273]. The number of reference frames also influences the power consumption since as it increases, more

data must be transferred and processed. Thus, Ma and Segall [274] decrease the memory bandwidth and power consumption by reducing the reference frames.

In all of the above works, the encoder optimization does not consider temperature as an important issue of today's multi-core servers. Nevertheless, a few works consider temperature constraints as well as encoding efficiency of next generation video encoders [275–278]. In particular, Palomino et al. [277] employ an adaptive approximate computing method at both algorithm and data levels to optimize the thermal profile. Alternatively, Palomino et al. [275], first, perform an offline analysis to explore the relation of video properties and encoding configuration with CPU temperature. Then, they propose an application-driven thermal management policy. Shafique and Henkel [278] address the complexity reduction of HEVC, use hardware accelerators for low-power HEVC, and apply a DTM similar to the work of Shafique et al. [279]. Authors in TONE [276] first extract Pareto optimal curves of the temperature associated with different encoding configurations. Then, a prediction model based on frame complexity is used to predict the temperature resulted from next frame. If a temperature threshold is exceeded, a new encoding configuration optimizing the encoding efficiency is selected.

Nonetheless, none of these works ([275–278] jointly consider power consumption, temperature, encoding time, and encoding efficiency. Moreover, when multiple videos are running at the same time on a multiprocessor system, power and thermal management of HEVC is more challenging and has not been addressed so far.

Real-time video transcoding (real-time encoding followed by real-time decoding) is another main challenge only partially addressed in the literature. Although there have been several works regarding real-time transcoding [280], video transcoding through next generation video coding standards, such as HEVC, which pose more challenges, has not been completely addressed so far. In fact, although there are a few works trying to achieve real-time software implementation of HEVC encoder, they are either focused on low-complexity, less-efficient encoding configurations [281] (such as main intra profile [178]), or limited to low-resolution videos [282].

Finally, HEVC encoders are composed of several processing blocks, each of which has multiple tunable parameters. Each parameter affects the encoder throughput, output video quality and compression, and power consumption. In addition, video format and contents play a major role in throughput, video quality and compression, and chip power consumption. Since video contents may vary frame by frame, encoding parameters should be adapted on a frame basis for QoS optimization under limited power budget. Based on these facts, recent works [276, 189, 242] have employed run-time adaptation of encoding parameters. These works, however, neither address QoS-aware real-time HEVC encoding nor consider a multi-user environment. More importantly, a few of previous works (e.g. [189]) have modeled the output and complexity of the HEVC encoder as a function of a few relevant encoding parameters by exhaustive profiling of the application. However, these models are considerably platform-

dependent and any change in the platform architecture may result in intolerable model error. In addition, a multi-user environment adds to the complexity and inefficiency of such models, since under limited resources and power budget, the parameters set for one video/user should be dynamically adjusted with respect to the encoding parameters set of other videos.

4.4.5 CNN Optimization and Design Space Search

Random Search [283], with almost no complexity overhead, is known as the baseline for hyperparameter optimization. Although under certain conditions and in specific problems it has been claimed that Random Search could be competitive to other neural architecture search and hyperparameter optimization approaches [283], in more complex scenarios with larger design space, as in the case of DCNNs, Bayesian Optimization, Genetic algorithms, and RL are shown to be superior solutions [284, 285].

Bayesian optimization is among the most popular methods for neural network architecture search and hyperparameter optimization [284, 286]. However, since Bayesian optimization is based on Gaussian processes, its application is limited to optimization problems with low dimensionality [287].

Evolutionary search and Genetic algorithms (GAs) are the most traditional approaches for neural network optimization [288] used recently in neural architecture search and hyperparameter optimization [289, 290]. These algorithms, however, are strongly dependent on heuristics. Unlike GAs, RL algorithms are based on Markov Decision Process (MDP), a mathematically grounded framework. Therefore, RL-based approaches for hyperparameter optimization have recently attracted a lot of attention.

These approaches can be divided in two different categories. In the first category, RL is used to optimize an existing network, either by tuning its hyperparameters or by adding and removing layers. In this context, Huang et al. [238] use policy gradient to prune filters in CNN while having the CNN perform at a desirable accuracy, whereas EAS [291] uses RL to enable extending pre-existing plain convolutional neural networks to more sophisticated structures.

In the second category, RL is used to build a complete network from scratch. Compared to the first category, this one tackles a larger design space, thus, it is more time-consuming. In this context, Zoph and Le [292] use Recurrent Neural Networks (RNNs) along with RL to build DNNs. MetaQNN [239] uses Q-Learning to build CNNs from scratch. A few works, rather than building the whole CNN, design blocks similar to famous Residual and Inception modules and build the network by concatenating them. Examples of these works are BlockQNN [293], PNAS [294], and ENAS [295].

One drawback of the sate-of-the-arts is that the types of layers are limited [239, 238]. Moreover, the maximum number of layers of the CNN should be known a priori in these works and the application of these approach in designing deeper CNNs remains in question [239, 292]. Finally, none of these works [292, 238, 239, 293–295] address multi-objective/constraint design of DC-

NNs. In contrast, MONAS [240] is a multi-objective neural architecture search approach which finds hyperparameters with respect to model accuracy and energy consumption. MONAS, however, considers a more limited subset of different action values (hyperparameters per layer) and requires redesign of the RNN-based controller for designing different CNNs.

Finally, all the works above only deal with architectures for image classification task. For other CNN applications, such as semantic segmentation, there are well-known architectures entirely different from those suitable for image classification.

Table 4.2 compares the state-of-the-art in terms of types of layer optimized, objectives and constraints considered, support for unconventional modules, and types of CNNs (tasks) evaluated. In this table, I differentiate between support for among arbitrary layers and support for residual modules, as the latter is an subset of the former. Also, since not in all these works stride size has been considered as a hyperparameter, I include it in the table.

4.5 Proposed DTM with Adaptive Fan Speed Control

Adaptive fan speed control as a DTM technique, if applied properly, can not only increase the lifetime reliability of multiprocessor systems, but also enhance performance while reducing cooling power. Nonetheless, adaptive fan speed control requires considering a new design parameter, probably with a wide range of values, into the conventional design space of multi-objective run-time management of multiprocessor systems. As a result of such explosive design space, conventional solutions such as grid search, and offline look-up tables [131], are infeasible, impractical, or insufficient. Therefore, many of existing DTM approaches completely discard this important design parameter to avoid coping with a more complicated design space. On the contrary, RL has recently proved to be promising in efficiently searching large design spaces.

To address adaptive fan speed control in this thesis, I adapt RL and, in particular, QL to effectively control the processor temperature by selecting DVFS points, fan speed values, number of active cores, and thread allocation. In particular, I let the QL agent explore the design space using a thermal simulator and learn the optimal DTM policy. Afterwards, I apply the learned DTM policy online on a real thermal test chip [299], proving the feasibility of my approach. Figure 4.9 shows an overall view of my work.

4.5.1 Experimental Setups

In what follows, first, I present the simulation framework, the thermal test chip, and the power and performance model used to generate arbitrary workloads.

Table 4.2 – State-of-the-arts on hyperparameter optimization and neural architecture search of CNNs

SoA	Optimized Layers			Stride	Objectives and Constraints				Unconventional modules			CNN task	
	Conv	Pool	Dense		Tr. time	Inf. time	Acc.	Size	Skip Conn.	Residual	Inception	Class.	Seg.
MetaQNN [239]	✓	✓	✓	✓	x	x	✓	x	x	x	x	✓	x
NAS [292]	✓	✓	✓	✓	x	x	✓	x	✓	✓	x	✓	x
EAS [291]	✓	✓	✓	✓	x	x	✓	x	x	✓	x	✓	x
BlockQNN [293]	✓	✓	x	x	x	x	✓	x	✓	✓	✓	✓	x
PNAS [294]	✓	✓	x	✓	x	x	✓	x	✓	✓	✓	✓	x
ENAS [295]	✓	✓	x	x	x	x	✓	x	✓	✓	✓	✓	x
MONAS [240]	✓	✓	x	x	x	✓	✓	x	✓	✓	x	✓	x
MANAS [296]	✓	✓	x	✓	x	x	✓	x	✓	✓	✓	✓	x
DARTS [297]	✓	✓	x	✓	x	x	✓	x	✓	✓	✓	✓	x
SNAS [298]	✓	✓	x	x	x	x	✓	x	✓	✓	✓	✓	x
Proposed	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

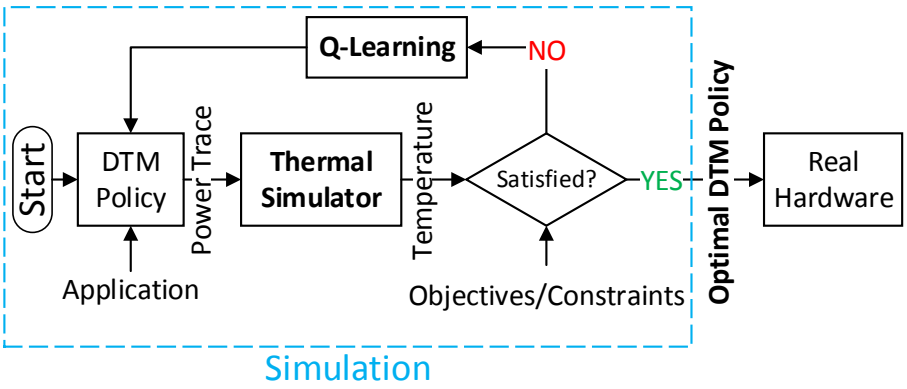


Figure 4.9 – Simulation framework and methodology for learning process

4.5.1.1 Simulation Framework and Methodology

I use a simulation framework to accomplish the learning process as shown in Figure 4.10. At each decision point, the RL agent selects a fan speed and system configuration, namely, number of active cores, mapping, and core frequency. The fan speed along with the fan geometry are used by the Open Modelica, an open-source environment for system simulating, to simulate heat transfer through the heat sink. 3D-ICE advances the chip thermal simulation with respect to power traces corresponding to the configuration system and the baseline dynamic power trace (i.e., baseline workload). To generate these traces from a baseline workload, I use a simplified power and performance model described in 4.5.1.4. Once the new decision time arrives, the RL agent receives the 3D-ICE output (heat map), application performance, and fan power. New state is determined and the reward corresponding to the previous state and the taken action is calculated. Such a framework lets the agent explore various system configurations under different conditions (workload, initial temperature, etc.) and optimize its behavior in the environment. Once the optimal behavior is learned, it can be used online on a real chip, as a DTM policy.

4.5.1.2 Thermal Test Chip

Validation of DTM is not a trivial problem due to the need to apply known spatial and temporal power profiles to a chip connected to the desired heat dissipation solution, as well as to measure the resulting temperature maps of the active silicon layer. Ordinary multiprocessor systems are unsuitable for this task, as they generally have few temperature sensors, with significant noise and their exact location on the silicon die is not publicly known. Moreover, it is not possible to apply a prescribed power spatial distribution to off-the-shelf multiprocessor systems. Decapping the chips and observing them through an IR camera is possible [300], however, it prevents connecting the chip to the desired heat dissipation solution.

For these reasons, I address the validation of the proposed DTM through a dedicated thermal test platform [299]. The chip is shown in Figure 4.11. This platform is built around a custom

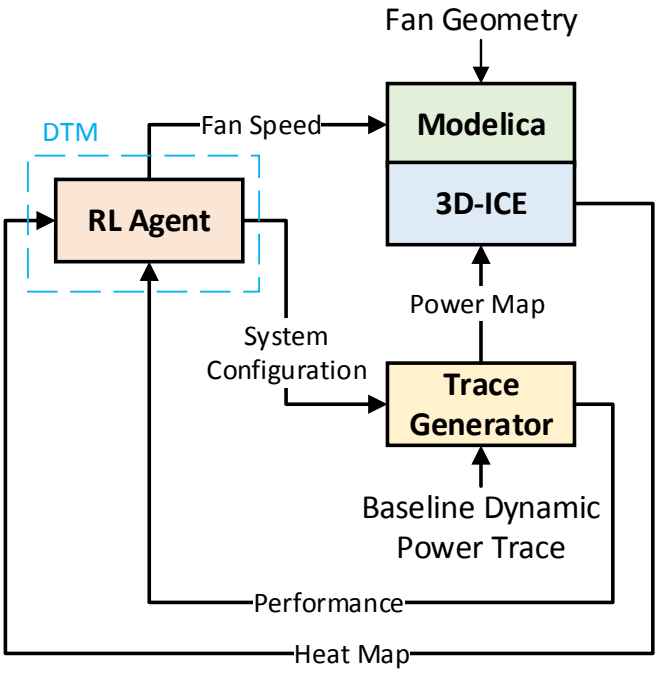


Figure 4.10 – Simulation framework and methodology for learning process

silicon integrated circuit providing a 4×4 array of heating elements and temperature sensors, as well as support electronics to apply arbitrary power profiles and measure temperatures with a $0.1°C$ resolution.

4.5.1.3 Heat Sink and Fan

I use an HS483-ND copper heat sink and P14752-ND fan whose operating voltage ranges from 12 V to 27.6 V. I assume 12 V, 18 V, and 24 V as the three available fan voltages. Therefore, by using the Fan Laws, I relate the fan speed to its power consumption, as follows:

$$P_{fan,k} = P_{fan,l} \times (\frac{S_{fan,k}}{S_{fan,l}})^3, \tag{4.2}$$

where P_{fan} and S_{fan} are fan power consumption and fan speed, respectively.

Although P14752-ND is a low-power small fan suitable for the TTC described in Section 4.5.1.2, my work is valid for any type of fan with any range of operating voltage/speed and chip.

4.5.1.4 Power and Performance Model

I assume that the performance of an application is equivalent to the speedup obtained compared to a baseline performance, which I consider when the application is running on a single core with the minimum operating frequency. Therefore, I consider the modified Amdahl's

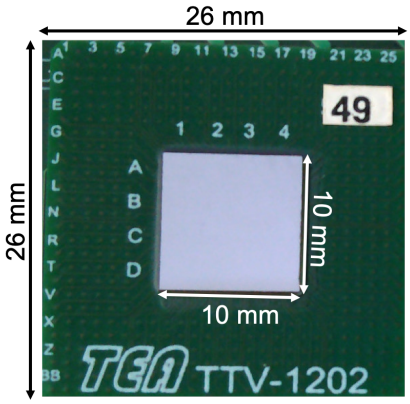


Figure 4.11 – The thermal test chip used for the validation.

Law [5], where the speedup also linearly scales with the ratio of current operating frequency over minimum frequency (i.e., in my case, F/F_{min}):

$$perf = \frac{1}{\frac{f}{n} + (1-f)} \times \frac{F}{F_{min}} \tag{4.3}$$

In this equation, n is the number of cores, $0 \leq f \leq 1$ represents the fraction of the application that can be run in parallel. In my model, I assume $f = 1$. Hence, for the experiments, I generate a baseline dynamic power trace with random fluctuation between 0.2 to 1.2 Watts (i.e., dynamic power of a single core operating under the minimum frequency). The execution time and power consumption are, thus, scaled according to the system configuration following Eq. (4.3). Moreover, I assume that all cells, representing the processing cores, have a static power consumption of 0.3 Watts. My assumptions in the power and performance model make static power, dynamic power value, and variation of frequency within the range of those of commercial boards, such as, Xilinx Zynq boards[1].

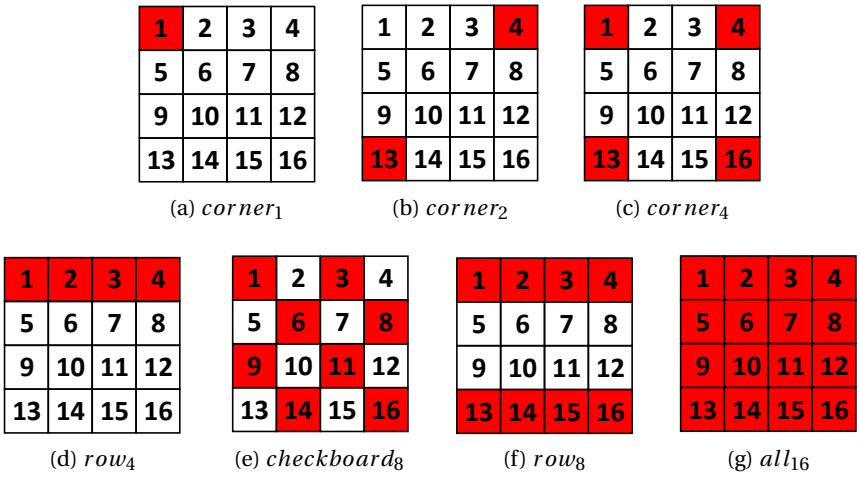
4.5.2 QL-Based Dynamic Thermal Management

Although RL-based solutions are not rare in power and thermal management of multiprocessor systems [126], they have not been used to incorporate fan speed control. In what follows, I explain, in detail, different components of my QL-based solution, i.e., actions, states, reward, and learning process.

4.5.2.1 Actions

The available actions to the agent consist of number of cores (N_c) along with the corresponding mapping, operating frequency (F_c), and fan speed (S). In particular, at each decision point,

[1] https://www.xilinx.com/products/technology/power/xpe.html



(a) $corner_1$ (b) $corner_2$ (c) $corner_4$

(d) row_4 (e) $checkboard_8$ (f) row_8 (g) all_{16}

Figure 4.12 – Number of cores and corresponding mapping

Table 4.3 – State definition

State	Hot spot condition	Gradient condition
1	$\theta_h(t) > \beta_1\theta_{crit}$	$\frac{\Delta\theta_h}{\Delta t} > 0$
2	$\theta_h(t) > \beta_1\theta_{crit}$	$\frac{\Delta\theta_h}{\Delta t} < 0$
3	$\theta_h(t) < \beta_1\theta_{crit}$	$\frac{\Delta\theta_h}{\Delta t} > 0$
4	$\theta_h(t) < \beta_1\theta_{crit}$	$\frac{\Delta\theta_h}{\Delta t} < 0$
5	$\theta_h(t) > \beta_2\theta_{crit}$	$\frac{\Delta\theta_h}{\Delta t} < 0$
6	$\theta_h(t) > \beta_2\theta_{crit}$	$\frac{\Delta\theta_h}{\Delta t} > 0$

the agent specifies a tuple of (N_c, F_c, S) with $S \in \{S_{min}, S_{mid}, S_{max}\}$, $F_c \in \{F_{min}, F_{mid}, F_{max}\}$, and $N_c \in \{corner_1, corner_2, corner_4, row_4, row_8, checkerboard_8, all_{16}\}$. The subscripts in N_c represent the corresponding application mapping on the multiprocessor system, as shown in Figure 4.12. The available actions to the RL agent are not limited to those suggested, and can be any arbitrary parameter and range of values.

4.5.2.2 States

I define the states based on the current hot-spot temperature ($\theta_h(t)$) and its gradient in the last time interval defined as $\frac{\Delta\theta_h}{\Delta t}$. I consider two constant coefficient, $\beta_1 < \beta_2 < 1$, to specify how close the current temperature is to the critical one. Finally, I define six different states as indicated by Table 4.3, where in the first four $\theta_h < \beta_2\theta_{crit}$.

4.5.2.3 Reward Function

Since my goal is to maximize the performance and minimize the fan power under the thermal constraint, I propose the following reward function:

Table 4.4 – Schedule of ϵ based on number of actions to be taken

ϵ	1	0.9	0.8	0.7	0.6	0.5	0.4	0.3	0.2	0.1
# **action**	70	60	50	40	35	30	25	20	15	10

$$R = \begin{cases} -2 & \theta > \theta_{crit} \\ \zeta_1 \frac{perf}{perf_{max}} + (1 - \zeta_2 \frac{P_{fan}}{P_{fan,max}}) & otherwise \end{cases} \tag{4.4}$$

where $0 < \zeta_i < 1$ is a constant denoting the significance of each objective set by the designer, and $\Sigma\zeta_i = 1$. This definition facilitates maximizing the performance close to the maximum performance, while minimizing the fan power and avoiding violation of the critical temperature.

4.5.2.4 Learning Process

I consider an $\epsilon - greedy$ policy where the agent takes a random action with probability ϵ and selects an action from already taken ones with probability $1 - \epsilon$. Since, in the beginning, the agent starts from the *exploration* phase and no action has been chosen, ϵ is equal to 1. In order to let the agent gradually move from the pure exploration to the exploitation phase, I follow the schedule shown in Table 4.4.

4.5.3 Experimental Results and Discussion

As explained in Section 4.5.2, I let the QL agent explore the design space and learn how to apply a sequence of actions to meet predefined objectives and constraints through a simulation framework. In my case of study, the agent learns to use all available frequency and fan speed values with a subset of available number of cores and the corresponding mapping. The results show that the QL agent learns to discard a few of the available mappings, including $corner_1$, $corner_2$, row_4, and row_8, since these actions are either unable to maximize the performance, or may result in thermal violation if applied in specific states. Once the exploration and exploration-exploitation phases finish on the simulation framework, the QL agent is ready to fully exploit its knowledge on a real hardware, in my case, the TTC.

To show how the proposed QL-based DTM with proactive fan speed control ($\pi_{QL,adaptive}$) can enhance performance with minimized fan power, I implement two additional thermal management policies that maximize performance. In $\pi_{QL,max}$, I implement the same QL-based approach while limiting the available actions to only frequency and number of cores with the fan speed fixed at the maximum value. In π_{soa}, I implement a reactive fan speed control policy similar to the default configuration explained by Singla et al. [119]. In this policy, the fan speed is initially set to the minimum value unless the two predefined thermal thresholds are violated. In such cases, the next two larger fan speeds are used. Finally, for performance maximization, I adapt the solution proposed by Hanumaiah and Vrudhula [132].

Table 4.5 – Comparison between different policies at two thermal constraints

θ_{crit}	Metric	$\pi_{QL,adaptive}$	$\pi_{QL,max}$	π_{soa}
$98°C$	#Violations/Duration (s)	0/0.0	0/0.0	2/10.8
	Normalized Fan Power	0.60	1.0	0.60
	Execution Time (s)	4790	4740	5309
	Average Hot Spot ($°C$)	74.9	70.8	73.8
$80°C$	#Violations/Duration (s)	0/0.0	0/0.0	7/39.2
	Normalized Fan Power	0.73	1.0	0.76
	Execution Time (s)	5073	4900	6228
	Average Hot Spot ($°C$)	73.1	67.7	70.5

Table 4.5 compares the number and duration of thermal violations, normalized fan power, execution time representing the performance, and the average hot spot temperature for two different thermal constraints, θ_{crit}. As shown in the table, my approach achieves a DTM policy through which the fan power is reduced by 40% and 27%, respectively when $\theta_{crit} = 98°C$ and $\theta_{crit} = 80°C$, compared to the maximum fan policy, with only 1% performance degradation and no thermal violations. With a lower θ_{crit}, $\pi_{Ql,adaptive}$ has to use higher fan speeds more frequently to avoid thermal violations while maximizing the performance. Compared to the state-of-the-art reactive DTM approach, my solution, as shown in the table, is able to improve the performance 11%, and 19% for the two thermal constraints without any further fan power and, yet, with no thermal violations. π_{soa} causes thermal constraint violations twice since, opposed to $\pi_{QL,adaptive}$ and $\pi_{QL,max}$, it cannot foresee the temperature increase resulting from changes in operating frequency, number of active cores, and workload variation, regarding a particular initial temperature or thermal history. This information, on the contrary, can be learned by RL.

Figure 4.13 shows the first 300 seconds of the hot spot and fan speed traces achieved from different implemented policies sampled at 100 Hz. Each value in these plots could belong to any of the 16 cells on the TTC. As shown in the figure, while $\pi_{QL,adaptive}$ and $\pi_{QL,max}$ do not cause any thermal violations, π_{soa} is unable to avoid thermal violations. For a lower thermal constraints, more violations occur by π_{soa}, i.e., seven for a total duration of 39.2 seconds. In fact, the TTC experiences delayed alleviation of hot spots due to the reactive behavior of π_{soa}. In contrast, $\pi_{QL,adaptive}$ is able to proactively tune the fan speed for each thermal constraint.

4.6 Proposed Workload Allocation of HEVC Streaming on Heterogeneous Systems

Hardware acceleration can contribute to performance enhancement of many application domains, including new trending ones such as HEVC encoding. Thus, several works have considered implementing hardware accelerators for the whole or a single process of HEVC encoders. In this context, Lu et al. [301] address Intra mode decision for real-time HEVC encoder through hardware acceleration. A high-level synthesis design flow that maps the

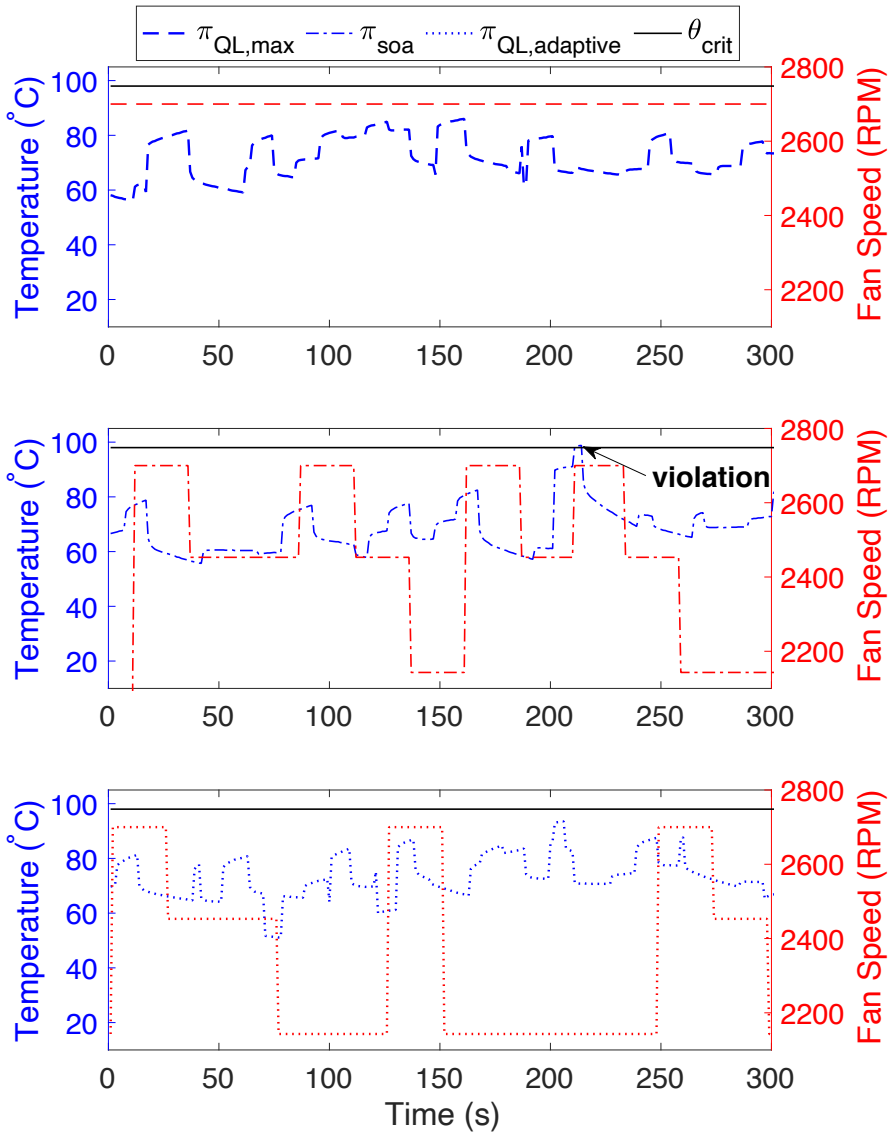


Figure 4.13 – Comparison of hot spot temperature and fan speed obtained from different approaches

intra-prediction block into a SoC-FPGA is presented by Sjövall et al. [302], while different design aspects of a heterogeneous multi-core model of an HEVC intra encoder are assessed by Brandenburg and Stabernack [303]. Nonetheless, intra mode of HEVC standard does not consider frame-to-frame motions, thus, most likely cannot achieve the desirable video compression. Inter-picture prediction, on the other hand, uses motion search and estimations to further compress the output video without any degradation in quality. Motion estimation (ME), however, is a very computationally expensive task. In fact, profiling the Kvazaar HEVC encoder through Valgrind performance profiler [304] indicates that 80% of the execution time is spent on inter-picture prediction when using inter mode encoding. This statistics implies that a wise practice would be to employ heterogeneous multiprocessor systems composed

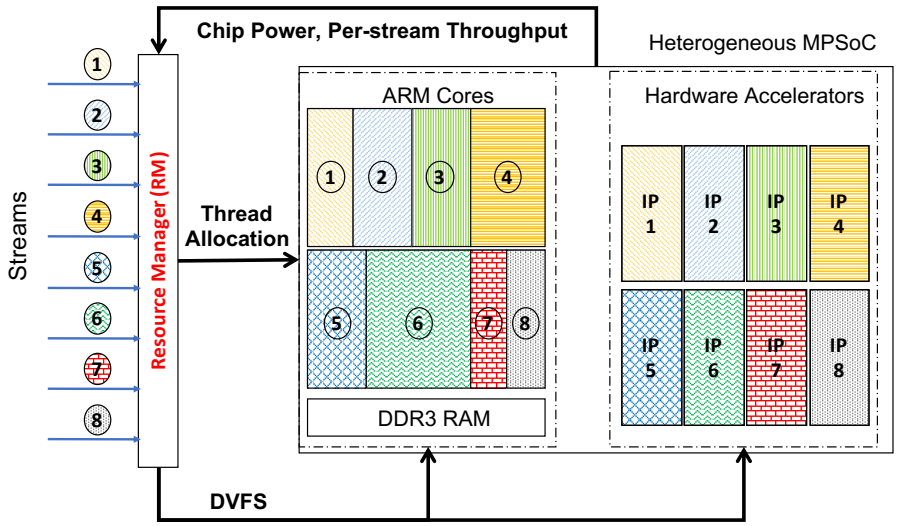


Figure 4.14 – Overall view of HEVC streaming on heterogeneous SoC.

of hardware accelerators to execute ME, and general purpose cores for the rest of encoding process.

A common scenario on video providers servers is that they need to serve more than one HEVC encoding request at the same time. In such a multi-user environment with limited resources, resource management (RM) is vital. Efficient heterogeneity-aware RM for HEVC encoding in MPSoCs requires tackling application configuration, stream allocation and DVFS for both general-purpose cores and accelerators. This requires exploring a very large and dynamic design space. On one hand, streams have different inherent features, such as frame resolution, and need specific encoding configurations, such as search area (SA), which drives motion estimation, that make the workload and resource demand vary from one stream to the other. On the other hand, among the different combinations of streams that could potentially be processed at the same time, there could be many sub-optimalities in regards to core allocation and frequencies of core and accelerator. An exhaustive search in the design space is required to avoid such sub-optimalities. However, conventional offline static approaches cannot guarantee handling the dynamic changes in the environment (e.g., when a new encoding configuration is needed, or video contents change rapidly).

In such scenarios, the large variety of devices requiring different video configurations, together with the high workload variation in terms of number of requests, makes RL a promising approach to deal with such large environment-dependent problems. Figure 4.14 shows an overview of HEVC streaming on heterogeneous SoC.

4.6.1 Problem Definition

With the new applications such as HEVC streaming task allocation and application mapping requires a novel scheme. In these applications, the workload variation is not simply due to

different functions and processes, but rather because of irregular input/content variations. In real-time HEVC streaming, the main goal of video providers is to maintain the minimum required throughput for each stream (e.g., 24 FPS), while maximizing the number of streams that can be encoded simultaneously. Therefore, stream allocation on available resources is of paramount importance as it directly affects the cumulative throughput of the system.

In this section of my thesis, I deal with throughput maximization of HEVC streaming with respect to the total number of streams that can be encoded concurrently on a heterogeneous multiprocessor systems. Each video inputs are not only different in contents and size, but also they do differ in the preset encoding parameters, particularly motion search area, which significantly affects the workload.

I consider heterogeneous MPSoCs as my target multiprocessor system composed of general-purpose cores and hardware accelerators to facilitate the encoding process. The available resources are limited in the target MPSoC and I consider a power cap as the constraint. In particular, I perform the experiments on the Xilinx ZC-706 equipped with a Zynq7000 SoC. The chip comprises a dual core Cortex-A9 ARM processor with a maximum frequency of 1 GHz. The chip also contains FPGA fabric consisting of 350K logic blocks and 19.2 Mb of BRAM. The board comes with a 1 GB DDR3 RAM chip clocked at 533 MHz, and an 8 GB SD card as primary storage. The experimental MPSoC utilizes FPGA-mapped accelerators to speed up ME, which allows HEVC encoding to be performed on the much smaller and more energy-efficient ARM core, while maintaining comparable or improved throughput compared to the Intel processors [158] at a much lower power consumption. Thus, eight accelerators are implemented on the FPGA fabric, allowing up to 8 encoding applications to run simultaneously on the FPGA. These applications will share the 2 ARM cores. The IPs can be individually clocked to 50, 100, 150, and 200 MHz, and the two ARM cores can be individually clocked to 333, 666, 800, and 1000 MHz. I assume the 3 resolutions listed in Table 4.6 and I limit SA to even values between 4 and 12. This already gives me a range of over 1.4 trillion combinations. I also assume that there are always streams queued to be added to the system.

4.6.2 Proposed Framework

The goal is to learn the best allocation of streams to cores, as well as the operating frequency of each core and accelerator, from the total power consumption and the output throughput, for each SA and resolution combination.

My proposed RL-based approach consists of two phases. In the *exploration* phase, once the learning process starts, at each observed state, the RL agent takes a random action from an action pool and calculates the reward, updating the Q-table by Eq. (4.1). Since I aim at learning per-stream SA and resolution, I need to create and keep one Q-table for each resolution-SA pair. The exploration phase for each state-action pair (s_t, a_t) continues until the corresponding

learning rate, which is defined as:

$$\alpha_t(s_t, a_t) = \lambda / Num(s_t, a_t), \tag{4.5}$$

drops below a threshold. In this formulation, $Num(s_t, a_t)$ is the number of observations of the state-action pair, and λ is a constant [126]. Afterwards, the *exploitation* phase begins, where the RL agent stops updating the Q-tables and selects the most appropriate action for each observed state.

In what follows, I describe the state space, the action set, and the proposed reward functions.

4.6.2.1 States

Since the goal of this work is power- and throughput-aware management of heterogeneous MPSoCs, states should include the total power consumption and throughput of each running stream as a performance metric. Hence, the state space is defined as:

$$\mathbf{S} = \{P_{total}, \boldsymbol{Th}\} \tag{4.6}$$

where P_{total} is the total power consumption, and $\boldsymbol{Th}$ is a vector of throughput for each running stream. As explained in Section 4.6.1 and illustrated by Figure 4.14, in this work I implement 8 accelerators on the FPGA and assume each one can be only assigned to one stream. Hence, up to 8 streams can be processed at the same time and the length of $\boldsymbol{Th}$ is 8. This assumption helps increasing the number of concurrent streams running on the platform

4.6.2.2 Actions

The proposed action set includes adding a stream, removing a stream, increasing or decreasing the frequency of an accelerator, and increasing or decreasing the frequency of an ARM core:

$$\boldsymbol{A} = \{Str_+, Str_-, f_{ACC,inc}, f_{ACC,dec}, f_{ARM,inc}, f_{ARM,dec}\} \tag{4.7}$$

While adding a new stream to the existing running streams may lead to a higher total throughput, removing a stream is not desirable. In other words, once an encoding request is accepted, the encoding process must be guaranteed to complete within a certain time. However, I introduce this action since it might be required to reduce power consumption. However, I let the RL agent learn it itself from the rewards corresponding to the state-action pairs.

At each decision step, only one action must be taken so that the reward can properly indicate its worthiness. Therefore, I can only change the frequency of one (and only one) accelerator or core. When the action is $f_{ACC,dec}(f_{ACC,inc})$ or $f_{ARM,dec}(f_{ARM,inc})$, both in exploration or exploitation, I let the agent apply the frequency change only for the stream with the highest

Table 4.6 – Reference throughput with respect to resolution and search area

Resolution	Search Area				
	4	6	8	10	12
704x576	0.62	0.31	0.19	0.13	0.09
1280x720	0.26	0.13	0.08	0.05	0.04
1920x1080	0.11	0.066	0.03	0.02	0.02

(lowest) throughput.

4.6.2.3 Reward Function

The reward function must provide useful feedback about the selected action for the previous state. Since my goal is to minimize power consumption and to maximize performance, I propose a reward function composed of two sub-functions, as follows:

$$r_{tot} = c_1 r_{perf} + c_2 r_{power}, \tag{4.8}$$

where r_{perf} and r_{pow} are the reward functions for performance and power, respectively. I consider equal significance coefficients (c_i) for both rewards. I define r_{perf} such that it encourages the RL agent to choose actions leading to higher performance while avoiding those resulting in any performance loss:

$$r_{perf} = \begin{cases} N_{str}^{\beta} \sum_{i=1}^{N_{str}} Th_i / Th_{ref,i} & \forall i \quad Th_{ref,i} < Th_i \\ -1 & \exists i \quad Th_i < Th_{ref,i} \end{cases} \tag{4.9}$$

where N_{str} is the total number of streams being processed, and $Th_{ref,i}$ is the reference throughput (FPS) for the i^{th} stream. The reference throughputs are calculated on an Intel E5-2690 v4 server [305]. These values represent the highest throughputs achievable on a high-performance homogeneous platform, and ultimately show the gains of my RL-based approach running on a low-power heterogeneous SoC. This server contains 14 cores with a maximum clock speed of 3.5 GHz, 28MB LLC, and 250GB of memory. When calculating the encoder throughput, I tie the application to one core to prevent OS interference. Table 4.6 contains the reference throughput, sorted by frame resolution and SA for default inter-picture prediction mode of Kvazaar encoder.

I experimentally prove that the following inequality holds for all SAs and resolutions given the maximum frequencies for the accelerators and the ARM cores:

$$1 \leq Th_i / Th_{ref,i} < 15. \tag{4.10}$$

This inequality indicates that on my target heterogeneous MPSoC, the achieved throughput

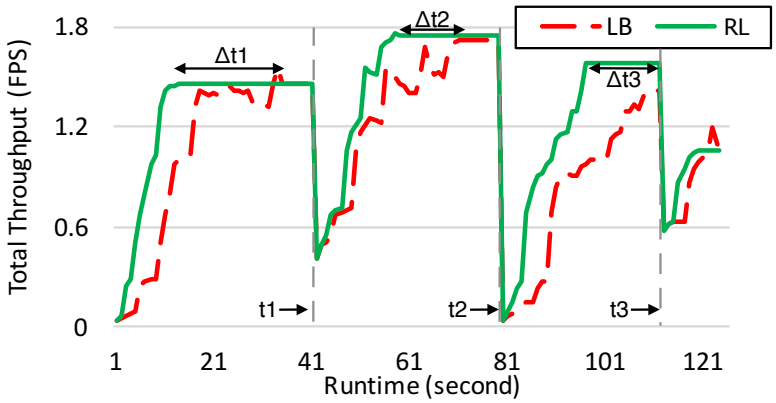


Figure 4.15 – Total throughput of the proposed RL vs. LB over time.

for an individual stream is always greater than the reference one, but not larger than 15 times. However, when considering multiple streams at the same time, the cumulative throughput does not increase beyond 40 FPS. This value, in the best-case scenario (all streams are of the lowest resolution and SA is 4), is always less than 16 FPS. Therefore, since the goal in Eq. (4.10) is to first maximize the number of served streams while meeting their minimum reference throughput, I choose β equal to 5.3 which satisfies the worst-case scenario. This value leads the RL agent to look for increasing the number of concurrent streams, while guaranteeing the minimum required throughput of each individual stream. After fully occupying all available resources, the RL agent takes actions to increase the throughput of each individual stream.

In order to keep power consumption under a user-defined constraint (P_{const}), I propose the following reward function:

$$r_{power} = \begin{cases} 0 & P < P_{const} \\ -2.5 \times 10^6 & P > P_{const} \end{cases} \tag{4.11}$$

On one hand, reducing the power consumption below the constraint should not give a positive reward, since it ultimately results in lower throughput. On the other hand, any violation of the power constraint must add a large enough negative value (here, -2.5×10^6 because the maximum value of the performance reward is always less than $8^{\beta} \times 40$) to the total reward function, such that the corresponding action is avoided.

4.6.3 Experimental Results and Discussion

I stress the system by randomly changing the number of streams running, which represents the effect of users that enter/leave the system. I apply the stress at random intervals ranging from 30 to 40 seconds based on the statistics reported by Delmondo[2], a social video analytics

[2] http://delmondo.co

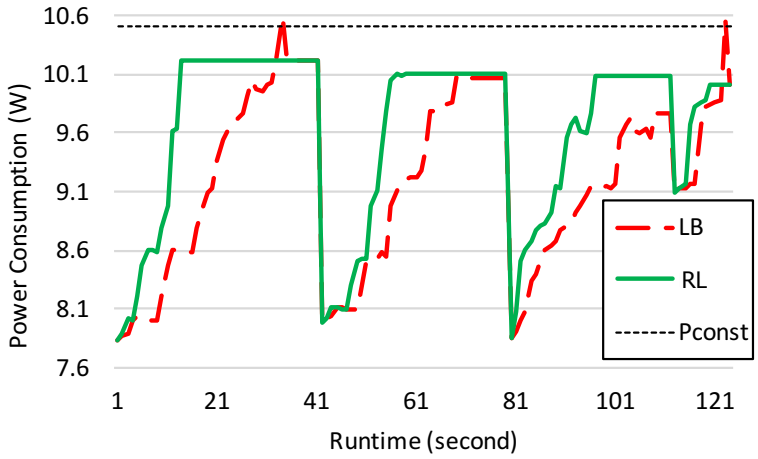


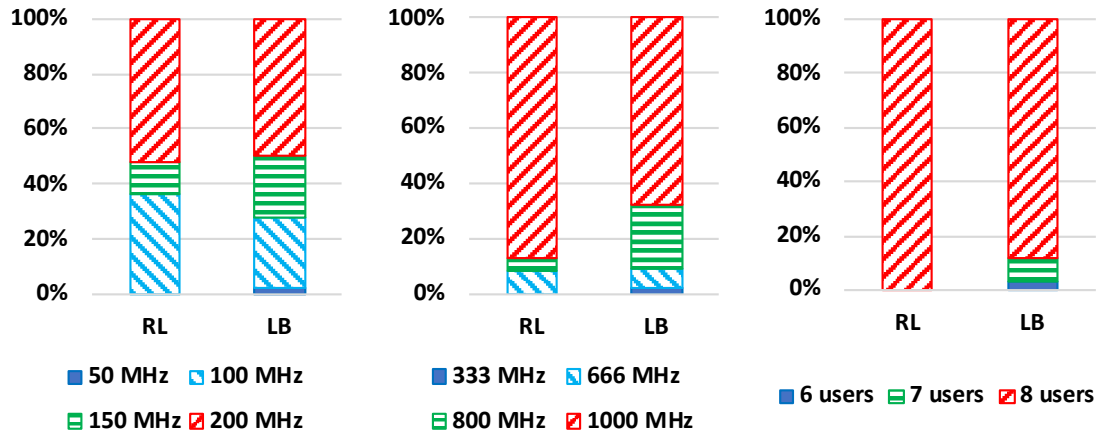
Figure 4.16 – Power consumption of the proposed RL vs. LB over time.

company. Each new stream has its own resolution and SA. In order to provide reliable results, I apply these stress points 10^5 times. For comparison, I implement a load balancing (LB) strategy [189] for stream allocation on the ARM cores. The frequency of all accelerators is set using a heuristic. In particular, at each decision step the frequency is increased by one step, starting from the minimum value, if $P < P_{const}$, to achieve higher throughput.

My RL-based approach improves average throughput by 20% over the LB algorithm when considering the whole runtime (including stress points), as shown in Figure 4.15. In addition, the proposed RL approach demonstrates greater robustness against system dynamism as seen at the 40, 78, and 112 second marks (shown by t_1, t_2, and t_3), where new streams with new requirements replace older streams. On the contrary, the LB algorithm requires more time to maximize throughput when faced to system changes (shown by Δt_1, Δt_2, Δt_3). The reason lies in the fact that the RL agent has learned the optimal actions that maximize throughput for different combinations of SAs and resolutions, while LB first optimizes the minimum throughput of each stream by scaling up the frequency from the initial value, and then adds more streams to increase total throughput.

In the absence of such stress, both systems will eventually reach a steady state in which throughput cannot be further increased. At this point, the total throughput obtained by RL and LB are similar, with RL achieving 7% higher throughput on average. This is because RL is aware of corner cases, which cannot be resolved through heuristics.

Then, Figure 4.16 shows the power consumption of the system for LB and RL, corresponding to the total throughput in Figure 4.15. My approach is able to optimally use the available power budget and increase the total throughput. In addition, since the RL has learned the optimal actions for each power state, the power consumption never violates P_{const}. On the contrary, as the LB algorithm always optimizes for higher throughput and cannot predict power consumption, it may perform an action resulting in violation of the power constraint. This occurs at time intervals of 34, and 123 seconds shown in Figure 4.16. On average, the



(a) accelerator frequency (b) ARM frequency (c) Users served

Figure 4.17 – Percentage of time that each frequency is used and number of users served by RL and LB.

power violation occurs once every 100 seconds for the LB algorithm.

Finally, Figure 4.17a and 4.17b show the percentage of the time that each frequency is used for the ARM cores and accelerators for both the RL and LB approaches. Although the maximum frequency is selected most of the time in both cases, it is not always the optimal choice for all combinations of SA, resolution, number of running streams, power consumption, and per-thread and total throughput. These cases can be distinguished by RL, resulting in serving more streams and increasing the total throughput at lower frequencies, not only after each stress point, but also when no stream is coming into or leaving the system. Figure 4.17c shows the percentage of time that 6, 7, and 8 users could be served by the RL and LB approaches. As shown in the figure, through the proposed RL approach always 8 users can be served, whereas due to the non-optimal usage of the available resources, in 88% of the time LB can serve the maximum number of streams.

4.7 Proposed Joint Application- and System-Level Runtime Management

In this section, I address multi-objective runtime management of multiprocessor systems for multi-user HEVC streaming shown in Figure 4.18. This figure illustrates a two-stage approach composed of an RL-based runtime management for joint optimization of application- and system-level parameters in addition to a heuristic video assignment strategy and migration scheme. My main contribution to the state-of-the-arts in this section, however, is the RL-based runtime management. When the users' encoding requests along with the corresponding videos are received, first, each video and, in particular, each frame, should be assigned to a core in the multi-core server. The videos are assigned to appropriate available cores based

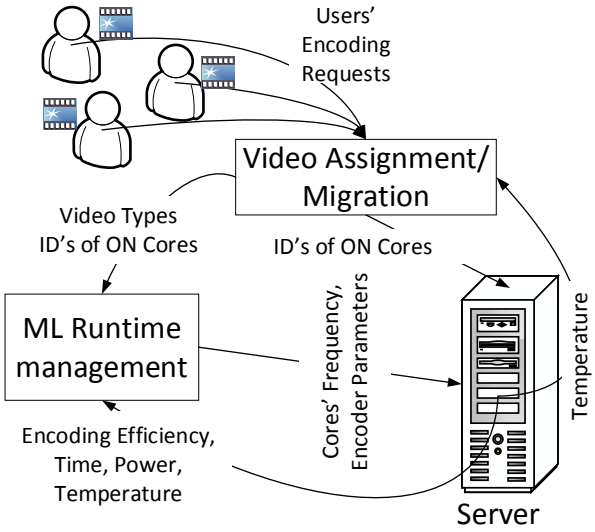


Figure 4.18 – Proposed Approach

on the core temperature and the resolution of the videos. Meanwhile, an RL-based runtime management takes care of adjusting the cores frequencies and tuning the application-level parameters. In particular, I adopt the QL algorithm, which is able to learn the states resulted from the taken actions. This approach suits well multi-core servers where multiple videos are to be processed at the same time.

The proposed approach can be implemented on top of any HEVC implementation, regardless of its specific performance as long as they implement a wide range of control parameters.

4.7.1 Workload Assignment and Migration

Video assignment plays an important role in power consumption, peak temperature, and encoding time. This is especially due to significant difference in thermal characteristics of different video types. Thus, a proper video assignment strategy aware of the exclusive features of videos, such as resolution, can provide reduced peak or average temperature. In addition, a proper video assignment strategy is able to affect the encoding efficiency. In other words, the temperature reduction resulted from the video assignment provides the RL-based runtime manager with more opportunities to increase the encoding efficiency and time by tuning the application- and system-level parameters.

As indicated in Section 4.2.1, higher frequency leads to higher average temperature and hot spots. On the other hand, Figure 4.19 shows the time spent on average for a frame to be processed for each of the seven different videos included in Table 3.1. These results refer to the use of the default *main intra* configuration (average and standard deviation are highly dependent on the encoding configuration and increase considerably if GOP is not one). Although video contents play a significant role on encoding time, the major driver of encoding

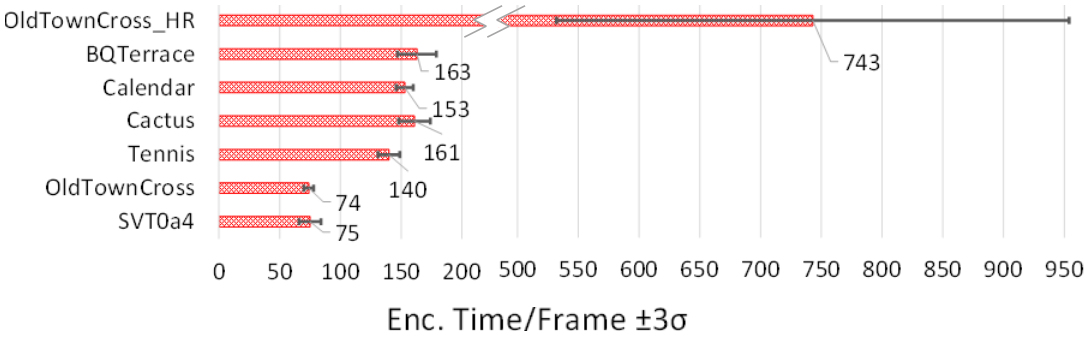


Figure 4.19 – Average encoding time/frame for different videos when encoded by default *Main Intra* configuration

Algorithm 4.1: Video assignment strategy

Input : $N_c, floorplan, \Theta, \boldsymbol{R_s} = \{r_i\}, \boldsymbol{C} = \{c_k\}, r_{new}$
Output: n_c ; `// Index of the assigned core`

1 **forall** k **do**
2 $\quad M_k = \sum_{i}^{Adjacent_k} r_{i,k} + r_{new}$
3 **if** $Num(\underset{k}{\text{argmin}}(M_k)) > 1$ **then**
4 $\quad n_c \leftarrow \underset{j}{\text{argmin}}(\theta_j)$
5 **else**
6 $\quad n_c \leftarrow \underset{k}{\text{argmin}}(M_k)$

time per frame is resolution. Consequently, use of higher frequencies is necessary for videos with higher resolution. However, when the power consumption is constrained by a power budget or a power cap, all the videos running concurrently on the target multi-core server cannot benefit from the highest available frequency. Therefore, in general, videos with higher resolution require higher frequencies so that the desirable frame rate can be achieved. As a consequence, prior to performing DVFS and tuning the encoding configuration parameters a careful video assignment on available resources is vital. For this purpose, I propose a low-overhead video assignment based on the resolution of the streams, as one of the main drivers of performance.

As shown in Algorithm 4.1, I propose a video assignment and migration strategy which takes into account the resolution of the videos and current thermal profile of the chip. In this algorithm, N_c is the total number of cores on the target multi-core server, r_i is the resolution of the video running on the i^{th} core, where $i \in \{1, ..., N_c\}$, $\mathbf{C}$ shows the set of available cores, Θ represents the set of temperature values read from the available sensors, and r_{new} represents the resolution of the unassigned video. Once a new video starts, the best core to process it is determined based on a merit function M (Line 2). For each available core k, the value of

the merit function, M_k, is calculated by summing the resolution of the videos running on the adjacent cores in addition to r_{new} (Line 2). Thereafter, the core whose merit function value is the smallest will be selected as the destination core for the new video (line 6). If there is more than one core with the minimum M value, I choose the one with the lowest temperature (Line 4).

In the proposed video assignment strategy, I first rely on the resolution of the assigned videos (r_i) and the new video (r_{new}), rather than on the instantaneous temperature, or even on the temperature history. Thus, the merit function considers the resolution of videos being processed on the adjacent cores of the available candidate, as well as the resolution of the new video. The higher this sum is, the higher the temperature becomes in the long term. In fact, as the current temperature could be strongly affected by a temporal high resolution video which does not exist anymore (i.e., its executions finished a few seconds ago), it is not possible to assign the new incoming video simply based on the temperature history as it does not guarantee to lower the average temperature in the future. Moreover, this strategy does not decide based on the instantaneous temperature, which is mainly the result of content variation of a video.

Since in multi-user streaming on multi-core servers after proper assignment of new videos other users may leave the server, video migration from one core to the other is vital to satisfy encoding time and temperature requirements. Therefore, I propose to perform video migration every few seconds. In particular, if there is (are) any available resource(s), the video running on the hottest core should move to an available one. Because the resolution plays a major role, I use the same merit function as for the video assignment except for the change of the subscript from "new" to "hot" to indicate that this re-assignment is performed for the video experiencing a high temperature. Unlike the initial video assignment strategy, only those unoccupied ones whose current temperatures are lower than the hottest occupied core are considered as proper candidates. Here, once again, for each idle core, c_k, the value of the merit function, M_k, is calculated by summing the resolution of the videos running on the adjacent cores, $c_{i,k}$, this time, in addition to r_{hot}.

The migration overhead depends on the encoding configuration and the resolution of the video. The latter plays a more important role and induces a maximum performance overhead of 0.2, 0.3, and 1.2 seconds, respectively, for 1280x720, 1920x1080, and 3840x2160 videos studied in this work. Comparing these value with average encoding time for different frame resolutions demonstrates that for the current HEVC test model [179] migration is applicable. Video migration conditions are checked every 2 minutes, and applied if a proper candidate is found, resulting in less than 1% performance overhead in the worst case.

In the proposed video assignment and migration strategy, if all cores are occupied, any new incoming video (encoding request) needs to wait in the non-preemptive queue until an available resource exists. In particular, I assume a first-come first-served policy to assign each video to a core. Hence, from all queued encoding requests, at each decision time (once a

new available core is found), I serve only the first one in the queue. Also, since the proposed strategy can be used in the servers of video providers such as YouTube and Netflix, I assume that no other simultaneous tasks are running on the cores and, hence, no scheduling conflict occurs [306].

4.7.2 RL-Based Runtime Management

Although each HEVC encoder block has its own model, the interaction of the application parameters and input video with the processing platform cannot be characterized by any already known model. Therefore, it is challenging (if not impossible) to apply more conventional power and thermal management strategies, while considering encoding efficiency and execution time. Moreover, when encoding multiple videos with different contents and different encoding parameters, it is practically infeasible to provide a modeled environment with predictable results. Nonetheless, in this context, RL can be used as is the best learning method when the goal is to perform a runtime optimization in a very dynamic environment. This dynamism exists in power and thermal management of multi-core servers for HEVC encoder with its hundreds of possible encoding parameters, as well as several video types and content variation within videos. Indeed, RL is able to figure out interactions of the application parameters, input contents, and output system- and application-level metrics, and choose accordingly the best strategy at each specific moment to maximize the defined objective or set of objectives. As a result, they are promising solutions for dynamic power and thermal management of multi-user HEVC streaming.

Hence, I adopt a QL algorithm that dynamically determines the best possible encoding configuration and per-core frequency to increase video quality and compression, without encoding time degradation under power and temperature constraints. QL, as a model-free algorithm of RL, makes acting optimally in Markovian domains possible by learning from consequences of the previously taken actions.

There are several works proposing alternative multi-objective RL algorithms [307–309]. In particular, Pareto Q-Learning (PQL) [309] provides an alternative multi-objective QL algorithm. However, the proposed PQL algorithm only suits problems with terminal states, where a sequence of actions at one episode of time leads to a final state. In my problem, however, at each decision time, only one action can be applied. Moreover, content variation within the videos, as well as the actions taken for other videos running on the multi-core server, can considerably affect the current state of each video, which makes impractical the definition of a terminal state at runtime.

4.7.3 QL-Based Quality-Aware Power and Thermal Management

Figure 4.20 shows a general description of my proposed approach. I consider three phases for the QL algorithm including *exploration*, *exploration-exploitation*, and *exploitation*. In the

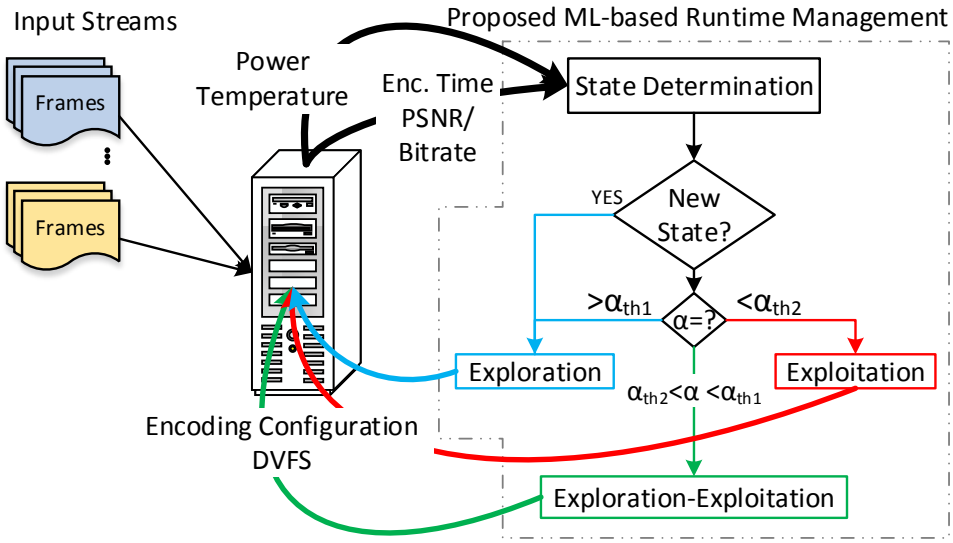


Figure 4.20 – RL-based approach block diagram

exploration phase, once the first frame arrives, an action is selected randomly from an action pool, including all available configuration modes and operating frequencies. Therefore, the state transitions from an initial state to a new one observed by the RL agent. Thereafter, the new Q-value corresponding to the selected action and the initial state is calculated.

As the second frame arrives, another random action is selected. However, this time, the action is selected from a subset of the initial action pool. The previous randomly selected action, in fact, blocks some of the configuration modes to be selected, for one or several incoming frames, since their reward is not observed instantly. For instance, when a configuration mode including GOP of size 8 is selected, observing the reward for this specific part of the whole selected configuration mode must be postponed until the 8^{th} frame is encoded.

The *exploration* phase for each pair of state-action continues until the learning rate decreases down to a predefined threshold. I define the learning rate as a function of number of state-action observations as:

$$\alpha(s_t, a_t) = \lambda / Num(s_t, a_t), \tag{4.12}$$

where λ is constant and set to 0.3, to increase the learning speed. In addition, α_{th1} and α_{th2} are experimentally set to 0.2 and 0.1 respectively. $\alpha(s_t, a_t)$ determines the learning rate and shows if the state-action pair of (s_t, a_t) has been sufficiently observed or not. As discussed in Section 4.3.3 , to facilitate transitions between my proposed learning phases, I followed the varying learning rate approach, which depends on the number of observations of each state-action pair. Thus, I empirically explored the best values with different video encoding inputs and configurations, and I found $\lambda = 0.3$ and $\alpha(s_t, a_t) = 0.3/1$. These values satisfy convergence conditions [235] and provide quick learning, and are similar to those found by state-of-the-arts in the literature [126].

In the *exploration-exploitation* phase, the agent keeps updating the Q-values, while selecting the best possible action. In the *exploitation* phase, the learning process stops and the agent relies on the obtained rewards and available Q-table to choose the most appropriate action for each state. However, if a new state is observed, the *exploration* phase restarts, but only for this newly observed one. This is, in fact, one of the major advantages of my work when compared to state-of-the-arts , such as TONE [276]. TONE statically extracts the Pareto optimal curves with respect to the encoding parameters. Hence, TONE is not able to select the best encoding configuration if a different video type affects the system and the application states differently. In fact, TONE uses the complexity difference between the current and previous frames to predict the future temperature. It categorizes this complexity difference into three classes and then uses the temperature prediction in the optimization problem. However, by evolving more complex videos, either in type or contents, the fixed three complexity classes defined by their work will fail to provide an accurate prediction. In contrast, learning from new states and actions is an inherent feature of QL.

In the following subsections I define the states and available actions, as well as the reward function.

4.7.3.1 State Definition

Power consumption. The instant total power consumption of the target multi-core server, ranging from static power, P_{static}, and a user-defined power cap, P_{cap}, are split into n_{power} intervals to create the power state subset.

Temperature. Temperature varies from ambient temperature ($\theta_{ambient}$) to critical temperature. I control the peak temperature of the multi-core server and define a temperature constraint (θ_{const}) below the critical temperature (θ_{critic}). The interval between $\theta_{ambient}$ and θ_{const} is divided into n_{θ} intervals.

Power consumption and per-core temperature data are measured directly from the multi-core server (see Section 4.7.4).

PSNR. Usual PSNR values for lossy video compression range from 30 dB to 50 dB, for a bit depth of 8 [310]. This range is divided into n_{psnr} intervals to constitute the quality state subset. However, not all these values can be representative of an acceptable video quality. Hence, the state subspace related to PSNR is defined as follows: (≤ 35), (35-36), (36-37), (37, 37.5), (37.5-38), (38-38.5), (38.5-39), (39-39.5), (39.5-40), (40-41), (41-42), and (42 ≤). The non-uniform PSNR ranges used for state definition originate from my experiments with different videos and PSNR values, where human vision was able to distinguish well between different qualities based on the PSNR value.

Bitrate. The achievable bitrate varies from a few hundreds of kbps to several thousands of kbps. Nonetheless, a target bitrate (BR_{targ}) for each video type is defined based on the required link speed. Thus, the following subset is used in this work: ($\leq 0.75BR_{targ}$), ($0.75BR_{targ}$-BR_{targ}),

(BR_{targ}-1.25BR_{targ}), and (≥1.25BR_{targ}).

Encoding time. The current version of reference software (HM 16.3) does not provide real-time encoding. As a result, for different video types and contents the encoding time varies considerably. Therefore, states must be defined regarding a unique reference for each individual video type. For this purpose, I consider the average encoding time per frame obtained from *Main Intra* configuration file for each of test sequences and let them be representative of all similar video types. Therefore, the following state subspace can be defined:
(≤ 0.7T_{ref}), (0.7T_{ref}-0.8T_{ref}), (0.8T_{ref}-0.9T_{ref}), (0.9T_{ref}-T_{ref}), (T_{ref}-1.1T_{ref}), (1.1T_{ref}-1.2T_{ref}), (1.2T_{ref}-1.3T_{ref}), (1.3T_{ref} ≤).

The reference software [179] reports PSNR, bitrate, and time. I use these data to build the application-level states. Moreover, although the overall state set is extremely large, the RL agent does not have to explore all these states, since a great number of them do not occur. For instance, it is not possible that the highest PSNR and the lowest bitrate are observed at the same time for a specific frame.

4.7.3.2 Action Pool and Action Set Definition

The action pool proposed in this work consists of the most effective encoding configuration modes in conjunction with the available CPU frequencies. Table 4.1 shows the design parameters and the corresponding values considered as the available actions to the RL agent.

Even with the constrained action set, there are 684 different combinations of encoding parameters and operating frequencies that can be applied for a single video at a specific state if static profiling approach instead of machine learning is used. In particular, when GOP size is one, QP, CU, and frequency can take all their available values of Table 4.1 (i.e., 4, 3, and 3, respectively) and there will be no choice for the rest, while if GOP is not one (8 or 16) action could be any combination of the encoding parameters and the available frequencies. Due to content variation within a single video, the application-level and system-level state changes constantly at runtime. This requires profiling every single frame, which means running each frame with all 684 different combinations of actions to figure out the best one. More importantly, configuring the encoding parameters is even more challenging when multiple videos running on a multi-core server are taken into account. This is because power, temperature and encoding time objectives are considerably affected by operating frequency, and available power and temperature budget. Moreover, when several videos are running on the server, the outcome of one of those 684 different combinations of encoding parameters and frequencies will be strongly affected by the values chosen for other videos, thus, an exhaustive profiling is required to include all possible combinations of encoding parameters and frequencies for all encoded videos in parallel. In addition, all combinations of frames that can potentially be running simultaneously should be also profiled carefully.

Different combinations of system and application parameters may ultimately result in a

unique application- and system-level output. Although a few works such as [189] tried to partially model these outputs based on application parameters, these models are very platform-dependent and only take a few encoding parameters. Nevertheless, RL is able to consider any interaction between arbitrary encoding and system parameters on any arbitrary platform as their effect will cumulatively appear in the defined states.

4.7.3.3 Reward Function

The proposed reward function must provide a proper feedback from the selected action for a previous state. Since I look for a solution to take bitrate, PSNR, power consumption, temperature and processing time under control, I propose a reward function composed of five sub-functions, one for each of these parameters.

The higher compression obtained by HEVC, in comparison with other video encoding standards, is one of the most important features to be maintained. However, the best achievable bitrate differs from one format to another. Thus, I assume a specific target bitrate for each video type. In particular, I propose the following reward function:

$$R_{br} = \begin{cases} -aBR^2 + bBR & BR < BR_{targ} \\ \frac{-c \times BR}{BR_{targ}} + d & BR > BR_{targ} \end{cases} \tag{4.13}$$

where BR shows the bitrate. The maximum reward is given to $BR = BR_{targ}$ and it degrades faster when the bitrate is larger than the target value. The quadratic part provides a larger difference between the granted rewards from point to point when the attained bitrate is far from the target. This difference decreases as the obtained bitrate approaches the target, letting the agent take into account other reward functions. $a = 1/BR_{targ}^2$ and $b = 2/BR_{targ}$ provide such behavior by making R_{br} local maximum at $BR = BR_{targ}$. The decreasing slope for the linear part is defined by c. I experimentally found c equal to 2. Therefore, d is equal to 3 to provide continuity for the reward function.

The reward sub-function corresponding to PSNR is defined as:

$$R_{psnr} = a \times e^{(PSNR/PSNR_{max})} - b \tag{4.14}$$

where $PSNR_{max}$ is 50 dB as discussed in Section 4.7.3.1. Also, a and b are constants and defined such that the maximum value obtained from this sub-function is one while the minimum (assuming $PSNR_{min} = 30\ dB$) is zero. The exponential reward helps getting higher rewards as the PSNR approaches to the maximum value.

Furthermore, as I seek shorter encoding time the proposed reward sub-function is as follows:

$$R_T = 1 - T/T_{ref} \tag{4.15}$$

where T is the encoding time of the frame, and T_{ref} is the reference time (see Section 4.7.3.1).

T_{ref} is a user-defined value and varies depending on the frame resolution. In this work, I assume 75, 150, and 750 seconds, respectively, for 1280x720, 1920x1080, and 3840x2160 resolutions.

In order to meet the predefined power cap, the reward function provides a negative value, which is large enough to cancel probable positive rewards attained by other sub-functions, if the constraint is not met (also applied for temperature reward). Higher Q-values are given to those state-action pairs leading to lower power consumption. Hence, the reward sub-function is:

$$R_{power} = \begin{cases} -4 & P > P_{cap} \\ P_{static}/P & P \leq P_{cap} \end{cases} \tag{4.16}$$

where P is the total power consumption of the multi-core server.

The temperature reward sub-function must facilitate preventing any state-action pair resulting in temperature higher than the peak temperature constraint. Thus, I employ a reward sub-function defined as:

$$R_{\theta} = \begin{cases} -4 & \theta > \theta_{const} \\ e^{(\theta_{ambient}-\theta)} & \theta \leq \theta_{const} \end{cases} \tag{4.17}$$

When the temperature is below the constraint, the reward exponentially increases as it approaches towards the ambient temperature. Nonetheless, reaching the ambient temperature is ideal and not a goal. Therefore, the values corresponding to this reward function are comparatively small and dominated by other reward functions.

The proposed reward functions for power and temperature only depend on the ambient temperature and the static power consumption which may differ for different systems and environments. However, it does not affect the validity of the proposed approach when the environment changes. Therefore, only a simple characterization phase for different systems with respect to the static power consumption and thermal design power (TDP) is required, which is not a complicated task for today's systems.

Finally, Eq. (4.18) forms my total reward function:

$$R_{tot} = c_1 R_{br} + c_2 R_{psnr} + c_3 R_{\theta} + c_4 R_T + c_5 R_{power} \tag{4.18}$$

The proposed reward function simply sums all the sub-functions without considering interactions among them. First, I recall that although the output PSNR, bitrate, encoding time, as well as the system power and temperature, all vary with changes in the encoding configuration, they are also strongly dependent on the video contents. Hence, modeling the total reward function with the interactions among its sub-functions will only add to the complexity of the reward function, providing only minor gains. Moreover, the interrelation of power and temperature in SoCs, and particularly in multi-core servers, is not straightforward, due to heat sharing. As a consequence, I consider both temperature and power independently in the reward sub-functions. Furthermore, the temperature reward function in my formulation

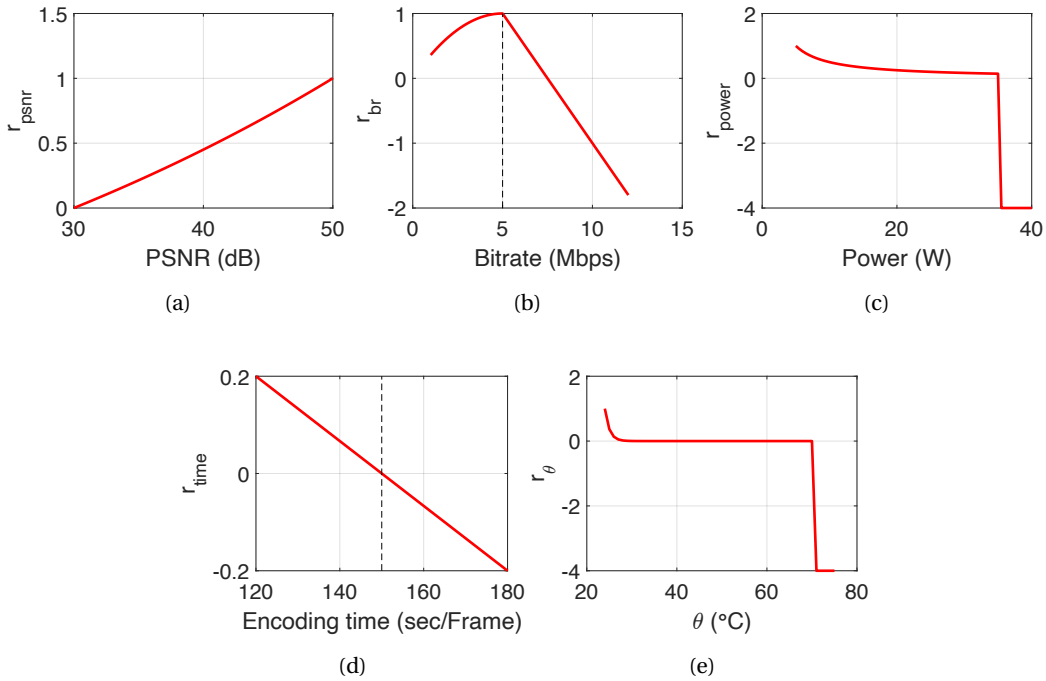


Figure 4.21 – Reward functions of a) PSNR, b) bitrate, c) power, d) encoding time, and e) temperature

affects the total reward function less significantly than what the power does. In fact, the main role of R_θ is taking care of the peak temperature of each individual component, while on the other hand, R_{power} is in charge of total power consumption of the chip.

The maximum value of all reward sub-functions is normalized to 1. When the power and temperature constraints are violated, a sufficiently large negative reward is considered so that the corresponding action can be discarded from future decisions.

Coefficients c_1 to c_5 are introduced to manipulate the effect of each reward sub-function. These constants are in charge of tuning the total reward to emphasize more on a particular sub-function. For my setup, since I am using sub-functions with different behaviors for each reward, this objective is already fulfilled. Thus, I set all these constants equal. With these equal weights, as shown in Section 4.7.5.4, the outcome solution is only marginally outperformed by the optimal solution. Yet, I may weigh specific constants more than the others to emphasize a specific objective based on my requirements posed by the system and/or the application.

4.7.4 Experimental Setup

4.7.4.1 Experimental Platform

The proposed runtime power and thermal management approach can be used for any platform architecture and HEVC implementation since it focuses on leveraging application-level parameters. As a result, the underlying architecture or platform, regardless of its type, needs to deal with an optimized application code, which ultimately leads to improved encoding efficiency and time. For this part of my thesis, I perform the experiments on an Intel S2600GZ server running CentOS 6.5. The server includes a 6-core SandyBridge-EP processor. The platform supports per-core DVFS and a frequency range from 1.4 GHz to 2.0 GHz spaced by 100 MHz, as well as a turbo boost frequency of 2.4 GHz. The server comes with 32 KB instruction and data L1, 256 KB private L2, and a 15 MB shared L3. I use RAPL to collect power measurements of CPU and DRAM, while IPMI is used to gather CPU temperature sensor measurements. In addition, the server is equipped with the default PWM-based thermal management mechanism of commercial servers, which keeps fan speed at low speed until 75°C, and then increases it to keep the CPU temperature below this value. Since cooling power is a cubic function of fan speed, I set the CPU thermal constraint equal to 70°C, in order to provide more power saving. In addition, the CPU power constraint is set to 33 W. Finally, given that the room temperature of the server is fixed at 24°C, I use the following temperature state subset: $(\theta_{ambient}, \frac{\theta_{const}+\theta_{ambient}}{2})$, $(\frac{\theta_{const}+\theta_{ambient}}{2}, \theta_{const})$, and $(\geq \theta_{const})$. I also split the range of (P_{static}, P_{cap}) into 5 equal portions. Although all temperature data for the runtime management are gathered via the available per-core thermal sensors, in order to visualize the thermal profile of the chip, I use the power traces measured on the server and an approximate floorplan available online for 6-core SandyBridge processors[3] to feed the 3D-ICE [159] simulator. Overall, the applicability of the proposed RL-based solution does not depend on these thresholds I consider in my thesis. Nonetheless, according to these thresholds the amount of improvements provided by the RL agent can be different. In this section, I have considered rather small values as the power and thermal thresholds. Such values are indeed selected to put the RL agent under pressure and assess the proposed solution in extreme cases. Obviously, if any of the constraints considered in my work are released the RL agent is able to further improve other objectives.

Finally, in order to perform per-core DVFS via the OS, I change the governor of the CPU frequency scaling to "*userspace*" via *cpupower* utility. I perform video assignment and migration through *taskset* utility. Moreover, since the predefined power and thermal budgets are lower than those defined in the CPU datasheet[4], the default shutdown mechanism or any other default DPM and DTM schemes are not invoked by the OS. Hence, the OS does not intrude my proposed approach.

[3] http://www.anandtech.com/show/5091/intel-core-i7-3960x-sandy-bridge-e-review-keeping-the-high-end-alive

[4] https://www.intel.com/content/www/us/en/motherboards/server-motherboards/server-board-s2600gl-gz.html

4.7.4.2 Compared Approaches

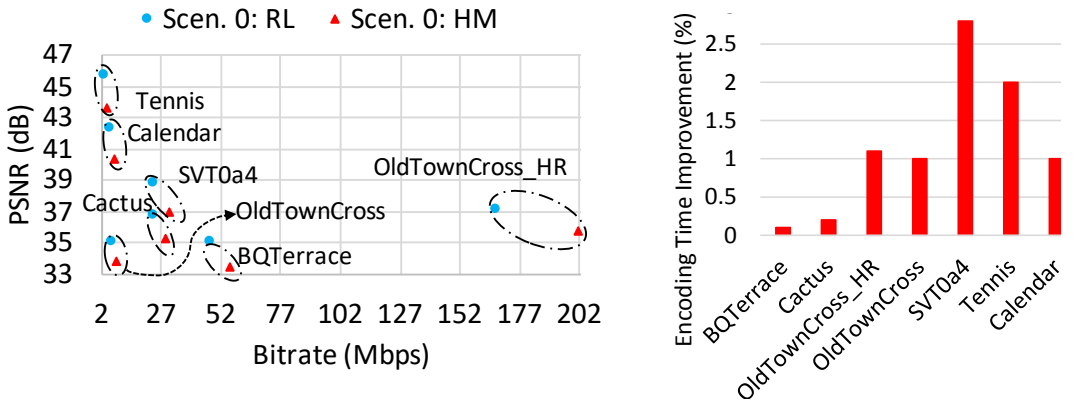
To make my solution as general as possible, I implement my RL-based approach and TONE [276] on top of the HEVC test model HM 16.3 [179]. In order to provide a fair comparison with the reference software, with respect to power and temperature, I implement an RL-based power and thermal management approach [311], which outperforms other recent pro-active approaches (such as TAPE [312]), on top of the reference software, and call it HM*.

Although there are several works for power and thermal management of multimedia workloads, I choose TONE for comparisons as it is the only existing thermal management work for HEVC encoders that uses application-level parameters. As I aim at approaching towards a target bitrate, for a fair comparison, I adapt TONE [276] so that it avoids reducing the bitrate far below the target.

4.7.4.3 Studied Scenario

In order to provide a better insight in how my RL-based approach is able to dynamically manage the output encoding efficiency and time, I compare it with HM 16.3 reference software. In this so-called Scenario 0, I study all test sequences while they are running alone with the maximum available core frequency. Since a single video running on my experimental platform does not violate the power and thermal constraints, power and thermal management over HM is bypassed. Besides, the proposed RL-based approach can now only consider the reward functions of encoding efficiency and time.

Thereafter, I evaluate the efficacy of the proposed approach against HM* (see Section 4.7.4.2) and TONE [276] in three scenarios. In the first scenario, I assume that all cores are fully utilized by receiving instances of the same video. In the second scenario, I assume a more realistic case where videos randomly start on cores and finish. In this scenario, I assume multiple instances of the same video. In the third scenario, I assume the same scenario as the second one but with different videos. In last two scenarios, in order to assign the videos to the cores I take advantage of the proposed video assignment strategy while DVFS is performed by my RL-based approach. Since in the last two scenarios videos randomly start and finish, I perform the experiments 100 times to obtain statistically significant results. At each run, I consider the average PSNR and bitrate for each instance separately. Finally, the average of all results over these 100 runs are reported as the final obtained gains. In all scenarios, to provide a fair comparison with HM, adaptive search range (ASR) and adaptive QP selection (AdaptiveQP) options are enabled, while the target bitrates (TargetBitrate) are set to those specified in Table 3.1.



(a) Encoding efficiency

(b) Encoding time

Figure 4.22 – ML vs. default HM in Scenario 0

4.7.5 Experimental Results

4.7.5.1 Evaluation of Encoding Efficiency and Time

In what follows, a scenario-wise discussion on the experimental results is presented. While Figure 4.22 shows the results of Scenario 0, Figure 4.23 and Figure 4.24 show the encoding efficiency and encoding time, respectively, for the other scenarios.

Scenario 0. Figure 4.22a shows the PSNR and bitrate obtained by the default HM and the RL-based approach for different test sequences. As shown in the figure, the proposed approach leads to larger improvements (i.e., shift to the upper left corner) in encoding efficiency when there is more frame-to-frame content variations, as in the case of *Tennis* and *SVT04a* (see Figure 3.2 to compare content variations within a single video). Such improvements come with a small encoding time enhancement, as shown in Figure 4.22b. This encoding time improvement comes from the more efficient and intelligent adaptation of encoding parameters compared to when ASR and AdaptiveQP options are enabled in HM. In fact, my RL-based approach is able to more efficiently find the most appropriate encoding parameters based on the contents of the video.

Scenario 1. In the first scenario, where all cores are occupied by instances of the same video, variations between frames result in a great opportunity for reducing the power consumption and, hence, the average temperature of the target multi-core server, while improving the encoding efficiency in terms of video quality and compression. The improvement in encoding time compared to that of HM* in Scenario 1 lies in the fact that my RL-based power and thermal management uses DVFS and encoding parameters jointly, while the power and thermal management scheme of HM* leads to application of lower core frequency for some frames. Thus, it is not able to further improve the encoding time by tuning the encoding parameters. TONE, on the other hand, suffers the most from encoding time degradation as

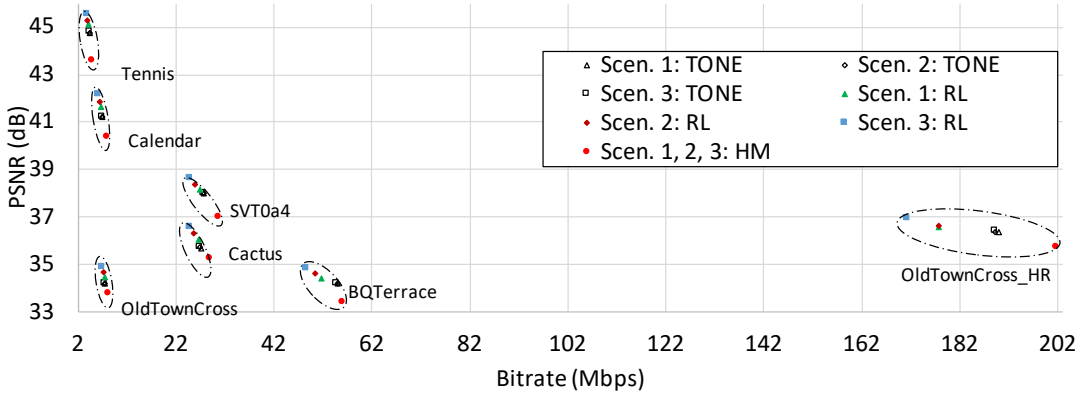


Figure 4.23 – PSNR vs. bitrate achieved by RL, TONE, and HM* for all test sequences and scenarios

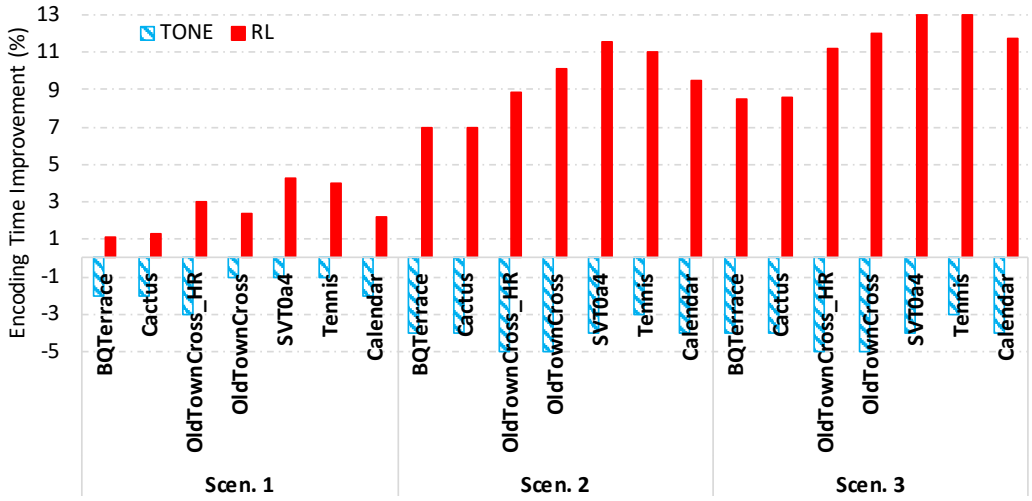
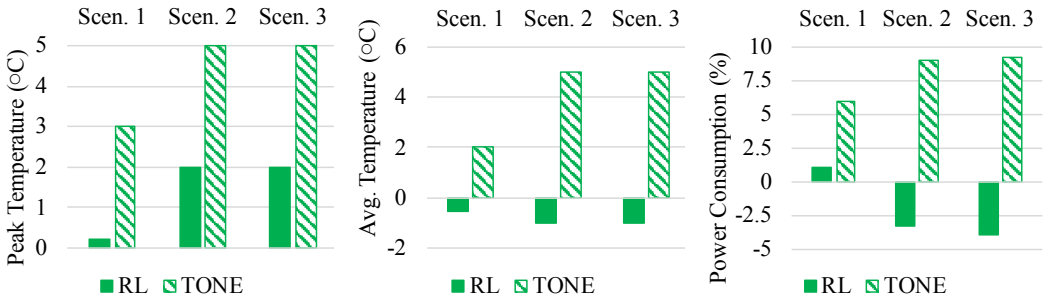


Figure 4.24 – Encoding time of RL and TONE compared to HM*, for all test sequences and scenarios

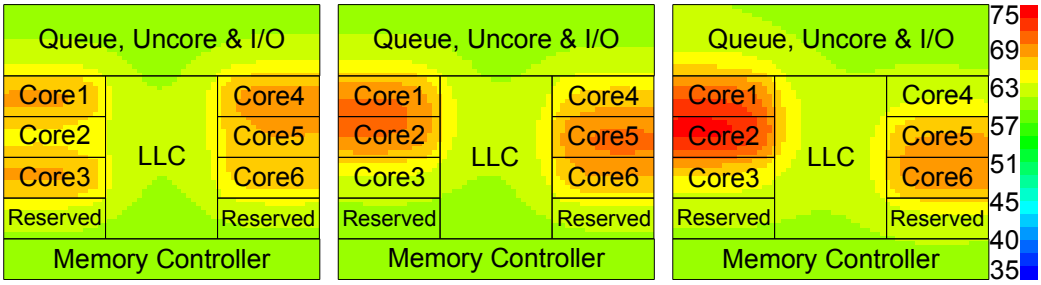
neither does it take advantage of intelligent DVFS, nor tunes the encoding parameters with respect to the output encoding time. Nonetheless, such an improvement provided by RL comes with the cost of less encoding efficiency compared to that obtained through Scenario 0, since the RL agent compensates the encoding time through adapting the computational complexity of the encoder which ultimately results in PSNR loss and less compression.

Scenarios 2 and 3. The main benefit of RL can be seen in the second and third scenarios, which are closer to the real cases of video providers' servers. In Scenario 2, HM* and TONE, unaware of the available potentials due to changes in the number of videos being processed, fail to improve encoding efficiency. On the contrary, my RL-based approach succeeds to increase the video quality (PSNR), and decreases the bitrate. More importantly, the proposed approach further improves the encoding time in comparison with HM* and TONE due to the



(a) (b) (c)

Figure 4.25 – (a) peak temperature, (b) average temperature, and (c) power consumption of the proposed approach (RL) and TONE compared to HM*



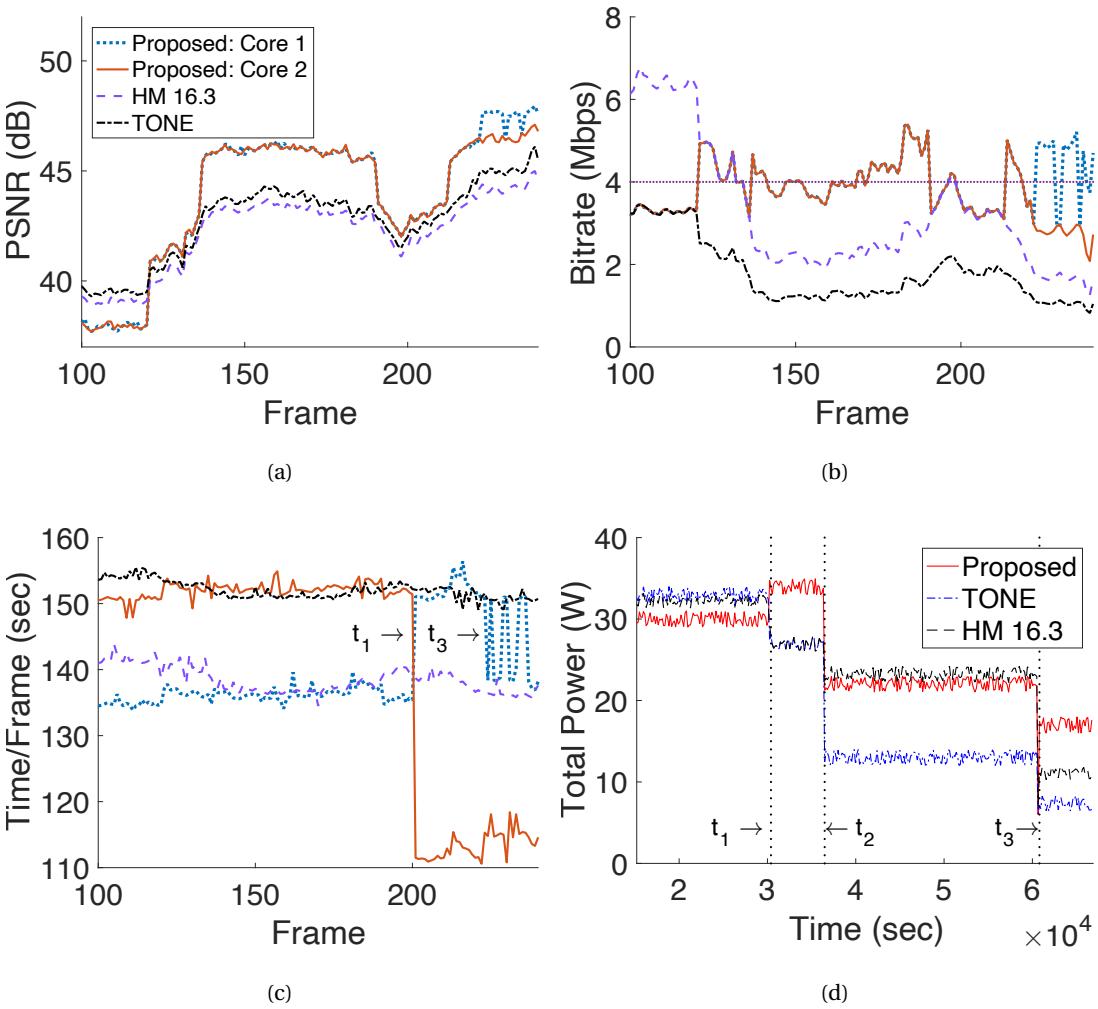
(a) (b) (c)

Figure 4.26 – Thermal map (°C) of the third Scenario for (a) RL and video assignment, (b) only RL, and (c) TONE

same reasons already explained for Scenario 1. In particular, encoding time enhancement comes from more frequent opportunities of intelligently using higher operating frequencies, especially when some videos leave the server. Indeed, thanks to another video leaving the server, the increased available power budget allows other cores run with higher frequency and/or RL agent tunes the encoding parameters of the videos.

The PSNR and compression obtained in these scenarios are higher than those in Scenario 1, but smaller than in Scenario 0. In contrast, as shown in Figure 4.23, the obtained PSNR and bitrate points by TONE are almost overlapping each other meaning that very marginal improvements in encoding efficiency are obtained.

The same trend is observed in Scenario 3, where different videos with different contents and resolutions result in more opportunities of well-tuning the encoding parameters. In other words, when the number of videos and their contents change, the system state changes and, hence, proper decision taken by my RL-based approach leads to improving encoding efficiency and time. However, similar to the previous scenarios, HM* and TONE are unaware of such variations, thus, they fail to achieve any improvement in encoding efficiency.



(a) (b)

(c) (d)

Figure 4.27 – Frame-by-frame results for Tennis: proposed RL-based approach (Core 1 and Core 2) versus TONE and HM 16.3 (the best core)

4.7.5.2 Discussion on Power and Thermal Awareness

Figure 4.25 shows power consumption, average temperature, and peak temperature achieved through my proposed RL-based approach compared to HM* and TONE [276], for scenarios 1 to 3. The RL-based approach is able to reduce the average temperature for all scenarios compared to HM* and TONE. However, while peak temperature decreases compared to TONE, the peak temperature reduction is not as large as the reduction in the average temperature. This is because the peak temperature occurs mostly due to a rapid change in the video contents, or mainly in the second and the third scenarios, because of releasing an additional video on the target multi-core server, resulting in a spike in the temperature.

Finally, Figure 4.26 shows the thermal profile of the chip at specific time for the third scenario. At this given moment, the total power consumption obtained from TONE, RL-based approach

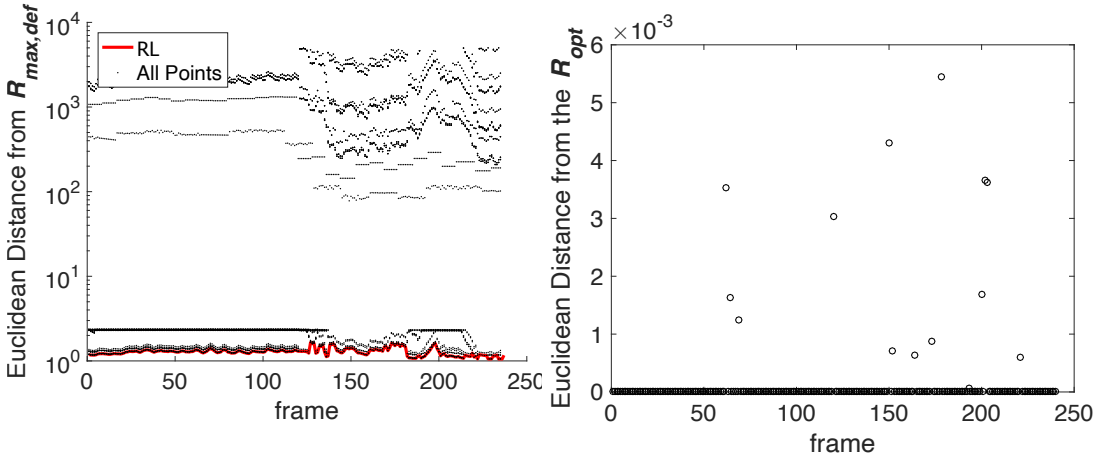
with, and without the proposed video assignment are all at the maximum level. As shown in Figure 4.26, the best thermal profile, in terms of lower temperature and less thermal gradients is achieved when the proposed RL-based approach is accompanied by the resolution-aware video assignment. This implies that although runtime management of multi-user encoding on the multi-core server is necessary, a proper thermal-aware video assignment is also vital. In particular, my proposed video assignment and migration strategy leads to peak temperature reduction. Such a thermal profile eventually provides more opportunities for the QL agent to enhance the encoding time and efficiency.

4.7.5.3 Frame-by-Frame Evaluation of the ML-based Approach

Figure 4.27 illustrates the second scenario, frame-by-frame, for *Tennis* where some videos leave the server. For the sake of clarity, the behavior of only 2 cores is shown for my approach, while only the curve related to the core with the best behavior is depicted for TONE and HM 16.3. At time t_1 (see Figure 4.27b, one video leaves the server. TONE and HM 16.3 suffer from significant PSNR loss and deviation from target bitrate (Figure 4.27a and 4.27b). My approach, however, instantly reacts and keeps the same power state by increasing the frequency of other cores to decrease the encoding time (Figure 4.27c). In addition, it finds an opportunity to exploit other more efficient encoding configurations. The same occurs at t_2 and t_3, respectively. The RL-based approach improves the encoding time on core 1 and 2 during the whole encoding process by 10% and 27%, respectively. As these results show, my RL-based solution outperforms TONE [276], especially when the temperature is below its constraint. This is due to the fact that TONE [276] starts seeking an appropriate encoding configuration only if the temperature exceeds the threshold. However, when only few cores are active, temperature drops, hence, granting an opportunity to RL agent for increasing the frequency and/or using another appropriate encoding configuration.

4.7.5.4 Overhead and Performance of RL-Based Approach

Although I deal with a multi-objective problem, I use a scalarization function over the predefined reward sub-functions. Despite its simplicity, scalarization works well for my specific target problem and the defined reward functions, and my solution is very close to the optimal one. To show how close my (sub-)optimal solution is to the ideal solution for each frame (decision point), I use the Euclidean distance metric. Based on the predefined reward sub-functions, an ideal solution would be the one giving a vector of reward ($\boldsymbol{R}_{def,max}$) values equal to (1, 1, 0.2, 1, 1), i.e., the maximum reward for each individual reward functions as shown in Figure 4.21, considering the reward vector as $\boldsymbol{R}(r_{br}, r_{psnr}, r_T, r_\theta, r_{power})$. At each current state, there are a finite number of actions available, each of which may ultimately result in observing a unique state, giving a reward vector $\boldsymbol{R}_i$, where $1 \leq i \leq N_R$ and N_R is the total number of possibly observable different reward vectors. Figure 4.28a shows the Euclidean distance of several possible reward vectors (dots) for each frame (decision point) for *Tennis* test sequence in the second scenario. The Euclidean distance obtained through my RL solution and its

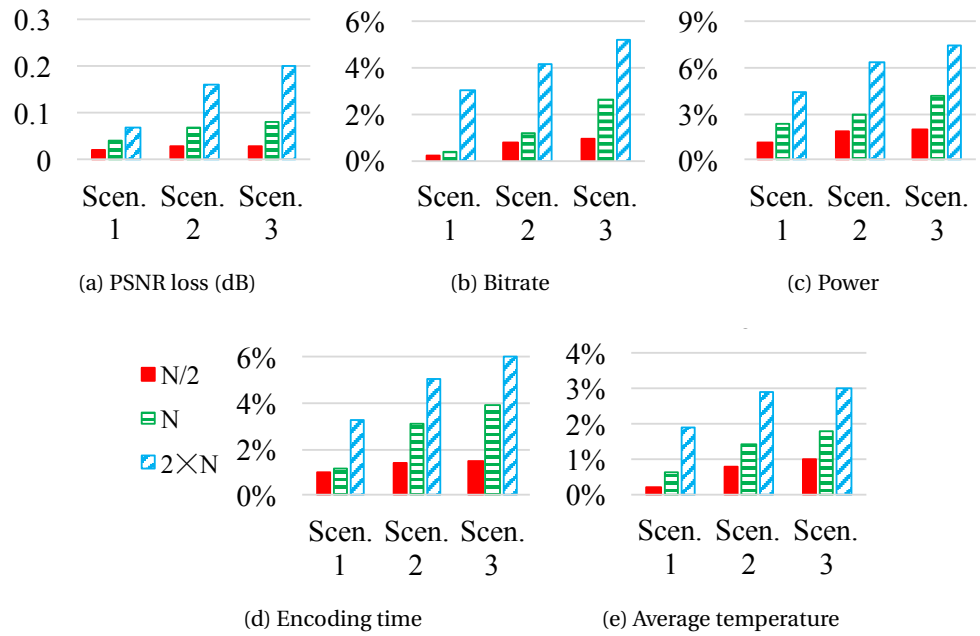


(a) (b)

Figure 4.28 – a) Euclidean distance of several solutions from the maximum defined reward $R_{max,def}$, and b) Euclidean distance of ML solution from the optimal reward, R_{opt}

scalarization function is shown by the red line. As shown in the figure, my RL-based solution provides optimal solutions for 226 frames out of 240 total frames which leads to the smallest distance from $\boldsymbol{R}_{def,max}$. Even for those remaining 14 points, in which my approach leads to a sub-optimal solution, the distance from the optimal one (the one with the smallest Euclidean distance, $\boldsymbol{R}_{opt}$) is negligible. This is shown in Figure 4.28b where for 226 frames the Euclidean distance of my solution from the optimal point is zero, while for the others this distance is in order of 10^{-3}.

The proposed RL-based approach does not have to be applied on a per-frame basis due to the fact that video contents, in spite of their constant variation, are rarely changing rapidly between two consequent frames. Hence, based on the frame rate of each running video, my proposed RL-based approach can be applied at different intervals to achieve less runtime overhead without degrading the fulfillment of my goals. Figure 4.29 shows how scaling the intervals of applying my proposed RL-based approach influences the encoding efficiency, time, power consumption, and average temperature during the exploitation phase compared with the per-frame basis (N/N). In this Figure, N is the frame rate when denoted as N Hz (such as 24 Hz, thus, the evaluated intervals are equal to 12, 24, and 48). As shown in Figure 4.29, there is slight degradation in the achieved PSNR, deviation from bitrate, power saving, average temperature, and encoding time when using larger decision interval. However, this degradation differs between the scenarios. For the first scenario, where the changes in the defined states only depend on the variation of the video content, increasing the decision intervals to $2 \times N$ leads to negligible degradation. On the contrary, in the second scenario, the defined states change as a result of changes in the number of running videos. In addition, the different videos used in the third scenario add to the dynamism of the states. Thus, more noticeable degradation is observable for these two scenarios. For a fair comparison, I did



(a) PSNR loss (dB)

(b) Bitrate

(c) Power

(d) Encoding time

(e) Average temperature

Figure 4.29 – a) PSNR loss (dB), and increase (%) in b) bitrate, c) power, d) encoding time, and e) average temperature when scaling up the decision interval compared to when using N/N interval

not apply my proposed video assignment strategy for the second and third scenarios when evaluating the impact of decision intervals.

The RL-based approach is limited by the sensor polling frequency, the availability of the application statistics, the readings from the Q-table, and updating it in *exploration* and *exploration-exploitation* phases. Updating the Q-table, however, consists of simple arithmetic operations, resulting in negligible overhead. Moreover, the proposed RL-based approach is used as a runtime power and thermal manager once it reaches the *exploitation phase*. Thus, these two phases are performed offline except when a new state is observed in the *exploitation* phase. As explained in Section 4.7.2, the RL agent has to select a random action in order to further explore this state. On average, 820 frames are required for each resolution (since I keep one Q-table per video resolution, as it has its own target bitrate and reference encoding time) after which the optimal decision is known for more than 96% of the observable states.

4.8 MARL for Runtime Management of Multiprocessor Systems

As a result of diversity in video formats, users' devices, and network bandwidth, media adaptation is required to transcode the original encoded videos to a new version in order to satisfy the resource constraints (e.g., bandwidth and resolution) and users' preference. A video transcoder consist of a decoder followed by an encoder that changes a bitstream from one format to another. Today, multiple versions of the same video are stored in different formats

and only the best one based on the user's demand is delivered. However, the fact that users daily upload more than 65 years of content to YouTube servers [221] whereas on average only the first 23 seconds of a video are watched (by Delmondo.co), implies inefficient and costly storage usage of such an approach. A promising solution is real-time video transcoder, which re-encodes the original video on the fly.

The bottleneck for achieving real-time HEVC transcoding is the encoder complexity, which is approximately 100x higher than that of the decoder [177]. Moreover, the numerous parameters available for adjusting the output quality and throughput add extra complexity. Finally, dealing with multi-user environments, where multiple different encoding requests have to be fulfilled simultaneously, poses other challenges on video providers' servers. While several works have tried to address efficient HEVC encoding through heuristics, machine learning, and model-based dynamic programming algorithms, none of them provide real-time HEVC transcoding in a multi-user environment.

In a multi-user environment, joint optimization of application- and system-level parameters should be conducted for all concurrent video streams (workloads). More importantly, the available resources to a single user is strongly affected by the resource utilization of other running streams. If there are $n_{param,app}$ and $n_{param,sys}$ different parameters of the application and system for each stream, then the design space for each stream would be $n_{param,app} \times n_{param,sys}$. Now, in a multi-user environment, the parameters for each stream should be set according to the parameters of other running streams. Thus, if there are n_{str} different streams running simultaneously, the overall design space is $(n_{param,app} \times n_{param,sys})^{n_{str}}$. Obviously, such a large design space cannot be efficiently explored by conventional heuristics and optimization approaches. Also, if the design space is extremely large, RL-based approaches employing a single agent for exploring the design space become inefficient. In fact, the RL agent has to either only partially investigate the design space, or continue exploring for long time. While the former may result in sub-optimal policies, the latter is not feasible in practice. An alternative approach is to employ multiple learning agents cooperatively to accelerate the learning process.

4.8.1 Proposed MARL Approach

In this section, I propose **MAMUT**, **MA**RL-based approach for runtime management of **MU**lti-user **T**ranscoding, which employs concurrent cooperative MARL to dynamically adapt the HEVC encoding parameters along with system-level parameters to achieve QoS-aware real-time HEVC transcoding under power budget constraints. Figure 4.30 shows an overview of the proposed approach where three agents cooperate with each other. The environment is composed of two parts: application (HEVC transcoder) and platform (i.e., server). The action set $\boldsymbol{A}$ is split to three subsets A_1, A_2, A_3 such that $\forall\, i \neq j$, $A_i \cap A_j = \emptyset$, and $\bigcup_{i=1}^{3} A_i = \boldsymbol{A}$. Agents can send messages such that each agent accesses the Q-table of the others. In addition, states and rewards resulting from one agent's action are observable to all agents.

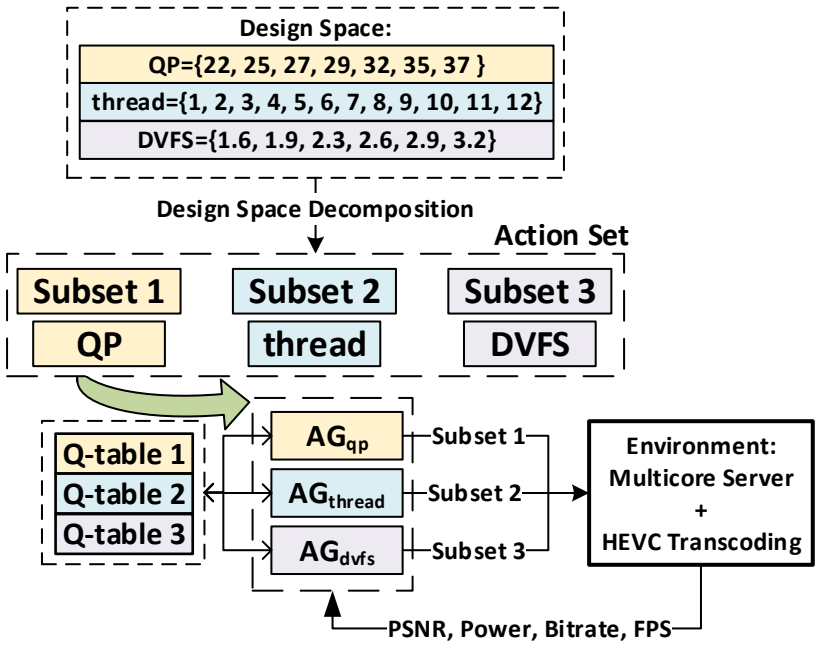


Figure 4.30 – Proposed multi-agent RL approach (MAMUT).

4.8.1.1 Agents

In MAMUT, I consider three different agents for tuning QP (AG_{qp}), deciding the number of threads used to encode a frame (AG_{thread}) through Wavefront Parallel Processing (WPP) [178], and per-core DVFS, (AG_{dvfs}).

4.8.1.2 Actions

QP. QP is one of the most important encoding parameters, as it affects FPS, PSNR, and bitrate [276]. Although QP can take a wide range of values, I use QP values of 22, 25, 27, 29, 32, 35, and 37 based on my observation on the output PSNR, bitrate, and throughput.

Number of Threads. While HEVC encoding can always benefit from multithreading to increase FPS, as shown and discussed in Section 3.2.3 and Figure 3.5, the throughput saturates above a certain number of threads. Based on this observations, I consider a limited number of threads, as described in Section 4.8.3.

DVFS. My specific platform (see Section 4.8.3) supports frequencies from 1.20 GHz to 3.2GHz. However, frequencies below 1.6 GHz can not provide real-time HEVC transcoding even if all constraints such as bandwidth and PSNR are released, and are therefore discarded.

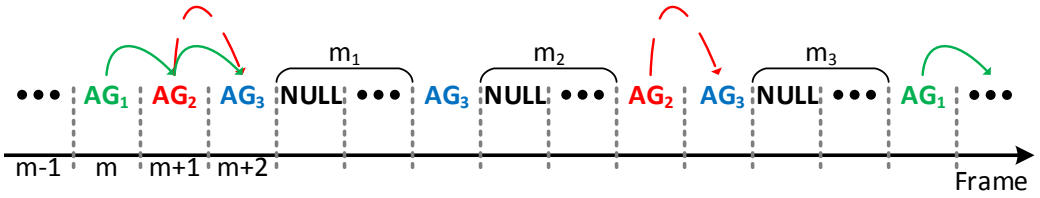


Figure 4.31 – Agent sequence. Arrows show which agents need to look at the Q-table of the next agent.

Agents Sequence. I experimentally determine how frequently each agent should act, based on overhead, impact on my target objectives, and the number of parameter values to be explored as it is desirable that all agents finish the exploration phase at the same time. For my setup, AG_{qp} acts every 24 frames. With one frame as the offset, AG_{thread} takes action every 12 frames. AG_{dvfs} takes action every 6 frames with an offset of 2 frames. Since AG_{dvfs} and AG_{thread} act after AG_{qp}, they can modify the output throughput if it is degraded (or above the required constraints) because of AG_{qp} taking an action to increase (decrease) the video quality. In addition, as AG_{dvfs} takes actions more frequently, it can take charge of content variations and tune the throughput to the desired FPS. Figure 4.31 shows the proposed sequence for the agents.

4.8.1.3 States

Agents observe the output bitrate, PSNR, throughput, and power as states. Since for 8-bit-depth videos and lossy compression PSNR should range from 30 dB to 50 dB for acceptable human vision, I divide this range in the following intervals to constitute PSNR states (S_{psnr}): $PSNR \leq 30$, ≤ 35, ≤ 40, ≤ 45, ≤ 50, and > 50 dB. Power state (S_{power}) is defined based on the power consumption constraints of the running server: $power < P_{cap}$ and $power \geq P_{cap}$. The user's available bandwidth is highly affected by different parameters such as the contract, location, etc. In order to take into account these parameters, I consider three different bitrate states (S_{br}) based on the usual bandwidth provided by a 3G network [313]: $bitrate < 3$Mb/s, 3Mb/s $\leq bitrate \leq 6$Mb/s and $bitrate > 6$Mb/s. Finally, the throughput (measured in FPS) is divided into the following states, since the target frame rate is 24: $fps < 24, < 26, < 28, < 30$ and ≥ 30.

4.8.1.4 Reward Function

In order to provide suitable feedback to each of the agents, I need to define four reward functions, one for each state:

Throughput. I define the following reward function based on the target frame rate (FPS_{target}):

$$R_{FPS} = \begin{cases} -4 & FPS < FPS_{target} \\ \frac{1}{FPS-(FPS_{target}-1)} & \text{otherwise} \end{cases} \tag{4.19}$$

This reward function provides negative values if the throughput is smaller than the target frame rate. The highest reward function is achieved if FPS exactly meets the target, however, if it is larger than FPS_{target} a smaller yet positive reward is provided. The reason is that achieving larger FPS may result in wasting resources, which ultimately means fewer users can be served. If $FPS > FPS_{target}$, spare encoded frames can be buffered. Buffered frames can be used to compensate the overall framerate if, at some points, FPS temporarily drops below the target.

PSNR. As explained in Section 4.8.1.3, a minimum PSNR of 30 dB is required. However, the goal of this work is to achieve higher video quality if there are enough resources. Hence, a higher reward is given when the agent moves to a state with larger PSNR, as follows:

$$R_{PSNR} = \begin{cases} -4 & \text{PSNR} < 30 \text{ or PSNR} > 50 \\ a \times e^{\text{PSNR}/50} - b & \text{otherwise} \end{cases} \tag{4.20}$$

where a and b are set to give a maximum reward of 1.0 when PSNR=50, and a reward of 0 when PSNR=30.

Bitrate and Power. The bitrate and power consumption are limited by the user's bandwidth and a power cap defined by the server manager (P_{cap}), respectively. Thus, I propose a reward function where a value of -4 is given if the constraint is violated, and 0.0 otherwise.

4.8.2 Learning Phases and Learning Rate Function

4.8.2.1 Exploration and Exploration-Exploitation Phases

Since each agent has its own action set, I let the agents explore only state-action pairs corresponding to their own actions. As I need to deal with a stochastic environment, applying action a_t^i by AG_i at state s_t may not always result in a particular s_{t+1}. The reason lies in the fact that 1) contents of a video can change from one frame to another, 2) other agents taking charge of a single video may apply an action that alters the next expected state to a different one, and 3) other videos existing in a multi-user platform with their corresponding contents and agents can change the state unexpectedly. Thus, once a_t^i is taken at state s_t, all state transitions to new states need to be recorded during the exploration phase. Assume that $Num(s_t \xrightarrow{a_t^i} s_{t+1})$ shows number of times that applying a_t^i at s_t resulted in s_{t+1}, and $Num(s_t, a_t^i)$ represents total number of times that a_t^i was taken at state s_t. Then, the probability by which, after taking a_t^i at

s_t, the agent observes s_{t+1} is $P(s_t \xrightarrow{a_t^i} s_{t+1}) = Num(s_t \xrightarrow{a_t^i} s_{t+1})/Num(s_t, a_t^i)$. This probability is updated throughout the learning process.

Whenever agent AG_i takes an action right before a frame starts, the next state (s_{t+1}) is observable right at the end of the frame by all agents. However, the immediate reward is only used to update the Q-value corresponding to (s_t, a_t^i). Then, the following agent takes a random action in s_{t+1} and the same procedure in observing states and updating Q-table is followed. However, when an agent is followed by no other agents (shown as NULL in Figure 4.31) the next observable state is the average of states containing the NULL action. This approach leads the agents to learn more about each others' behavior rather than about rapid video content variation, which can be regarded as noise in this case.

When the learning rate for each state-action pair drops below a threshold, α_{th1}, the agents start exploration-exploitation for that particular state. In this phase, agents do not take random actions, though after applying this particular action the Q-table is updated.

4.8.2.2 Learning Rate

Each agent must have its own learning rate for each state-action pair. The proposed learning rate function is a decreasing function of the number of state-action observations, differently from those proposed by the literature [126, 241]. The reason is that if a learning rate function similar to the literature is considered, it is likely that an agent claims the end of the exploration phase even if other agents have not taken enough different actions. This issue ultimately makes one or more agents behave sub-optimally as taking action a_t in state s_t may not move the agent to state s_{t+1} as it expects. Thus, the agent cannot maximize the reward by following the Q-table.

Alternatively, I use the following learning rate function for each agent, AG_i, which allows each agent to monitor the number and variety of actions taken by other agents:

$$\alpha^{(i)}(s_t, a_t^i) = \frac{\beta_i}{Num(s_t, a_t^i)} + \frac{\beta_i^{'}}{1 + \sum_{j \neq i} (\min_{a \in A_j}(Num(a)))} \tag{4.21}$$

Here, the first term is taken from the literature [241], while in the second one $Num(a)$ is the number of times agent A_j has taken action a. Then, $\min_{a \in A_j}(Num(a))$ gives the minimum number of times that all actions available to AG_j have been selected. Subsequently, constants β_i and $\beta_i^{'}$ need to be set such that the exploration phase for (s_t, a_t^i) cannot finish until the following two conditions are satisfied: 1) (s_t, a_t^i) is observed so many times that $\frac{\beta_i}{Num(s_t, a_t^i)}$ can drop below a threshold and, 2) other agents have tried all their actions (at least once).

Due to the different frequencies at which each agent takes an action, in addition to the different sizes of the sub-spaces each agent has to explore, the learning rate parameters can vary from

Algorithm 4.2: Exploitation phase

Input : Q^i, $P(s_t \xrightarrow{a_t^i} s_{t+1})$, $\mathbf{A}$; // $i \in \{1,\ldots,N\}$
Output: a_t^{i*} ; // current action taken by the i^{th} agent

$a_t^{i*} \leftarrow \underset{a \in A_i^*}{\arg\max}\left(\sum P(s_t \xrightarrow{a} s'_{t+1}) \times \boldsymbol{E}[Q_{Value}(AG_i.next(), s'_{t+1})]\right)$

function $\boldsymbol{E}[Q_{Value}(AG, s)]$: // list of agents, state

  if *(AG.next() == NULL)* **then**
    return $\max\limits_{a \in A_{AG}^*}\left(Q^{AG}(s,a)\right)$
  else
    $a \leftarrow \underset{a \in A_{AG}^*}{\arg\max}\left(Q^{AG}(s,a)\right)$ **return** $\left(\sum P^{AG}(s \xrightarrow{a} s') \times \boldsymbol{E}[Q_{Value}(AG.next(), s')]\right)$

one agent to the other. In this work, I experimentally set $\beta_i = 0.3$ and $\beta_i^{'} = 0.2$, $\alpha_{th1} = 0.1$ and $\alpha_{th2} = 0.05$, and $\gamma = 0.6$. γ is the discount factor, that controls the significance of the history of Q-values versus recently obtained rewards.

4.8.2.3 Exploitation Phase

The exploitation phase starts when the learning rate drops below a threshold, α_{th2}. Entering the exploitation phase, however, does not mean that exploration is finished. In fact, whenever a new state is observed by one agent, exploration phase starts for this particular state.

Although each agent learns separately and has its own Q-table, it needs to act in the exploitation phase cooperatively and, as explained in Section 4.8.1.2, in sequence. Consequently, the goal of each agent is not simply maximizing the Q-value attainable for its own Q-table, but rather, maximizing the expected Q-value after a sequence of actions taken by all agents. Imagine the sequence of agents shown as in Figure 4.31. Starting from the m^{th} frame, the first agent, AG_1, is followed by two different agents, AG_2 and AG_3. Thus, the action taken by AG_1 should consider the probable transitions from one state to the other throughout the entire chain, composed of these three agents, in order to maximize the Q-value.

Indeed, AG_1 should select an action which ultimately moves the entire system to a state in which an action taken by AG_3 is capable of providing the highest Q-value. This is equivalent to considering the expected Q-value given that a particular action is selected by AG_1. Hence, the conditional expected Q-values should be computed for all available actions in the current state s_t, in the chain of $AG_1 \rightarrow AG_2 \rightarrow AG_3$, as shown in Algorithm 4.2.

Moreover, it is possible that an agent moves to the exploitation phase earlier than the others since the number of actions that belongs to each agent is different. In such a case, the first agent in the sequence cannot rely on the behavior of the following agents. Hence, it only

follows its own Q-table regardless of the expected Q-value that is achievable at the end of the sequence. Clearly, this behavior is not optimal as the whole system is not in the exploitation phase yet.

4.8.3 Experimental Setup

4.8.3.1 Case-Study HEVC Encoder

I use the Kvazaar open source encoder as the baseline of this work, using the default *ultrafast* configuration for high-resolution (HR) videos, and the default *slow* configuration for the low-resolution (LR) videos.

4.8.3.2 Compared Approaches

In order to show the efficiency of MARL in dealing with joint optimization of application- and system-level parameters, I consider several approaches for comparison. First, I develop and implement a single-agent RL approach, where only one agent is in charge of exploring the same design space. For this purpose, I adapt the proposed framework presented in Section 4.7 such that the agent has the same action set as in the case of MARL. Moreover, for a fair comparison, I use the same reward function proposed in Section 4.8.1.4. The learning rate function remains the same as in the proposed SARL-based solution (Eq. (4.12)). Design objectives and constraints are also limited to those MAMUT is coping with, i.e., throughput (FPS), PSNR, bitrate, and power consumption. Finally, since in this part of my thesis I aim at real-time encoding, I use Kvazaar HEVC encoder as the case-study application, rather than the reference software earlier used in Section 4.7.

Second, I adapt a control-based heuristic [314] that sets the number of threads (targeting FPS), adapts QP (targeting PSNR), and applies DVFS (for power management).

Finally, I also address the joint optimization of application- and system-level parameters through a heuristic. In what follows, I describe how this heuristic works.

Proposed Heuristic for Runtime Adaptation of HEVC Encoder Parameters: I build the idea of run-time adaptation of HEVC parameters by modifying the proposed heuristic for thread allocation and DVFS of HEVC encoders presented in Section 3.5. In this context, after finishing the re-tiling of the frame according to its motion and texture, encoding parameters including search area and QP are set for each tile independently. Figure 4.32 depicts the modification applied to Figure 3.9.

While smaller QPs are necessary for high-texture tiles, larger QPs can satisfy the required video quality and compression for the low texture tiles. Therefore, I utilize QP equal to 37, 32, and 27 for the low, medium, and high texture tiles, respectively, as default values. However, I keep

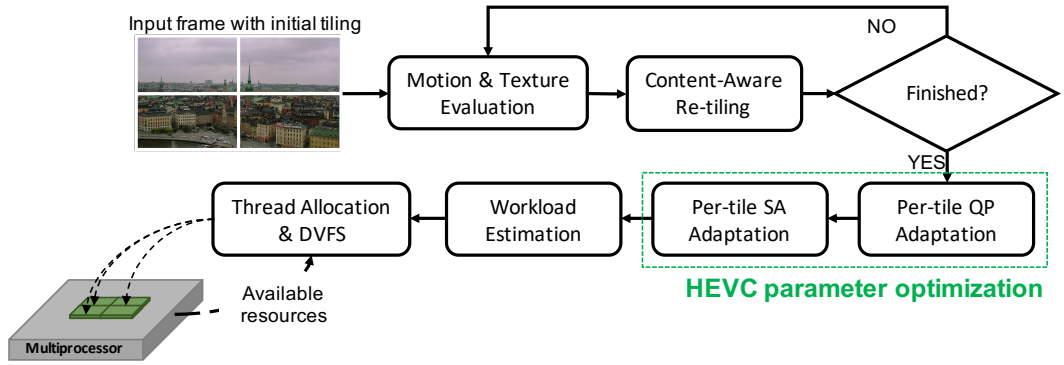


Figure 4.32 – Proposed heuristic framework for run-time adaptation of HEVC encoder parameters

Algorithm 4.3: QP adaptation

Input : $PSNR_{const}, PSNR_{margin}, \boldsymbol{PSNR_{t-\Delta t}}, \boldsymbol{M}, \boldsymbol{T}$
Output: $\boldsymbol{QP}$

```
forall Tile with T ∈ T and M ∈ M do
    if PSNR_{t-Δt} > PSNR_const + PSNR_margin then
        QP ← QP + ΔQP
    else if PSNR_{t-Δt} < PSNR_const then
        QP ← QP − ΔQP
    else
        Use default QPs w.r.t T ∈ T and M ∈ M
```

evaluating the outcome video quality (in PSNR) and compression (in bitrate) and consider two other extreme QP values (42 and 22). I observe that for very low-texture tiles $QP = 42$ can be used to further reduce encoding time and bitrate without PSNR degradation. Also, for extreme cases of high-texture tiles $QP = 22$ should be used to meet the PSNR constraint. Starting from the default QP values for different textures, I update the QP selection based on the measures of the corresponding tile of the previous frame. Algorithm 4.3 shows the pseudo code of per-tile quality-aware QP selection where $\boldsymbol{PSNR_{t-\Delta t}}$ denotes PSNR of all the tiles of the previous frame, and $PSNR_{const}$ and $PSNR_{margin}$ show the constraint of video quality and the margin by which I can guarantee that further increasing of QP value would not result in dissatisfaction of PSNR constraint. Finally, $\boldsymbol{M}$ and $\boldsymbol{T}$ are arrays containing the motion and texture of the tiles in the frame, respectively.

I consider search windows of size 128 × 128, 64 × 64, and 32 × 32. However, for low-motion tiles, a search window size of 64 × 64 is sufficient for the first frame in the GOP and it can be further decreased to 32 × 32 for the next frames to reduce the computational complexity of the motion estimation. Moreover, for each high-motion tile of the first frame the maximum allowable search window is considered, while from the second frame smaller values are used to reduce the computational complexity.

4.8.3.3 Experimental Platform

In my experiments, trying to cover the extreme cases of the design space, I consider videos with two very different resolutions: High Resolution (HR) (1080p/fullHD videos with resolution 1920 × 1080 pixels), and Low Resolution (LR) (832 × 480) videos. The specific video sequences have been extracted from the JCT-VC benchmark [181]. I perform my experiments on a 16-core (32-thread) server composed of two Intel Xeon E5-2667 v4 CPUs. The measured overhead introduced by the system is negligible (i.e, less than 0.05% of the encoding time).Per-core DVFS is available with frequencies ranging from 1.2 GHz to 3.2 GHz. However, my observations reveal that frequencies below 1.6 GHz do not allow real-time transcoding.

4.8.4 Experimental Results

I consider two different scenarios to evaluate MAMUT and compare it against SARL and the heuristic approaches described in Section 4.8.3.2. All reported results hereafter are extracted after five repetitions of the transcoding process under equal conditions, reporting the average values.

4.8.4.1 Scenario I: Serving Videos of Same Resolutions and Different Contents

In the first scenario, I assume that several different videos of the same resolution, but different contents, need to be encoded on the target multi-core server. The dynamism of the environment, therefore, includes content variation within a single video and among different videos. Figure 4.33 shows the average power consumption and QoS violation in terms of percentage of time that each video could not be encoded with the desirable throughput (24 FPS). For this figure, I consider several cases each with different number of transcoding requests. As shown in Figure 4.33 MARL outperforms all other approaches in terms of both power consumption and QoS violation. In particular, MAMUT can serve up to 5 HR and 8 LR videos with less than 20% and 15% QoS violations. Among all compared approaches, MAMUT consumes the least amount of power, implying that MARL could find more appropriate application- and system-level parameters in different situations. Overall, MARL is consistently able to reduce power consumption between 10% to 24% when compared with the SoA and up to 7% compared with SARL implementation. A maximum improvement of 8x, and 5x in terms of FPS violations is achieved, when compared with the SoA and the SARL. SARL, on the other hand, outperforms the proposed heuristic method and SoA in most cases unless when 4 and 5 HR videos are to be encoded. As discussed earlier, when the design space becomes extremely large, SARL cannot well explore all corner cases, even though it still achieves desirable results in more common cases. The proposed heuristic method behaves slightly better than the SoA in power savings and QoS violations due to its ability to analyze the input data. Especially, its QoS violation is close to SARL when serving only one video. However, when the number of concurrent videos increases the proposed heuristic which can take care of encoding of individual videos fails to find proper parameters.

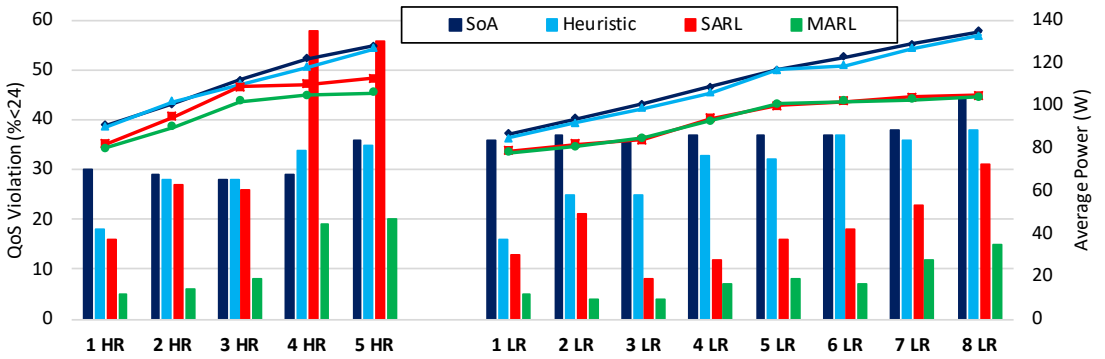


Figure 4.33 – QoS violation (in terms of percentage of frames under QoS threshold) and power consumption for the SoA, heuristic, SARL and MARL (MAMUT) encoding different combinations of HR and LR videos.

Table 4.7 – Number of threads and frequency used in average

	MARL		SARL		SOA		HEURISTIC	
	N_{th}	Freq.	N_{th}	Freq.	N_{th}	Freq.	N_{th}	Freq.
HR	10.1	2.8	9.2	2.9	5.9	3.2	7.2	3.1
LR	3.7	2.8	3.2	2.7	2.6	3.2	6.9	3.0

Several important notes can be derived from the experimental results. First, when comparing MAMUT with the Heuristic approach, the run-time behavior is inherently different. The Heuristic approach tries to achieve QoS requirements using maximum frequency and a low number of threads, whereas MAMUT encodes each frame using a higher number of threads, but lower frequency, as shown in Table 4.7. This behavior greatly improves power consumption.

SARL also applies the same policy as of MAMUT. On the contrary, the proposed content-aware heuristic use high number of threads and larger frequency for both HR and LR videos. An important note here is that the heuristic approach does not take into account the resolution for setting the frequency and number of threads. Second, MAMUT is able to better use the available resources and adapt to different loads, as in situations in which the load is low it achieves much larger QoS with lower power, and under high load, it obtains slightly better QoS and saves power. In approach of SoA, QoS is almost constant and MAMUT manages to adapt the computation to the available resources (constantly achieving better results). Third, although both RL-systems are able to learn a similar policy, SARL is not able to provide the same QoS measurements. Additionally, the amount of time it takes SARL to finish exploration phase is 15 times larger than that of MARL. Finally, the PSNR achieved for all the proposals is close to 34 dB in the case of HR videos, ranging from 36 dB when one LR video is encoded by a RL-system, to 41 dB when it is encoded by the heuristic approach. Concerning bitrate and power, all the implementations meet the constraints.

Figure 4.34 shows the traces of actions selected by MAMUT and output FPS for a randomly selected video when encoding 5 LR videos. As shown in the figure, the main action in charge

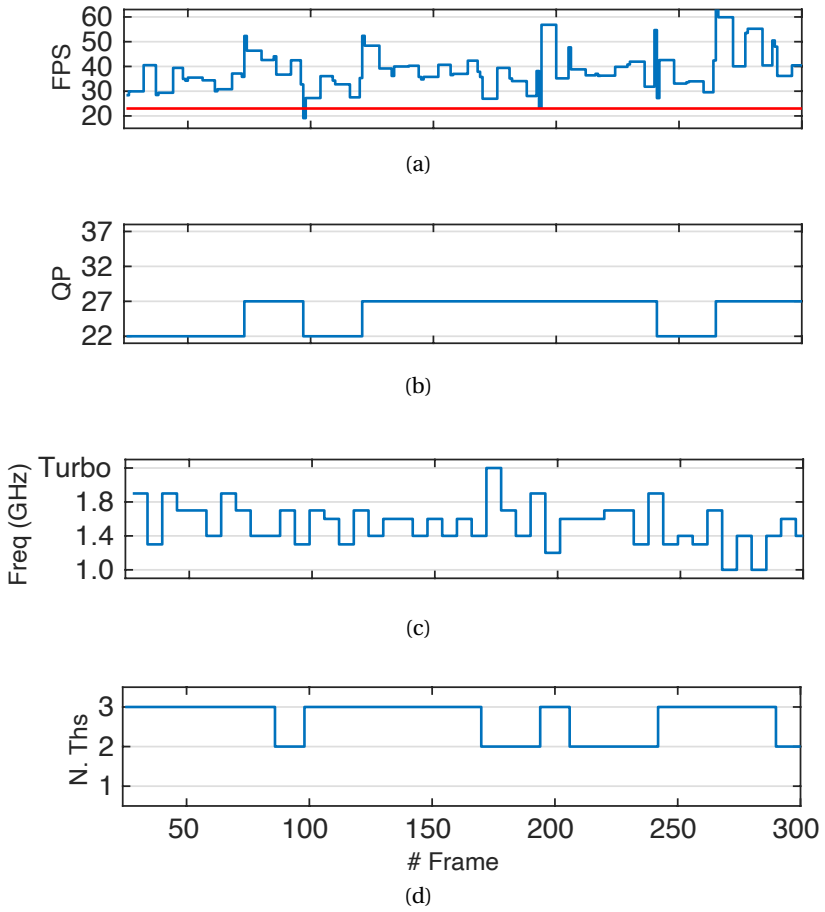


Figure 4.34 – Traces of actions selected by MAMUT and output FPS for a randomly selected video when encoding 5 LR videos.

of adapting the requirements of a single video according to its content variation in addition to other videos requirements is the operating frequency. This action has negligible overhead and does not need to be applied once a frame starts or is finished. As indicated by Figure 4.34a, during these 300 frames of the selected video a very small portion is encoded with below 24 FPS. More importantly, the amount of QoS violation in terms of the distance of output FPS to the required one is not large. This is, in fact, the case for both MARL and SARL. Although, QoS violation occurs in both approaches, the amount of violation is not large. This small violation can be well compensated by having frames constantly encoded above the required FPS. This approach is very common and is realized by means of buffers.

4.8.4.2 Scenario II: Serving Videos of Different Resolutions

In the second scenario, I assume that a set of transcoding requests is simultaneously received from different users with different resolution requirements. In contrast to the first scenario, here I consider sequences of random videos being simultaneously transcoded, simulating a real scenario. To restrict the extent of the experiment, I consider that each initial video is

Table 4.8 – Scenario II, average results. Each row reports metric for a sequence of a specific combination of videos.

	SoA			SARL			MAMUT			Heuristic		
	W	N_{th}	FPS	W	N_{th}	FPS	W	N_{th}	FPS	W	N_{th}	FPS
1Hr +1Lr	96.0	4.2	25.4	92.4	6.4	28.1	88.4	7.9	31.1	95.4	6.8	25.6
1Hr +2Lr	106.3	4.1	24.6	96.7	5.6	26.4	93.4	6.5	31.3	106.0	6.9	25.2
2Hr +1Lr	109.7	4.7	25.2	102.3	7.6	26.5	97.4	9.7	30.4	109.6	6.9	25.3
2Hr +2Lr	114.5	4.5	25.0	105.4	6.6	25.8	100.3	7.6	29.5	112.2	7.1	24.9
2Hr +3Lr	123.3	4.2	24.9	107.4	5.9	25.0	101.9	5.9	26.2	120.4	7.8	24.9
2Hr +4Lr	124.5	4.4	25.1	108.5	5.8	24.7	100.9	6.3	27.7	121.8	7.9	23.5
3Hr +1Lr	122.5	5.3	24.4	113.7	7.9	23.3	104.3	8.7	26.2	121.5	7.1	25.0
3Hr +2Lr	129.9	5.4	24.5	110.9	7.0	21.7	105.2	7.0	25.3	128.1	7.6	23.9
3Hr +3Lr	134.6	6.1	23.4	111.8	6.2	20.8	106.9	6.2	25.1	134.7	7.4	23.7

followed by a sequence of four different videos of the same resolution, randomly selected. With this scenario, I demonstrate the capability of my approach to satisfy QoS requirements of different users with different demands during time. Also, given the random nature in video contents, I explore the capability of the approach in dealing with different video contents.

Table 4.8 shows the average values for the main metrics for a specific combination of video types in scenario II. Qualitatively, the behavior of all implementations is similar to that in the previous scenario, and all are able to satisfy QoS requirements if the workload is not close to the resource saturation point, achieving an average PSNR ≈ 36dB in all approaches. When the server is fully utilized (e.g., when transcoding three Hr videos simultaneously), MAMUT still achieves the best results in terms of QoS, whereas SARL cannot meet the QoS requirements. The reason lies in that in the present extremely large design space, SARL cannot adapt to all situations. MAMUT consumes 8% to 20%, and 4% and 7% less power when compared to the SoA and SARL approaches, respectively. Finally, QoS violations are reduced by up to 8x and 4x compared with the SoA and SARL approaches, respectively.

4.9 Design Space Search for CNN Optimization

CNN design and optimization is one of the problems that can be efficiently addressed through RL, due to its very large design space. In this section, I address automation of hyperparameter optimization of an existing architecture rather than Neural Architecture Search [292] from scratch. Particularly, I develop a MARL-based framework that statically optimizes hyperparameters of CNN with respect to any design objective and constraint. My idea lies in the fact that many existing architectures, in terms of connections between different layers, have been already shown to achieve satisfactory results for a wide-range of applications, such as image classification, semantic segmentation, and object tracking, under particular open-sourced datasets. These models, however, mostly have been designed for particular competitions

such as Large Scale Visual Recognition Challenge (ILSVRC). Thus, they may not suit other datasets, even for the same tasks, and they are not usually optimized with respect to other design objectives or constraints, such as inference time, energy consumption, and model size. However, hyperparameter optimization of the existing CNN architectures can adapt these networks for particular datasets and design objectives and constraints. Building new architectures from scratch, on the contrary, is more expensive and time-consuming, even though there are several works addressing automation of this procedure.

4.9.1 Proposed MARL Framework for Hyperparameter Optimization of CNNs

Hyperparameter optimization of CNNs is intrinsically different from runtime management of multiprocessor systems since in the former the designer needs to deal with static optimization. Indeed, the problem of hyperparameter optimization of CNNs is about searching through an extremely large design space efficiently to find appropriate values such that they provide a desirable network in terms of design objectives and constraints, such as inference time and accuracy. Once these parameters are found, there is no need to dynamically tune them at runtime (i.e., while using the designed CNN). On the contrary, runtime management of multiprocessor systems requires continuous supervision of the problem as it may change dynamically according to workload variations.

Nonetheless, these two problems have one feature in common, which is their extremely large design space. Also, although the problem in runtime management of multiprocessor systems may change dynamically, RL is in charge of exploring a design space to find the best solution for every different situation. Similarly, in CNN hyperparameter optimization, RL can be used to explore a design space. The difference is that in the latter the RL agent performs actions episodically, while runtime management is not an episodic problem. Therefore, the same approaches proposed in previous sections using SARL and MARL cannot be directly applied for hyperparameter optimization of CNNs. Nevertheless, the results of using MARL solution in dealing with very large design spaces, while speeding up the exploration phase by 15x compared to the SARL approach, strongly motivates leveraging multiple agents for hyperparameter optimization of CNNs.

Figure 4.35 shows an overview of my proposed MARL-based hyperparameter optimization of an n-layer CNN with arbitrary skip connections. In particular, I adapt QL and define learning agents per layer to split the design space to smaller independent sub-spaces; thus each agent can fine-tune the hyperparameters of the assigned layer with respect to a global reward. In the proposed multi-agent QL-based approach, agents are able to communicate through my novel definition of state-action pairs, Q-tables, and Q-table update rule.

In my problem definition, the design space is too large for a single agent to explore. Moreover, having multiple agents explore the same whole design space and then sharing their learned policies, as in the case of homogeneous agents, is not of much help. This lies in the fact that, due to the excessively large design space, each agent can only partially explore the design

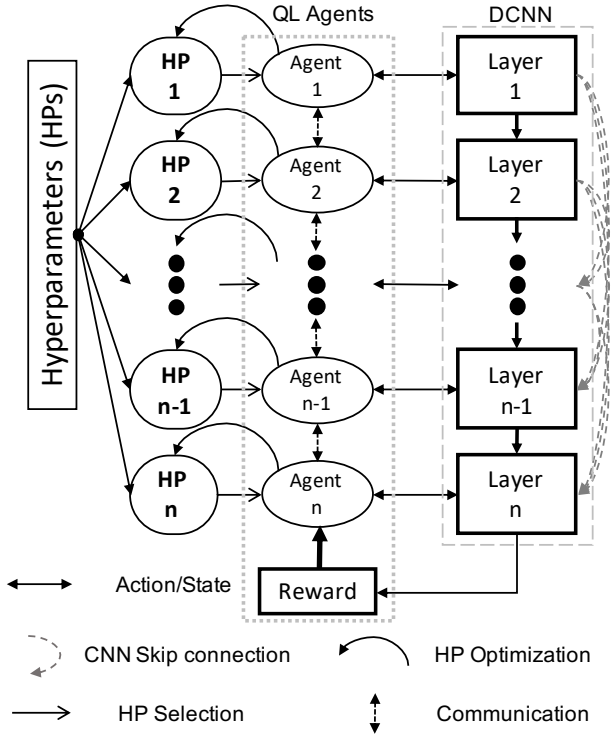


Figure 4.35 – Overview of proposed MARL-based hyperparameter optimization

space, and sharing incomplete experience and defective learned policies may hardly result in an optimal or near-optimal policy. Consequently, I address the problem of DCNN design and hyperparameter optimization through splitting the design space into smaller sub-spaces among heterogeneous agents, letting each agent fully explore its own space and episodically share its experience to other agents.

In particular, I propose to use multiple QL agents to design a DCNN, layer by layer, through a cooperative teamwork. During the learning process, each agent is assigned to design a single layer. An agent's action, however, affects the following agents' behavior. Therefore, agents target to find the best action at a given state to lead the team gain a higher reward. Since the current state observed by each agent is a direct consequence of the previous agent's action, each agent has to optimize its behavior such that the next agent is situated in a desirable state. Such a behavior, then, should propagate throughout the whole CNN and all layers, where, finally, the last agent's action results in a reward signal.

Think of a soccer game, where each player can be considered as an agent similar to my problem statement. The game starts from the goalkeeper, i.e., the first agent, where the objective is to pass the ball from one player to another in a predefined order (agent sequence determined by the CNN architecture) until the last player, i.e., the last agent, finally shoots it towards the target to score. The task of each player, therefore, is to learn how to pass/shoot the ball, perhaps with which strength and direction (different action types and values). As a consequence, the quality

of a player's shot determines the state (a nice pass or a bad one!) in which the next player receives the ball to continue the game. Finally, the last player's shot gives the whole team a reward (based on whether they score or not, or how fast they could score, etc.). This is similar to using multiple agents to sequentially design a CNN, where the quality (accuracy, time, etc.) of the teamwork is observable at the end of one episode. In the soccer game described above, one episode starts from the goalkeeper and ends with the last player. Similar to the agents designing a CNN, players need to evaluate different actions in various given states to finally learn how to adjust their behavior so that the next player can benefit the most. Ultimately, the last player can successfully score if other previous players could accomplish their given tasks.

Figure 4.36 shows an abstract view of an arbitrary CNN composed of 5 layers to be designed. The design of each layer L_i, $i \in \{1,2,3,4,5\}$, is managed by an agent, AG_i, with a particular available action set, $\boldsymbol{A_i} = \{a_{i,j} | j = 1,2,3,...\}$, after splitting the design space into smaller sub-spaces, as the shown in the figure. After observing its current state, each agent is able to apply an action from this action sub-set. A unique reward signal is also received by each agent, based on which further updates on the Q-tables are possible. In particular, each two successive agents share one Q-table, as shown in Figure 4.36. In conventional Q-learning, a Q-table is simply a table where rows and columns represent, respectively, states and actions of a single agent. In my defined multi-agent environment, however, each agent has its own action set, whereas its current state is determined by the actions taken by the previous agent in the sequence as will be explained in Section 4.9.1.3. Thus, the state set, $\boldsymbol{S_i}$ observable by the i^{th} agent, is the same as the action set available to the $(i-1)^{th}$ agent, $\boldsymbol{A_{i-1}}$. In order to model this scenario in the proposed multi-agent QL-based solution, I propose to use Q-tables shared between each two consecutive agents, as shown in Figure 4.36. In what follows, I detail the different elements of my multi-agent environment, namely, agents, states, actions, Q-tables, and reward function for hyperparameter optimization of DCNNs.

4.9.1.1 Agents

Given a CNN architecture, there are as many agents as the number of layers whose parameters need to be decided. In a particular layer, there could be more than one parameter value to be selected, however, I let a single agent be in charge of tuning these parameters. Moreover, agents do their part one after each other, i.e., sequentially, in the same order as of the CNN architecture, from the input to the output.

4.9.1.2 Actions

Each agent is responsible for hyperparameter tuning of a particular layer. Therefore, the action space of an agent is defined by the number of different parameters and their corresponding values in the layer. Each layer may contain more than one hyperparameter. In addition, each of these hyperparameters can take different values. Hence, an agent's action set is composed of tuples of all possible values of all different available hyperparameters. For

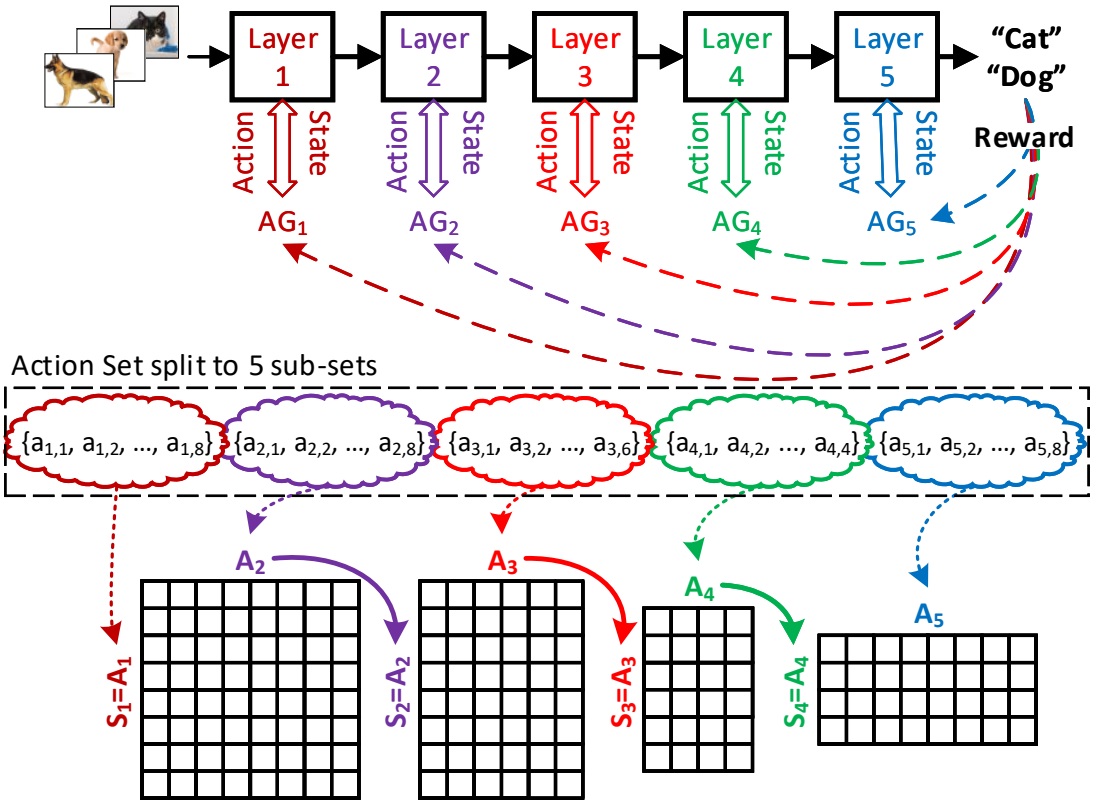


Figure 4.36 – Schema of proposed MARL-based approach on a 5-layer CNN

instance, in a convolution layer, parameters such as number of kernels ($\{n_{k_1}, \cdots, n_{k_N}\}$), kernel size ($\{s_{k_1}, \cdots, s_{k_M}\}$) , and stride length ($\{l_{s_1}, \cdots, l_{s_L}\}$) can be considered as hyperparameters and their corresponding values. Thus, for the case of this convolution layer, the agent's action set is defined as:

$$\begin{aligned} \boldsymbol{A_{conv}} = \{&(n_{k_1}, s_{k_1}, l_{s_1}), (n_{k_2}, s_{k_1}, l_{s_1}), \cdots, (n_{k_N}, s_{k_1}, l_{s_1}), \\ &(n_{k_1}, s_{k_2}, l_{s_1}), (n_{k_1}, s_{k_3}, l_{s_1}), \cdots, (n_{k_1}, s_{k_M}, l_{s_1}), \\ &(n_{k_1}, s_{k_1}, l_{s_2}), (n_{k_1}, s_{k_1}, l_{s_3}), \cdots, (n_{k_1}, s_{k_1}, l_{s_L})\}. \end{aligned}$$

Consequently, when the convolution agent takes an action, it is, in fact, choosing a tuple from $\boldsymbol{A_{conv}}$.

4.9.1.3 States

By splitting the design space to multiple independent sub-spaces, states observed by each particular agent is different from others during the whole learning process. In particular, I define the state sub-space, to be experienced by an agent, as the actions taken by the previous agent in the sequence of agents explained in Section 4.9.1.1. Therefore, whenever an agent takes an action, it modifies the state of the next agent in the sequence. The first agent in the

sequence, however, is always considered to remain in an initial state.

4.9.1.4 Multi-Agent Q-Tables

Each Q-table, Qt_i, is an array of size $N_{A_i} \times N_{A_{i+1}}$ where N_{A_i} and $N_{A_{i+1}}$ are the number of actions available to the i^{th} and $(i+1)^{th}$ agent. Each two consecutive agents in the agent sequence share one Q-table, hence, there are $N_{AG}-1$ Q-tables in total, where N_{AG} denotes the number of agents (layers). I initialize each Q-table with zeros. In the proposed MARL approach, although there are $N_{AG}-1$ Q-tables, each Q-table occupies only a few kilo bytes of the memory. For instance, only around 255KB is required to store all the Q-tables of U-Net through the proposed approach.

4.9.1.5 Reward Function

The reward function in my framework can contain a large variety of signals readable, observable, or measurable at the end of each episode, such as training loss/accuracy, validation loss/accuracy, training/validation time, GPU/CPU/memory utilization, model size, etc. In this work, I use validation accuracy and model size as the main parameters of the reward function. Moreover, I use training loss and training time as monitoring parameters in early termination of the current episode (training with current selected hyperparameters). In particular, I monitor training time for each batch (t_{batch}), and activate early termination of the current episode if the current training time violates a predefined threshold time (t_{th}) for $N_{th,t}$ consecutive batches. Moreover, I monitor the training loss at the end of each epoch. If the training loss (l_{epoch}) violates a predefined loss threshold (l_{th}) for $N_{th,l}$ consecutive epochs, training with the current hyperparameter set is terminated. Figure 4.37 shows how these two mechanisms work to provide early termination, where $N_{violation,t}$ and $N_{violation,l}$ are two counters used for counting the number of consecutive violation of training batch time and epoch loss, respectively.

Finally, whenever the early termination is activated through batch training time, an immediate penalty is given to the agents, whereas, if the early termination is due to the epoch loss, I propose to postpone this penalty. The reason lies in the fact that a CNN may perform poorly in the beginning due to the initial values of the weights. By restarting the training of the CNN with the same hyperparameters, I target to decrease the impact of these initial values. However, I tolerate such behavior only once, i.e., if the early termination is activated for the second time, agents are given a penalty (negative reward). The complete definition of the reward function is as follows:

$$\mathscr{R} = \begin{cases} -1 & Termination = True \\ c_A A_{val} - c_S S_{model} & otherwise \end{cases} \tag{4.22}$$

where A_{val} is the validation accuracy and S_{model} is the model size in MB, and c_A and c_S are

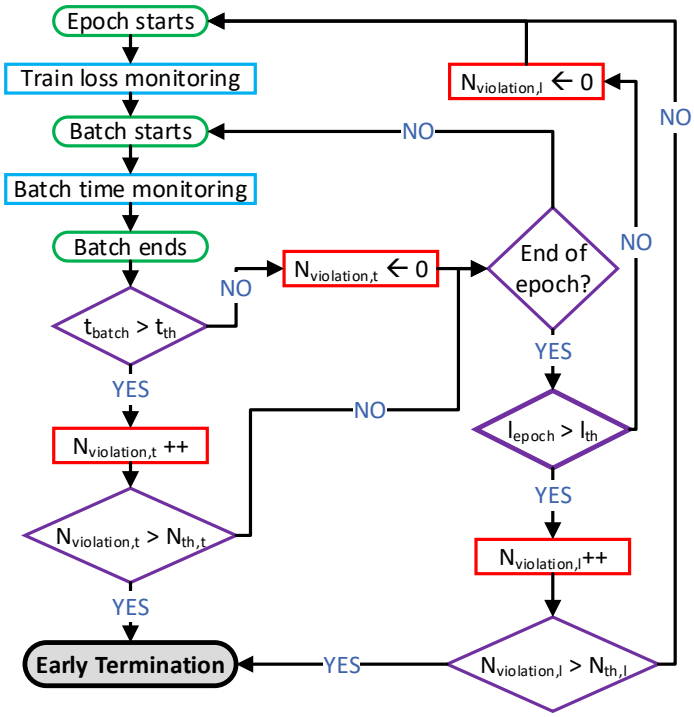


Figure 4.37 – Early termination mechanism

coefficients such that $c_A + c_S = 1$. These two coefficients can be used to prioritize one objective over the other to the agents.

4.9.2 Learning Process

The proposed learning process consists of two main phases, namely *exploration* and *exploration-exploitation*. In each phase, the strategy to take actions is different. Moreover, since my multi-agent QL-based solution is different from conventional QL algorithm with a single agent, the QL update rules cannot be applied. Thus, I propose a new rule to update the Q-tables. The complexity of the conventional QL update rule algorithm in $O(log(n_{q,sarl}))$, where $n_{q,sarl}$ is the size of the Q-table. For the proposed approach the complexity becomes $O(Nlog(n_{q,marl})$, where $n_{q,marl}$ is the size of the largest Q-table, and N is the number of Q-tables. However, through my MARL-based solution, $n_{q,marl}$ is significantly smaller than $n_{q,sarl}$ (e.g., more than 10^{45} times smaller for GoogLeNet).

4.9.2.1 Exploration Phase

In the exploration phase, agents are only allowed to take random actions. This is similar to an $\epsilon - greedy$ [235] policy with ϵ always kept at 1. However, to help each agent explore more completely different states, I propose Algorithm 4.4. As shown in my algorithm, each agent takes diverse actions without repetition until its Q-table does not contain any cells initialized

Algorithm 4.4: Random action in exploration phase

```
   Input  : Qt_i, A_i = {a_ij j ∈ {1,...,N_Ai}}
   Output : a*_i ;                                   // action of the i^th agent
1  forall i do
2      if i = 1 then
3          find (a*_i, a*_{i+1}) st. Qt_i[a*_i, a*_{i+1}] = 0
4      else
5          find a*_{i+1} st. Qt_i[a*_i, a*_{i+1}] = 0
```

to zero. The first two agents take their actions at a single step without any condition on the next agent (i.e., AG_3). On the contrary, the next agents take actions whose values are still equal to the initialized value (zero), but with respect to the action already selected by the previous agent (Line 5). Moreover, if there is no action found such that $\boldsymbol{Qt_i = 0}$ (Lines 3 and 5), AG_i does not follow Algorithm 4.4 and takes a completely random action.

Then, based on Algorithm 4.4, the number of episodes to eliminate all zeros (initial values) from all the Q-tables is determined by the largest Q-table. In other words, I require at least $\max(N_{A_i} \times N_{A_{i+1}}), i \in \{1,...,N_{AG}\}$ episodes during the exploration phase.

Each CNN created by the agents is trained for a very limited number of epochs. Although the validation and/or train loss and accuracy at the end of these epochs would be far from the one to which the designed CNN can ultimately converge, it gives a useful insight into the behavior of the CNN with respect to training and validation time, and model size, as well as whether the CNN hyperparameters sound or could be discarded later on. This number of epochs for each CNN and input data, however, may be different. In order to automatically set this number, I first set the maximum number of epochs in each episode equal to 10 similar to [239]. At the end of the episode, I monitor the loss and the epoch number where the predefined loss threshold discussed in Section 4.9.1.5 is already satisfied. If this satisfaction occurs at an epoch smaller than 10, I set the number of epochs in each episode to this smaller value.

For the first episodes, the agents may benefit from more number of epochs to reach the loss threshold. In order to provide a fair comparison among different episodes, I use an accuracy-like score value instead of the absolute accuracy in the reward function of Eq. (4.22), where A_{val} becomes $\frac{A_{val}}{N_{epoch}}$, with N_{epoch} indicating the minimum number of epochs through which the loss threshold constrained is satisfied during one episode. My results (Section 4.9.4) indicate that the number of epochs required to train the CNN in an episode finally converges to a minimum value. Hence, my experimental validations indicate that this minimum value should not be lower than three, and I do not look for smaller values once the minimum number of epochs is found to be three. This number, ranging from 3 to 10, is yet considerably smaller than the total number of epochs to fully train a state-of-the-art CNN (i.e., 100 to more than 200 epochs, depending on the CNN).

4.9.2.2 Q-table Updates

After all agents apply an action to their own layers, the reward is available. This reward is used to update the agents' Q-tables. I propose to follow Algorithm 4.5 as the Q-table update rules in my specific problem. The main idea of this new Q-table update rule is that if there is any cells remained with the initial zero value within the next Q-table ($\boldsymbol{Qt_{i+1}}$), then, the current Q-table ($\boldsymbol{Qt_i}$) is updated only according to its own current Q-values and the obtained reward (Lines 5-6). Otherwise, $\boldsymbol{Qt_i}$ is updated by the maximum expected Q-value of $\boldsymbol{Qt_{i+1}}$ and the obtained reward (Line 8). Finally, as shown in Algorithm 4.5, the last Q-table is updated slightly differently from the others, since there is no Q-table afterwards (Lines 2-3).

One of the key elements of the proposed Q-table update rule is the learning rate value, α. I treat this parameter in two different ways in exploration and exploration-exploitation phases. In this context, I initialize α to 0.95 and do not change it during the exploration phase. However, once the exploration-exploitation phase starts, I reduce the learning rate at every episode, as follows:

$$\alpha_{new} = \alpha_{old} \times 0.999^{n_{episode}} \tag{4.23}$$

where $n_{episode}$ is the number of episodes already passed in the exploration-exploitation phase.

4.9.2.3 Exploration-Exploitation Phase

In this phase, I use a decay function to reduce ϵ and provide a tradeoff between exploration and exploitation similar to Eq. (4.23). I clarify that, in my work, exploitation does not mean to apply an already taken set of hyperparameters, but to have each agent look into its Q-table shared with the next agent to pick an action that maximizes the expected reward. In this phase, if agents are to exploit their previous experience, Algorithm 4.6 is used, otherwise a random action is taken. In the action strategy shown in Algorithm 4.6, the first two agents take their actions simultaneously to maximize their shared Q-table (Line 3). The next agents, however, always select an action which maximizes a particular row of its Q-table, determined by the previous agent in the sequence. As explained in Section 4.9.2.2, each Q-table cell is updated with respect to the next Q-table. Therefore, when in the exploitation phase an agent looks into its own Q-table and selects the best action accordingly, it is, indeed, selecting the one that is expected to benefit the next agent the most. All in all, this procedure most probably provides a completely new set of hyperparameters in the beginning. Due to these new findings of the agents, I keep updating the Q-tables by Algorithm 4.5. This way, agents are able to further revise their behavior. Nonetheless, if a hyperparameter set exactly matches a previously experienced one, Q-tables are not updated. At the end, an optimal set of hyperparameters may be selected for several episodes by the agents. This point is where I achieve the convergence of the MARL-based approach.

Algorithm 4.5: Q-table update rule

Input : $\boldsymbol{Qt_i}, a_i^*$; // $i \in \{1, ..., N_{AG}-1\}$
Output: $\boldsymbol{Qt_i}$; // Updated Q-table

1 **forall** i **do**
2 **if** $i = N_{AG} - 1$ **then**
3 $\boldsymbol{Qt_i}[a_i^*, a_{i+1}^*] \leftarrow \boldsymbol{Qt_i}[a_i^*, a_{i+1}^*] + \alpha\mathscr{R}$
4 **else**
5 **if** $\exists a_{i+2} \ st. \ \boldsymbol{Qt_{i+1}}[a_{i+1}^*, a_{i+2}] = 0$ **then**
6 $\boldsymbol{Qt_i}[a_i^*, a_{i+1}^*] \leftarrow \boldsymbol{Qt_i}[a_i^*, a_{i+1}^*] + \alpha\mathscr{R}$
7 **else**
8 $\boldsymbol{Qt_i}[a_i^*, a_{i+1}^*] \leftarrow (1-\alpha)\boldsymbol{Qt_i}[a_i^*, a_{i+1}^*] + \alpha(\mathscr{R} + \gamma \max_{a_{i+2}}(\boldsymbol{Qt_{i+1}}[a_{i+1}^*, a_{i+2}]))$

Algorithm 4.6: Action selection in exploitation phase

Input : $\boldsymbol{Qt_i}, \boldsymbol{A_i} = \{a_{i,j}, j \in \{1, ..., N_{A_i}\}\}$
Output: a_i^* ; // action of the i^{th} agent

1 **forall** i **do**
2 **if** $i = 1$ **then**
3 $(a_i^*, a_{i+1}^*) \leftarrow \arg\max_{a_i, a_{i+1}} \boldsymbol{Qt_i}[a_i, a_{i+1}]$
4 **else**
5 $a_{i+1}^* \leftarrow \arg\max_{a_{i+1}} \boldsymbol{Qt_i}[a_i^*, a_{i+1}]$

4.9.2.4 Support for Skip Connections, Residual, Inception, and other Unconventional Modules

Algorithms 4.4, 4.5, and 4.6, as explained in this section, work for all classical CNNs such as AlexNet and VGG. However, they require modifications when dealing with more modern CNNs where skip connections and other modules such Inception [315] and Residual [225] are added to the network. Figure 4.38 shows two examples of different unconventional connections between layers in modern CNNs. In such modules, the layer that feeds multiple layers, or the one that is fed by multiple layers, shares one separate Q-table with the layer to which it is connected.

In Figure 4.38a, layer m needs to take action and update its shared Q-table with layer $m-1$, with respect to layers $m+1$ to n. First, in Algorithm 4.4 only Line 3 is affected if $m = 1$. This line changes to the following:

$$\textbf{find} \ (a_j^*, a_{j+1}^*) \ \ st. \ \boldsymbol{Qt_j}[a_j^*, a_{j+1}^*] = 0, j \in \{1, ..., n\}$$

Second, lines 5-8 of Algorithm 4.5 change to the following such that the Q-table shared between

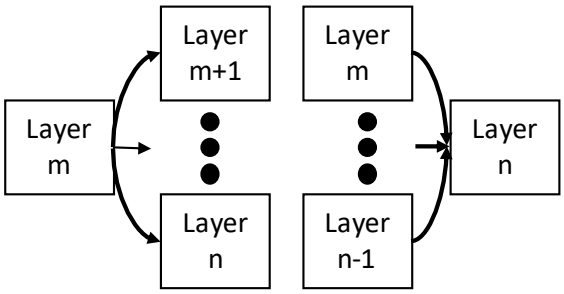


Figure 4.38 – Two types of unconventional connections in modern CNNs: a) one layer feeds multiple layers, b) one layer is fed by multiple layers

layer m and $m-1$, i.e., $\boldsymbol{Qt_{m-1}}$ is updated:

$$\begin{aligned}
&\textbf{if} \quad \exists a_{m+j} \;\; st. \;\; \boldsymbol{Qt_{m-1+j}}[a^*_m, a_{m+j}] = 0, j \in \{1,...,n\} \\
&\qquad \boldsymbol{Qt_{m-1}}[a^*_{m-1}, a^*_m] \leftarrow \boldsymbol{Qt_{m-1}}[a^*_{m-1}, a^*_m] + \alpha\mathscr{R} \\
&\textbf{else} \\
&\qquad \boldsymbol{Qt_{m-1}}[a^*_{m-1}, a^*_m] \leftarrow (1-\alpha)\boldsymbol{Qt_{m-1}}[a^*_{m-1}, a^*_m] + \\
&\qquad\qquad \alpha(\mathscr{R} + \gamma \max_{a_{m+j}}(\boldsymbol{Qt_{m-1+j}}[a^*_m, a_{m+j}]))
\end{aligned}$$

Finally, only Line 3 in Algorithm 4.6, where $m = 1$, changes to the following:

$$(a^*_m) \leftarrow \arg\max_{a_m} \boldsymbol{Qt_{m+j}}[a_m, a_{m+j}], j \in \{1,..,n\}$$

In Figure 4.38b, layers m to $n-1$ need to take random actions in the exploration phase, such that their corresponding Q-tables shared with layer n do not have any zeros (Algorithm 4.4). Then, each $\boldsymbol{Qt_i}, i \in \{1,...,n-1\}$ is updated according to the maximum Q-value of the Q-table shared between that layer and layer n (Algorithm 4.5). Finally, if I follow Algorithm 4.6 to apply action in the exploitation phase, each layer m to $n-1$ may point to a different action in layer n. However, I modify this algorithm such that agent n selects an action that maximizes more number of Q-tables shared between layer n and layers m to $n-1$. If such an action does not exist, agent n selects an action that obtains the highest average Q-value among all Q-tables shared.

4.9.3 Experimental Setup, Test-Case DCNNs, and Datasets

In this work, I use Keras with Tensorflow backend to implement all test-case CNNs and I perform all experiments on an NVIDIA V100 GPU. In order to show my proposed MARL-based solution is capable of optimizing hyperparameters of CNNs with different architectures, I

Table 4.9 – Model settings and datasets

	VGG-16	**GoogLeNet**	**U-Net**
Batch size	128	32	20
Epoch	50	50	50
Loss	Categorical crossentropy	Categorical crossentropy	Dice
Optimizer	SGD	SGD	Adam
Dataset	CIFAR100	CIFAR100	BraTS'18
	ImageNet	ImageNet	ISIC'18

Table 4.10 – Layers and hyperparameters

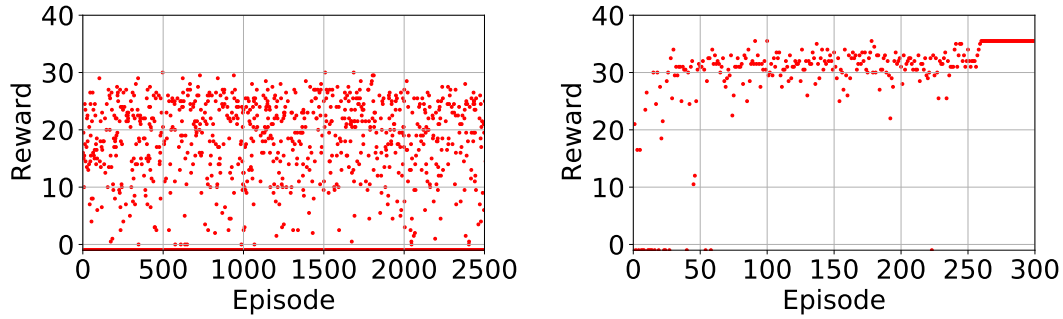
Layer Type	**Convolution**			**Pooling**	**Dense**
Hyperparameter	Number of Kernels	Kernel Size	Stride	Size	Output Dimension
VGG-16	{16, 32, 64, 128, 256}	{3,5}	{1,2}	{2,3}	{128,256,512, 1024, 2048,4096}
GoogLeNet	{32, 48, 64, 80, 96, 112, 128, 160, 208, 384}	{1,3,5}	{1,2}	{2,3,5,7}	{128,256,512, 1024, 2048,4096}
U-Net	{16, 32, 64, 128, 256, 512, 1024}	{3,5,7}	{1,2}	{2,3}	-

apply it to U-Net [316], VGG-16 [224], and GoogLeNet [315]. Table 4.9 shows the datasets and the settings used to train each CNN. The VGG-16 and GoogLeNet architectures were originally used for ImageNet datasets. In order to make them compatible to CIFAR100 datasets, I change the size of output *softmmax* layer to 100. In addition, in the case of VGG-16, I use only one Dense layer before the output, while I keep the same number of dense layers as in the original architecture for the case of GoogLeNet. Moreover, I rescale the CIFAR100 images from 32×32 to 224×224. Finally, I use data augmentation only for CIFAR100 and BraTS'18.

By using different datasets on each CNN, I show how the proposed MARL-based approach is able to provide a data-driven solution. In particular, the inputs for U-Net are BraTS'18 [230–232] and ISIC'18 [233, 234] for semantic segmentation tasks, whereas I consider ImageNet [317] and CIFAR100 [318] for VGG-16 and GoogLeNet as image classification tasks. Although both BraTS'18 and ISIC'18 are bio-medical databases, the former is for brain tumor detection and segmentation, while the latter is concerned with skin lesion boundary segmentation. As shown in the table, since U-Net is used for semantic segmentation, I consider Dice Loss as the loss metric, whereas categorical crossentropy is considered for both VGG-16 and GoogLeNet.

For training the original networks and those designed by Random Search and my approach, I do not apply any forms of optimizations. In other words, for the sake of fair comparison, I only use the original plain CNN architectures and adapt their hyperparameters through these two approaches. Consequently, by original CNNs, I mean the plain architectures and original hyperparameters without any further optimization. Then, all the networks are trained until the validation accuracy does not improve more than 0.01% for 10 successive epochs.

In addition, Table 4.10 shows different types of layers and their corresponding hyperparameters considered for each DCNN. I limit the hyperparameter values to those commonly used in the literature for optimizing each DCNN. As shown in the table, unlike VGG-16 and GoogLeNet,



(a) Exploration

(b) Exploration-exploitation

Figure 4.39 – Convergence of proposed MARL-based approach with respect to reward values

U-Net does not contain any dense (fully connected) layer.

4.9.4 Experimental Results and Discussion

In the following, first, I show the convergence of the proposed MARL-based approach with respect to the reward values. Second, I evaluate the DCNNs designed through my proposed approach compared to the original DCNNs and those designed by random search approach. Third, I show how the number of episodes in the exploration phase may affect the final accuracy and model size of the designed CNN. Finally, I discuss the impact of number of epochs in each episode with respect to the outcome accuracy and model size of the designed CNN.

4.9.4.1 MARL Convergence

Figure 4.39 shows the reward value during the exploration and exploration-exploitation phases. As shown in Figure 4.39a, where actions are taken randomly, the reward values range from -1 to 31, where those with -1 belong to hyperparameter sets unable to satisfy the predefined constraints on training time and loss. The exploration-exploitation phase, shown in Figure 4.39b starts from random actions based on the $\epsilon - greedy$ policy. In the first 50 episodes, where actions are mostly random, small and unsatisfactory rewards are provided. However, for the next episode agents enter the exploitation phase more frequently, as a result of the decayed ϵ value. Yet, based on the value of ϵ agents may enter the exploration phase and pick random action which may result in smaller reward values. This behavior can be observed right before episode 50, where the reward value is close to 10 and, thus, much lower than the rewards obtained through the exploitation. As shown in this figure, when the agents exploit their Q-tables they are able to statistically obtain higher rewards. With the continuation of the exploitation of the Q-tables, agents are finally able to find the set of actions (hyperparameters) for which the reward is maximized (i.e., 36). If for a couple of number of consecutive episodes agents pick the same hyperparameters set, then Q-tables have converged. In the

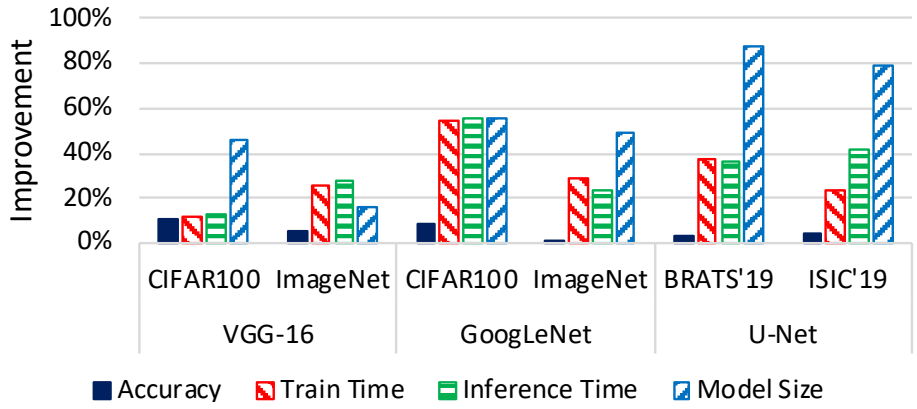


Figure 4.40 – Improvement in accuracy, model size, and training/inference time provided by MARL-based approach compared to Random Search

case of studied CNNs, it is sufficient to consider only two subsequent episodes with the same hyperparameter values are selected by all agents, because in the exploitation phase the agents rely on their past experience while updating it according to the obtained reward. Hence, if the hyperparameters selected in one episode result in an undesirable reward value, then the corresponding Q-values would also degrade. Therefore, in the next episodes of the exploitation phase not all agents take the same hyperparameters. In Figure 4.39b, the optimal actions are probably found at around episode 260. From this point, agents will take the same actions in the exploitation phase unless, due to the $\epsilon - greedy$ policy, another random action is taken.

4.9.4.2 Comparison to Random Search and Original Networks

Table 4.11 compares the accuracy, training/inference time, and model size obtained for each CNN designed through my proposed MARL-based approach, Random Search [283] and the original one (with original hyperparameters in the literature). For Random Search, I find the best hyperparameter set with respect to the same reward function defined in my MARL-based approach (Section 4.9.1.5). To measure the accuracy of GoogLeNet and VGG-16, I consider the Top-1 accuracy, whereas for the case of U-Net, Dice Coefficient is considered. In addition, I consider 2500, 1000, and 4500 random episodes in Random Search and during the exploration phase of my MARL-based approach for U-Net, VGG-16 and GoogLeNet, respectively. Note that these numbers are higher than the minimum number of episodes required for each network and available hyperparameters described in Section 4.9.2.1 (1764, 400, 4356, respectively). Moreover, the number of epochs for which the CNN is trained in each episode is automatically found to be 3, 6, and 6, respectively, for the case of U-Net, VGG-16 and GoogLeNet for both datasets shown in Table 4.11. As shown in the table, for all three DCNNs studied, model size and training/inference time are reduced considerably by my MARL-based approach while the accuracy, in the worst case, is the same as the original DCNNs. Such observation indicates that the proposed reward function (Section 4.9.1.5) is more biased to optimizing the model size, rather than the accuracy. This bias, however, can be changed by adjusting the coefficients

Table 4.11 – Experimental results: Top-1 accuracy for image classification and Dice coefficient for semantic segmentation.

CNN	Dataset	Method	Accuracy (%)	$\frac{tr_{time}}{batch}$ (ms)	Model Size (MB)	$\frac{inf_{time}}{batch}$ (ms)
VGG-16	CIFAR100	Proposed	73.35	23	7	7
		Original	69.32	37	60	14
		Random Search	63.03	26	13	8
	ImageNet	Proposed	68.84	44	53	16
		Original	68.88	74	141	27
		Random Search	63.72	59	63	22
GoogLeNet	CIFAR100	Proposed	73.48	54	22	13
		Original	70.19	98	48	35
		Random Search	64.68	118	49	40
	ImageNet	Proposed	67.96	35	25	15
		Original	67.93	66	54	23
		Random Search	66.75	49	49	17
U-Net	BRATS'18	Proposed	83.24	98	2	33
		Original	83.25	204	138	68
		Random Search	80.16	157	16	52
	ISIC'18	Proposed	82.35	70	3	18
		Original	81.57	123	138	39
		Random Search	77.84	92	14	31

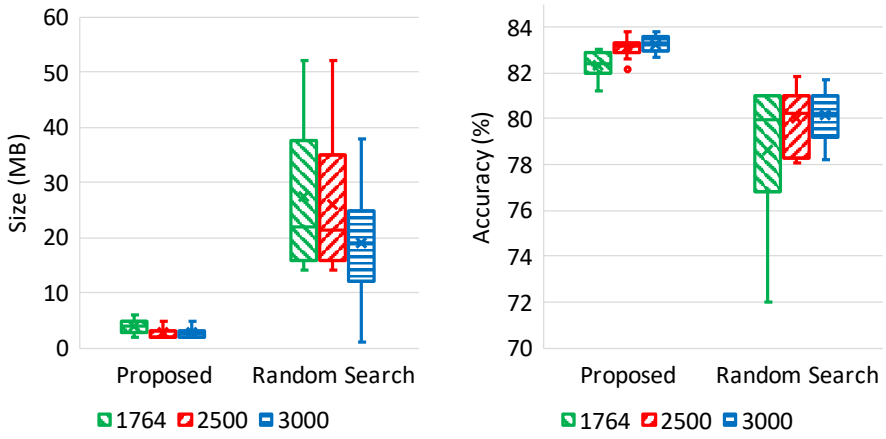
introduced in the reward function.

Conversely, Random Search cannot improve the accuracy in any case, compared to the original CNNs. Figure 4.40 compares the improvements provided by my solution compared to the CNNs designed through Random Search method. In fact, the overhead of my solution compared to Random Search is only $2ms$ for each episode. Therefore, considering that Random Search can also benefit from the same number of episodes I take in both exploration and exploration-exploitation phases, the MARL-based approach can come up with the solution with only 4, 17, and 29 extra seconds for VGG-16, GoogLeNet, and U-Net, respectively. These values, nonetheless, are very pessimistic, since during the exploration-exploitation phase, hyperparameters are not taken randomly, and the training time for each episode improves in comparison to that of Random Search. As a result, MARL-based approach can even spend less wall-clock time for designing DCNNs compared to Random Search.

4.9.4.3 Impact of Number of Episodes in Exploration Phase

As explained in Section 4.9.2.1, in the exploration phase agents take random actions for quite large number of episodes. Since each episode contains several epochs, this phase is the most time-consuming part of the proposed MARL-based hyperparameter optimization approach.

Figure 4.41 shows how the number of episodes in the exploration phase can affect the outcome of the exploration-exploitation phase, and ultimately, the model size and accuracy of the designed DCNN. In the box plot of Figure 4.41, I consider three different numbers of episodes in designing a U-Net for BraTS'18 dataset, each run for 10 times. The first one equals the minimum number of episodes to fill all cells of the Q-tables according to Algorithm 4.4 and Table 4.10, i.e., $(7 \times 3 \times 2) \times (7 \times 3 \times 2)$ for two consecutive Convolution layers. Even with this minimum value, the proposed MARL-based approach is able to find a quite satisfying



(a) Model size

(b) Accuracy

Figure 4.41 – Impact of number of episodes in exploration phase

hyperparameter set with respect to the model size and accuracy. However, compared to the other two larger number of episodes, there is more variation in the outcome. Thus, I suggest to use more episodes in the exploration phase than the minimum required to fill all Q-tables. Nevertheless, as depicted in Figure 4.41, although increasing the number of episodes to 2500 provides more consistent results, it only improves trivially when using 3000 episodes. This behavior is, in fact, desirable because it means with a very small increase from the minimum number of episodes, the agents are not only able to improve the outcome, but also can provide more statistically consistent results. Hence, there is no need to further increase time overhead of the exploration phase.

For comparison, I apply random search for the same three different numbers of episodes. As shown in the figure, although with increasing the number of episodes model size and accuracy improves, it is not able to defeat the outcome of the MARL-based solution, nor it can provide more statistically consistent results. My results reveal that similar trends exist for the other DCNNs and datasets considered in this work.

4.9.4.4 Impact of Number of Epochs in Episodes

As discussed in Section 4.9.2.1, in each episode, I train the designed CNN for only a limited number of epochs, automatically found by my proposed solution. Figure 4.42 shows how this number can affect the outcome of the exploration-exploitation phase and, ultimately, the model size and accuracy of the designed U-Net, VGG-16, and GoogLeNet. In this figure, I show U-Net outcome for BraTS'18 dataset, whereas, CIFAR100 dataset is considered for both VGG-16 and GoogLeNet. Similar results can be shown for other datasets considered in this work. As shown in Figure 4.42, the optimal number of training epochs in each episode is less than the maximum number initially defined in Section 4.9.2. Moreover, by considering number of epochs less than 3, 6, and 6, for U-Net, GoogLeNet, and VGG-16, respectively, the

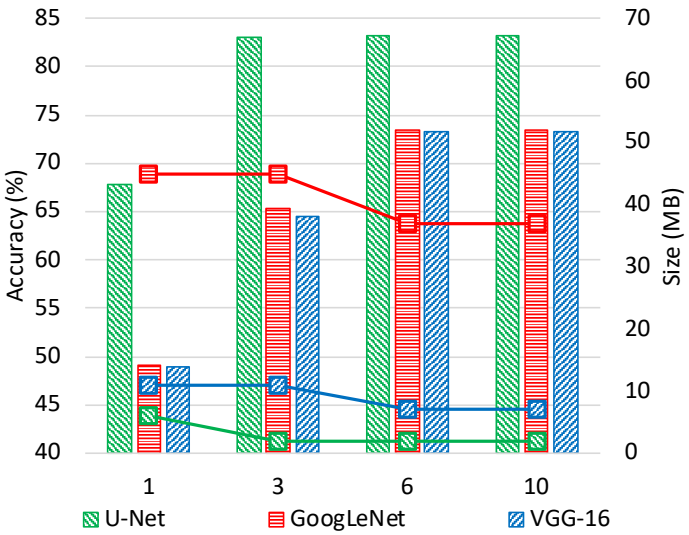


Figure 4.42 – Impact of number of epochs in an episode

agents are not able to well assess the hyperparameters selected at each episode due to the insufficient change in validation accuracy and, thus the reward signal. Moreover, the impact of the number of epochs is more evident in the accuracy of the designed CNN than in the model size, because the model size is known from the first epoch and does not change with increasing the number of epochs.

4.10 Summary

Conventional runtime management and design space search methods are not flexible enough to address very large and dynamic problems, such as heuristics, are not able to address. Persistence in using these incompetent approaches, such as heuristics, leaves many crucial aspects of multi-objective management of multiprocessor systems unaddressed. Adaptive fan speed control along with the well-known DTM techniques that leverage system-level parameters is one of these aspects. In spite of the undeniable role of adaptive cooling in lifetime reliability, power consumption, and performance, it has not been holistically addressed. One of the main reasons is that once this set of design parameters is added to the already-existing parameters, such as DVFS, task allocation, and thread mapping, the design space becomes too large and can be only partially investigated by the traditional approaches. This problem is even more challenging with the new trending applications and services, such as video streaming and deep learning (DL). These applications incorporate a great number of internal parameters that need to be set either at design time or dynamically tuned at runtime. Consequently, the design space is too large to handle through traditional approaches. In addition to these internal parameters, the workload of these applications are input dependent, i.e., workload can vary abruptly over time. These input data, on the other hand, usually are very difficult to model or predict in case of streaming, due to the content variation.

Nonetheless, Reinforcement Learning (RL) provides designers with model-free algorithms, such as Q-Learning (QL), where one or multiple learning agents can directly interact with the large and dynamic problem and learn from consequences of each design parameter (so-called action) in different situations (i.e, states). RL do not require any prior knowledge of the application and the underlying system, and unlike heuristics, is mathematically grounded.

In this chapter, after having assessed two trending applications, namely HEVC streaming and CNNs, with respect to their internal design parameters and their impact on application and system-wide objectives, I applied RL to several corresponding real-life problems.

First, I proposed an RL-based approach for dynamic thermal management of multiprocessor platforms with proactive fan speed control for performance maximization and fan power minimization under thermal constraints. The proposed RL approach is able to dynamically adjust the fan speed, determine the required number of threads, assign processing cores to each thread, and perform DVFS at runtime. My results showed up to 40% decrease in fan power consumption when compared to the state-of-the-art DTM policy with a fixed fan speed, and up to 19% improvement in performance with no further fan power consumption compared to a state-of-the-art reactive DTM policy.

Second, I addressed runtime workload allocation of HEVC encoding on heterogeneous MPSoCs through RL. In this approach, I dealt with a multi-user environment where multiple streams were to encode simultaneously. The goal of the RL agent was to maximize throughput while meeting a user-defined power constraint by learning optimum stream allocation as well as optimum core and accelerator frequency from the total throughput and power consumption of the system. In the problem definition, I assumed that streams could have different resolutions and requirements in terms of motion search area. My proposal achieved 20% higher throughput, and converges 1.5x faster to the optimal solution than a heuristic load balancing strategies.

Third, I leveraged RL for joint optimization of application- and system-level parameters for multi-objective runtime management of multiprocessor systems. In particular, I presented a comprehensive quality-aware power and thermal management approach multi-user HEVC streaming. I considered five different objectives and constraints in this work, including encoding time, video quality, video compression, power consumption, and peak temperature, and let the QL agent learn the best actions by directly interacting with multiple different streams running on a multi-core servers. The efficacy of the proposed RL-based solution was evaluated against the state-of-the-arts in different scenarios. Overall, the proposed approach outperformed state-of-the-arts [276] mainly due to its awareness of the content variation within and across videos. On average, for the most realistic scenario, my RL-based approach improved BD-PSNR and BD-rate [319] by 0.54 dB, and 8%, respectively, and reduced the encoding time, power consumption, and average temperature by 15.3%, 13%, and 10%, respectively. Moreover, my approach improved BD-PSNR and BD-rate compared to the reference software (HM) by 1.19 dB and 24%, respectively, without any encoding time degradation.

When the design space is too large, using a single learning agent to interact with the environment leads to either sub-optimal solutions, or very time-consuming exploration phase. To address this issue, I presented MAMUT, a novel multi-agent RL (MARL) solution for efficient real-time multi-user video transcoding. My solution enabled fine-grained and more accurate search by splitting the design space in different smaller sub-spaces, each of them explored by one different agent, while working cooperatively with the others to decide the next actions to take. In my design, agents tackled both intrinsic application-l and system-level parameters (number of threads and processor frequency). My solution outperformed the conventional single-agent RL (SARL) approach with respect to both energy consumption (up to 7%) and less QoS violations (up to 5x), while satisfying restrictions in power and compression. Also, I compared MAMUT to SoA, as well as to a new proposed content-aware heuristic policy. MARL saved up to 24% of energy consumption and reduced QoS violations by up to 8x.

Finally, in this chapter, I showed how RL can be used for efficient design space search. In this context, I have addressed hyperparameter optimization of Deep CNNs (DCNNs) through a MARL-based approach. This approach used different QL agents per layer to split the design space into smaller independent sub-spaces to provide faster, yet accurate design space search. In contrast to the state of the art, my approach was not limited to particular types of layers, could scale well with the depth of CNNs without any search time overhead, and could optimize the CNN hyperparameters with respect to any arbitrary set of constraints and objectives, thus eliminating the time-consuming and manual human effort. I assessed my MARL-based approach by applying it to three different CNNs, VGG-19, GoogLeNet, and U-Net, each with two different datasets. The results have shown that, compared to the original CNNs, the MARL-based approach can reduce model size, training time, and inference time by up to, respectively, 83x, 52%, and 54% without any degradation in accuracy. Moreover, my approach can improve accuracy, training time, inference time, and model size compared to the Random Search method by up to 10%, 54%, 55%, and 87%, respectively.

5 Conclusion and Future Work

To conclude my thesis, I first summarize its most remarkable contributions. Thereafter, I provide important notes concerning the future research direction based on the results and the findings accomplished.

5.1 Summary of Contributions

The main goal of this thesis was to reveal several already-existing and emerging challenges in multi-objective management of multiprocessor systems, and to address them through novel solutions.

5.1.1 Heuristic Multi-Objective Management of Multiprocessor Systems

In Chapter 2, I have addressed multi-objective runtime management of multiprocessor systems through heuristics with the focus on lifetime reliability. In particular, first I have proposed TheSPoT, a thermal stress-aware power and performance management framework, which takes into account thermal stress as the new dominant factor in lifetime reliability of modern multiprocessor systems. TheSPoT provides a low-overhead heuristic multi-level runtime management framework that leverages core consolidation and deconsolidation, thread migration, and DVFS at different intervals. I have compared TheSPoT against conventional heuristic DTM approaches. The results showed that TheSPoT can improve the mean time-to-failure by 47%, on average, thanks to considering thermal cycling and spatial and temporal thermal gradients, in addition to other well-known factors in lifetime reliability, such as TDDB and Electromigration. Moreover, I have formulated a convex optimization problem for DVFS in TheSPoT. My experiments on 4-, 8-, and 16-core processors have indicated that, unlike the optimization approach, the heuristic TheSPoT can be applied as low-overhead runtime management when the number of cores scales up. In addition, I have assessed TheSPoT under large workload variation showing that while state-of-the-art DTM cannot prevent the multiprocessor system from experiencing large thermal cycles, TheSPoT alleviates the thermal stress.

As the second main contribution in this chapter, I have scrutinized one of the most recent cooling technologies, the micro-scale two-phase liquid cooling thermosyphon. Having highlighted its potential and limits, I have optimized the thermosyphon design with respect to the features of the workload and the underlying multi-core processor. Then, I have proposed a cooling-aware heuristic DTM for performing workload allocation and DVFS, as well as dynamically tuning the run-time parameters of the thermosyphon, such as water flow-rate. The proposed heuristic DTM along with the design optimization of thermosyphon led to reducing the thermal hot spots and spatial gradients by up to $10°C$, and 45%, respectively, while requiring at least 45% less cooling power for the chiller, compared to state-of-the-art DTM and thermosyphon solutions.

Publications: One journal and one conference papers constitute the main publications achieved by this chapter of my thesis, as follows:

- The proposed TheSPoT framework has been published in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD) [320].
- The cooling-aware DTM method has been presented at Design Automation and Test in Europe (DATE) [321].
- I have also collaborated with Heat and Mass Transfer Laboratory (LTCM), EPFL, Switzerland, for the development of the thermosyphon from a computer and electrical perspective, presented at IEEE Intersociety Conference on Thermal and Thermomechanical Phenomena in Electronic Systems (ITHERM) [20].

5.1.2 Machine Learning for Runtime Management of Time-Variant Workloads

In Chapter 3, I have tackled power and performance runtime management of time-variant workloads, especially with a focus on new trending applications, such as real-time HEVC streaming. In particular, by detailing the impact of video content variations on application and system objectives, I have shown that for such non-deterministic workload variation traditional heuristics are inadequate. Thus, I have proposed a machine learning framework for workload prediction and throughput estimation under different system-level design parameters, such as operating frequency and number of active cores. The proposed ML-based framework takes hardware events as input to classical machine learning algorithms such as Kmeans++ for clustering and Random Forest for both classification and regression. The framework outputs the predicted future workload and the corresponding estimated throughput for each available system configuration. For comparison, I developed a content-aware application-specific heuristic that, based on the video contents, determines the number of threads and sets the operating frequency. Moreover, I adapted a state-of-the-art neural network(NN-)-based approach that leverages hardware events for workload prediction and DVFS. I have also implemented a state-of-the-art load balancing (LB) approach that uses heuristics for

runtime management of multimedia applications. Compared to the NN-based approach, my ML framework could reduce the QoS violations (in terms of throughput) by at least 3.4x, while decreasing power consumption by 15% due to its more accurate predictions. Also, the ML framework decreased the power consumption of the target multi-core server by 33% and 12% compared to the LB and proposed heuristic approaches, respectively, while enhancing the QoS violations by 4.5x and 4.0x.

Publications: The publications achieved throughout this chapter are as follows:

- The ML-framework for workload prediction and throughput estimation of time-variant applications has been presented at International Conference on Very Large Scale Integration (VLSI-SoC) [322].
- The proposed content-aware heuristic management of HEVC encoding has been presented as a part of a conference paper at Design Automation and Test in Europe (DATE) [323].
- I have also collaborated with a visiting PhD student at ESL from UFRN, Brazil, in developing a performance counter-based profiling tool for next-generation workloads published in Journal of Energies [324].

5.1.3 Reinforcement Learning for Multi-Objective Management of Multiprocessor Systems

In chapter 4, I have addressed runtime management and design space search for extremely large and dynamic problems. For this purpose, I have adapted Reinforcement Learning in various forms to conform with different challenges in multi-objective management of multiprocessor systems. Moreover, I have shown that once the design space becomes extremely large or dynamic, RL is a more promising solution than the traditional heuristics. Specifically, I have explicated that for new trending applications and services such as real-time HEVC streaming and CNNs, the design space does not simply contain well-studied system-level parameters, but it also includes several application-level parameters set at design-time or dynamically adjusted at runtime.

First, I have leveraged RL to incorporate fan speed, as a design parameter, into DTM techniques. In particular, the proposed approach have enabled dynamic adaptation of fan speed along with other conventional DTM techniques, such as DVFS and thread allocation, to maximize performance under thermal constraints, while minimizing the cooling power. I have assessed the efficacy of the proposed approach on thermal test chip composed of 16 power cells. I have also imitated the workload variation of HEVC streaming by generating highly time-variant power traces. For comparisons, I have developed an RL-based DTM approach, excluding adaptive fan speed control from the available actions to the agent. Compared to this approach,

the proposed RL-based proactive fan speed control scheme can save the cooling power by up to 40%, with less than 1% performance degradation and no thermal violations. In addition, I have compared the proposed solution to the state-of-the-art heuristic DTM approach that reactively adjusts fan speed. The results showed that RL achieves up to 19% performance enhancement without any further cooling power. Moreover, unlike the reactive fan speed control method, RL does not violate thermal constraints.

Second, I have addressed workload allocation of HEVC streaming on heterogeneous multi-processor systems through RL. The proposed approach is able to consider the input video resolution and the preset application-level parameters and accordingly allocate the streams to the general-purpose cores and hardware accelerators, while applying DVFS. The proposed solution has been designed to maximize the number of streams that can be concurrently encoded on a power-constrained heterogeneous platform. Compared to the state-of-the-art load balancing approach, RL achieves 20% higher throughput, and converges 1.5x faster to the optimal solution, i.e., maximum number of streams allowed to be encoded simultaneously.

Third, I have addressed the joint optimization of application- and system-level parameters for runtime management of multiprocessor systems through RL. In particular, I have focused on HEVC encoding application as it incorporates several wide-range internal parameters that makes the design space too large to be tackled by conventional heuristics. Moreover, content-dependent workload variations add to the complexity of QoS-aware multi-objective management problem. The proposed RL-based approach is able to dynamically tune the encoder parameters along with DVFS. I have considered video quality, video compression, encoding time, power consumption, and thermal profile as the objectives and constraints. I have evaluated the efficacy of RL compared to state-of-the-arts on a multi-core server while using the HEVC reference software. The results showed that RL improves the video quality and compression by 0.54 dB, and 8%, respectively, and reduces the encoding time, power consumption, and average temperature by 15.3%, 13%, and 10%, respectively. Moreover, my approach improves video quality and compression compared to the reference software by 1.19 dB and 24%, respectively, without any encoding time degradation.

Moreover, I have used RL for real-time multi-user HEVC streaming having considered number of threads as an additional design parameters to take advantage of parallelization features introduced in HEVC standard. In particular, I have defined multiple learning agents and used them within a novel cooperative MARL-based method (MAMUT) to deal with a large design space. Through the proposed learning agents, I have enabled splitting the design space into smaller independent sub-spaces to accomplish faster, yet accurate exploration, compared to single-agent RL (SARL). The proposed MARL-based approach dynamically sets HEVC encoder parameters, specifies the number of threads, and applies DVFS to maximize the quality and number of videos that can be simultaneously served under a predefined power and bandwidth constraints. I have assessed the proposed MARL-based approach in comparison with several methods. In this context, I have developed a content-aware heuristic for determining the number of threads, DVFS, and setting internal parameters of the encoder.

I have also implemented a SARL-based approach with the same design space available to MARL. In addition, I have adapted a state-of-the-art heuristic that sets the number of threads and HEVC parameters as well as DVFS. The results showed that although MAMUT is 15x faster, it still outperforms the conventional SARL approach with respect to QoS violation by up to 5x, while satisfying restrictions in power and compression. Also, MAMUT was shown to achieve 8x less QoS violation compared to both proposed and state-of-the-art heuristic approaches.

Finally, I have proposed a MARL-based framework to address hyperparameter optimization of CNNs. Having used multiple learning agents and assigned them to each layer of Deep CNNs, I have split the design space into multiple smaller sub-spaces. Each sub-space could be accurately explored by the assigned agents, while the agents could share their experiences with the next successive agent to cooperatively select optimal hyperparameter values for each layer. The proposed MARL-based framework was compared to the original CNNs, the MARL-based approach can reduce model size, training time, and inference time by up to, respectively, 83x, 52%, and 54% without any degradation in accuracy. Moreover, my approach could improve accuracy, training time, inference time, and model size compared to the random search method by up to 10%, 54%, 55%, and 87%, respectively.

Publications:

- Thermal characterization of HEVC video encoding workloads has been presented at International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS) [158].
- The proposed approach for adaptive fan speed control has been published in the Proceedings of Design, Automation and Test in Europe Conference and Exhibition [325].
- The RL-based workload allocation framework for HEVC streaming has been presented at International Symposium on Circuits and Systems (ISCAS) [326].
- The proposed RL-based framework dealing with joint optimization of application- and system-level parameters has been published as a journal paper in IEEE Transactions on Parallel and Distributed Systems (TPDS) [327].
- The MARL-based approach has been presented in Design, Automation and Test in Europe Conference and Exhibition [328].
- I have also collaborated for extending the latter to a journal paper, published in IEEE Transactions on Parallel and Distributed Systems (TPDS) [329].
- Finally, the proposed framework for hyperparameter optimization of CNNs has been submitted to the journal of IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD).

5.2 Discussion on RL Use in Different Optimization Problems

To satisfy the ever-increasing demand for higher performance while having power consumption and thermal profile under control, multi-objective runtime management is crucial for today's multiprocessor systems. Traditionally, heuristics have been widely used in different manners for this purpose thanks to their simplicity and low-overhead run-time execution. As shown in Chapter 2, heuristics provide more practical solutions compared to optimization approaches for multi-objective runtime management of multiprocessor systems. However, these simple heuristics are insufficient for tackling emerging challenges of new trending applications and services, such as real-time streaming and Deep Learning design. Since heuristics are not flexible enough in coping with very complex and dynamic environments, they are not promising candidates for runtime management of such applications.

Classical machine learning algorithms, on the contrary, provide opportunities to directly learn from the input data. As demonstrated in Chapter 3, a general-purpose ML framework, if well designed, can outperform not only very well-known and trusted state-of-the-art heuristics, but also is superior to other application-specific heuristics. Although unlike heuristics, machine learning does not require specific prior knowledge or intuitions, ML design is not as straightforward as heuristics generally are. For developing machine learning-based solution, usually a sufficiently large amount of training data is necessary. Collecting these data is not always feasible. Other technical challenges, such as overfitting, also necessitate sound knowledge of machine learning for runtime management of multiprocessor systems. Hence, classical machine learning is not always the best alternative method for heuristics. In fact, complexity of the problem is the main driving factor to move towards machine learning, yet machine learning may not suit all problems.

In contrast to classical supervised and unsupervised learning, Reinforcement Learning (RL) deals with multi-objective runtime management of multiprocessor systems from a completely different perspective. RL enables learning an optimal behavior in very complex and dynamic environments by directly interacting with the problem. As shown in Chapter 4, once the design space becomes extremely large, neither heuristics nor classical machine learning may satisfy design objectives and constraints. Especially for cases where joint optimization of application- and system-level parameters are required, design space exploration can be addressed through RL. Although RL does not require collecting the training data it needs to sufficiently explore the design space to gain experience from the consequences of different actions in various situations. This phase, depending on the design space and dynamism of the problem may be time-consuming. In such cases, cooperative Multi-Agent RL (MARL) is a promising approach enabling splitting the design space into smaller sub-spaces for faster, yet accurate search. Despite its more efficient design space search, MARL induces more implementation complexity than single-agent RL (SARL) because the agents need to communicate. All in all, in spite of the fact that both SARL and MARL are paid more attention nowadays, they require a few important preliminaries. These preliminaries include proper state space definition, equipping the learning agent with tools that provide real-time observation from the environment,

and rewarding the agents properly according to the quality of their actions representing the design objectives and constraints. Obviously, due to these challenges, RL may not be worth it for simple problems, even though it may still outperform conventional approaches such as well-studied heuristics.

5.3 Future Work

In what follows, I highlight several research lines and topics that can be followed as future research in the same direction as of my contributions in this thesis. In particular, in Sections 5.3.1, 5.3.2, and 5.3.3 I explain three interesting research problems to tackle as the continuation of different parts of this thesis. Then, in Section 5.3.4 I introduce one important research line on top of the topics I have covered in my thesis, but in the same direction.

5.3.1 Thermal Stress-Aware Lifetime Reliability of 3D SoCs

In this thesis, I focused on 2D SoCs, however, 3D SoCs offer higher density, lower latency, and higher bandwidth. One of the main concerns regarding 3D SoCs is thermal hot spots as the thermal dissipation through natural convection is restricted. Although since its introduction, the lifetime reliability of 3D SoCs have been addressed through peak temperature reduction, thermal-stress have neither been studied nor addressed in the literature. This is an interesting topic for the continuation of my contributions in Chapter 2. In fact, a comprehensive study of thermal stress impact on lifetime reliability of 3D SoCs is necessary. For instance, in 3D SoCs, temperature of a particular cell is affected not only by the adjacent cells, but also by the one beneath. Spatial thermal gradients, thus, should be considered accordingly. Moreover, it has not been studied if thermal cycling is still a dominant factor of lifetime reliability in mid- and low-range temperature, as it is in 2D SoCs.

5.3.2 Joint Optimization of Application- and System-Level Parameters in Multi-Application Platforms

In Chapter 4, where I have addressed joint optimization of application- and system-level parameters, I have considered one type of application, even though with multiple instances. An interesting problem for research in this direction is multi-application platforms, where different types of application can be processed simultaneously. In such a scenario, because the application-level parameters can affect the power consumption, and thermal profile of the target multiprocessor system, they need to be adapted according to other applications requirements running at the same time. Thus, the design space increases hyper-exponentially, and not only traditional approaches are insufficient, but also the efficiency of SARL is in question.

5.3.3 Design Automation of Deep Learning Architectures

In Chapter 4, I tackled hyperparameter optimization of CNNs through MARL. However, Deep Learning optimization can be extended to other types of networks, such as Recurrent Neural Networks (RNNs) and Deep Learning Recommendation Models (DLRMs). The former is widely used in Natural Language Processing and Machine Translation, while the latter helps businesses to build up recommendation systems to maximize the Click Through Rate (CTR) and, ultimately, their revenue. My contribution in Chapter 4 can be extended as a future work in several different ways. One area is automating hyperparameter optimization of the existing networks. The other area is to build these models from scratch, i.e., types of layers, their connections, and their hyperparameters are designed based on the input and the desired output. For both cases, RL is a promising solution. However, due to the intrinsic differences among CNNs, RNNs, and DLRMs, the best approach most probably is not the same as the one I proposed in this thesis. Currently, I am working on automating Neural Architecture Search (NAS) for DLRMs through RL.

5.3.4 Multi-Objective Runtime Management of Fog Computing Systems

Fog Computing is a highly virtualized platform that enables expensive computation at the edge of the network, serving a large number of heterogeneous devices and applications. Although Fog Computing provides an appropriate platform for many Internet of Things (IoT) applications and services, there are still several unaddressed or partially addressed challenges. Since Fog is located at the edge of network and it hosts numerous different applications, its network is extremely heterogeneous. For IoT at large scale, managing such a network is challenging and requires novel runtime management techniques. Moreover, QoS in Fog encompasses several metrics, such as reliability, latency, and capacity. Traditional approaches for reliability management, such as checkpointing and rescheduling do not suit many latency-sensitive applications such as real-time streaming. Capacity, in terms of network bandwidth and storage, also need to be satisfied. Nevertheless, due to the excessive dynamism of the Fog environment, addressing all the QoS requirements is not trivial at all. Besides, application-aware provisioning is another significant challenge of Fog, where due to the mobility of the end node some important metrics including bandwidth, storage, computation and latency can change constantly.

As a consequence, multi-objective runtime management of Fog systems is inevitable. However, the dynamic environment makes it very challenging to address it through well-known methods. Based on the insights attained from my thesis, RL can be a promising approach to address multi-objective runtime management of Fog Computing systems.

Bibliography

[1] R. Zurawski, *Embedded Systems Handbook: Embedded systems design and verification*. CRC press, 2018, vol. 6.

[2] A. K. Das, A. Kumar, B. Veeravalli, and F. Catthoor, *Reliable and Energy Efficient Streaming Multiprocessor Systems*. Springer, 2018.

[3] L. Benini, A. Bogliolo, and G. De Micheli, "A survey of design techniques for system-level dynamic power management," *IEEE transactions on very large scale integration (VLSI) systems*, vol. 8, no. 3, pp. 299–316, 2000.

[4] M. D. Hill and M. R. Marty, "Amdahl's law in the multicore era," *Computer*, vol. 41, no. 7, pp. 33–38, 2008.

[5] D. H. Woo and H.-H. S. Lee, "Extending amdahl's law for energy-efficient computing in the many-core era," *Computer*, 2008.

[6] J. Li, J. F. Martinez, and M. C. Huang, "The thrifty barrier: Energy-aware synchronization in shared-memory multiprocessors," in *10th International Symposium on High Performance Computer Architecture (HPCA'04)*. IEEE, 2004, pp. 14–23.

[7] M. Pedram and Q. Wu, "Design considerations for battery-powered electronics," in *Proceedings of the 36th annual ACM/IEEE Design Automation Conference*, 1999, pp. 861–866.

[8] J. Whitney and P. Delforge, "Scaling up energy efficiency across the data center industry: Evaluating key drivers and barriers," *Issue Paper No. IP*, pp. 14–08, 2014.

[9] A. Beloglazov and R. Buyya, "Energy efficient allocation of virtual machines in cloud data centers," in *2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*. IEEE, 2010, pp. 577–578.

[10] Microprocessor power impacts. [Online]. Available: https://www.glsvlsi.org/archive/glsvlsi10/pant-GLSVLSI-talk.pdf

[11] A. K. Coskun, T. S. Rosing, and K. C. Gross, "Temperature management in multiprocessor SoCs using online learning," in *Design Automation Conference, 2008. DAC 2008. 45th ACM/IEEE*. IEEE, 2008, pp. 890–893.

[12] R. Viswanath, V. Wakharkar, A. Watwe, V. Lebonheur *et al.*, "Thermal performance challenges from silicon to systems," 2000.

[13] T. Ishihara and H. Yasuura, "Voltage scheduling problem for dynamically variable voltage processors," in *Proceedings of the 1998 international symposium on Low power electronics and design*. ACM, 1998, pp. 197–202.

[14] N. K. Jha, "Low power system scheduling and synthesis," in *IEEE/ACM International Conference on Computer Aided Design. ICCAD 2001. IEEE/ACM Digest of Technical Papers (Cat. No. 01CH37281)*. IEEE, 2001, pp. 259–263.

[15] J. Donald and M. Martonosi, "Techniques for multicore thermal management: Classification and new exploration," *ACM SIGARCH Computer Architecture News*, vol. 34, no. 2, pp. 78–88, 2006.

[16] K. Skadron, M. R. Stan, W. Huang, S. Velusamy, K. Sankaranarayanan, and D. Tarjan, "Temperature-aware microarchitecture," *ACM SIGARCH Computer Architecture News*, vol. 31, no. 2, pp. 2–13, 2003.

[17] A. K. Coskun, T. S. Rosing, K. Mihic, G. De Micheli, and Y. Leblebici, "Analysis and optimization of MPSoC reliability," *Journal of Low Power Electronics*, vol. 2, no. 1, pp. 56–69, 2006.

[18] T. Chantem, X. S. Hu, and R. P. Dick, "Temperature-aware scheduling and assignment for hard real-time applications on MPSoCs," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 19, no. 10, pp. 1884–1897, 2011.

[19] S. H. Gunther, "Managing the impact of increasing microprocessor power consumption," *Intel Technology Journal*, 2001.

[20] A. Seuret, A. Iranfar, M. Zapater, J. Thome, and D. Atienza, "Design of a two-phase gravity-driven micro-scale thermosyphon cooling system for high-performance computing data centers," in *2018 17th IEEE Intersociety Conference on Thermal and Thermomechanical Phenomena in Electronic Systems (ITHERM)*. IEEE, 2018, pp. 587–595.

[21] H. Honda and J. Wei, "Enhanced boiling heat transfer from electronic components by use of surface microstructures," *Experimental Thermal and Fluid Science*, vol. 28, no. 2-3, pp. 159–169, 2004.

[22] I. SANDVINE. (2015) Global internet phenomena report. 2016. [Online]. Available: https://www.sandvine.com/hubfs/downloads/archive/2016-global-internet-phenomena-report-latin-america-and-north-america.pdf

[23] Sandvine, "The global internet phenomena report," Sandvine, Tech. Rep., 2019. [Online]. Available: https://www.sandvine.com/phenomena

[24] F. Behmann. (2009) Embedded. com-the ITRS process roadmap and nextgen embedded multicore SoC design.

[25] A. W. Service, "Trends in AWS spending 2019," institution, Tech. Rep., 2019. [Online]. Available: https://www.cloudhealthtech.com/blog/trends-aws-spending-2019

[26] J. Huang, A. Raabe, C. Buckl, and A. Knoll, "A workflow for runtime adaptive task allocation on heterogeneous MPSoCs," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2011*. IEEE, 2011, pp. 1–6.

[27] Z. J. Jia, A. D. Pimentel, M. Thompson, T. Bautista, and A. Núñez, "Nasa: A generic infrastructure for system-level MPSoC design space exploration," in *2010 8th IEEE Workshop on Embedded Systems for Real-Time Multimedia (ESTIMedia),*. IEEE, 2010, pp. 41–50.

[28] S. Murali, M. Coenen, A. Radulescu, K. Goossens, and G. De Micheli, "Mapping and configuration methods for multi-use-case networks on chips," in *Asia and South Pacific Conference on Design Automation, 2006.* IEEE, 2006, pp. 6–pp.

[29] J. Hu and R. Marculescu, "Energy-and performance-aware mapping for regular NoC architectures," *IEEE Transactions on computer-aided design of integrated circuits and systems*, vol. 24, no. 4, pp. 551–562, 2005.

[30] C. A. Marcon, E. I. Moreno, N. L. Calazans, and F. G. Moraes, "Evaluation of algorithms for low energy mapping onto NoCs," in *2007 IEEE International Symposium on Circuits and Systems.* IEEE, 2007, pp. 389–392.

[31] J. Zhou, T. Wei, M. Chen, J. Yan, X. S. Hu, and Y. Ma, "Thermal-aware task scheduling for energy minimization in heterogeneous real-time MPSoC systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 35, no. 8, pp. 1269–1282, 2016.

[32] M. Goraczko, J. Liu, D. Lymberopoulos, S. Matic, B. Priyantha, and F. Zhao, "Energy-optimal software partitioning in heterogeneous multiprocessor embedded systems," in *Proceedings of the 45th annual design automation Conference.* ACM, 2008, pp. 191–196.

[33] Y. Wang, K. Li, H. Chen, L. He, and K. Li, "Energy-aware data allocation and task scheduling on heterogeneous multiprocessor systems with time constraints," *IEEE Transactions on Emerging Topics in Computing*, vol. 2, no. 2, pp. 134–148, 2014.

[34] W. Zhang, E. Bai, H. He, and A. M. Cheng, "Solving energy-aware real-time tasks scheduling problem with shuffled frog leaping algorithm on heterogeneous platforms," *Sensors*, vol. 15, no. 6, pp. 13 778–13 804, 2015.

[35] A. Lifa, P. Eles, and Z. Peng, "On-the-fly energy minimization for multi-mode real-time systems on heterogeneous platforms," in *2015 13th IEEE Symposium on Embedded Systems For Real-time Multimedia (ESTIMedia).* IEEE, 2015, pp. 1–10.

[36] A. Aminifar, S. Samii, P. Eles, and Z. Peng, "Control-quality driven task mapping for distributed embedded control systems," in *2011 IEEE 17th International Conference on*

Embedded and Real-Time Computing Systems and Applications, vol. 1. IEEE, 2011, pp. 133–142.

[37] J. Paul, W. Stechele, B. Oechslein, C. Erhardt, J. Schedel, D. Lohmann, W. Schröder-Preikschat, M. Kröhnert, T. Asfour, É. Sousa *et al.*, "Resource-awareness on heterogeneous MPSoCs for image processing," *Journal of Systems Architecture*, vol. 61, no. 10, pp. 668–680, 2015.

[38] J. Huang, A. Raabe, C. Buckl, and A. Knoll, "Runtime adaptive allocation of dynamically mixed tasks on a heterogeneous MPSoC platform," in *2010 Conference on Design and Architectures for Signal and Image Processing (DASIP)*. IEEE, 2010, pp. 34–41.

[39] F. Wronski, E. W. Brião, and F. R. Wagner, "Evaluating energy-aware task allocation strategies for MPSoCs," in *From Model-Driven Design to Resource Management for Distributed Embedded Systems*. Springer, 2006, pp. 215–224.

[40] W. Quan and A. D. Pimentel, "A hybrid task mapping algorithm for heterogeneous MPSoCs," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 14, no. 1, p. 14, 2015.

[41] ——, "A system-level simulation framework for evaluating task migration in MPSoCs," in *Proceedings of the 2014 Int. Conference on Compilers, Architecture and Synthesis for Embedded Systems*. ACM, 2014, p. 13.

[42] ——, "Scenario-based run-time adaptive MPSoC systems," *Journal of Systems Architecture*, vol. 62, pp. 12–23, 2016.

[43] ——, "A run-time self-adaptive resource allocation framework for MPSoC systems," in *ECCTD, 2015 European Conference on*. IEEE, 2015, pp. 1–4.

[44] ——, "Towards self-adaptive MPSoC systems with adaptivity throttling," in *2015 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS)*. IEEE, 2015, pp. 157–164.

[45] A. Pahlavan, M. Momtazpour, and M. Goudarzi, "Variation-aware server placement and task assignment for data center power minimization," in *2012 IEEE 10th International Symposium on Parallel and Distributed Processing with Applications*. IEEE, 2012, pp. 158–165.

[46] ——, "Data center power reduction by heuristic variation-aware server placement and chassis consolidation," in *The 16th CSI International Symposium on Computer Architecture and Digital Systems (CADS 2012)*. IEEE, 2012, pp. 150–155.

[47] A. Pahlevan, P. G. Del Valle, and D. Atienza, "Exploiting cpu-load and data correlations in multi-objective vm placement for geo-distributed data centers," in *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2016, pp. 1333–1338.

[48] A. Pahlevan, "Multi-objective system-level management of modern green data centers," EPFL, Tech. Rep., 2019.

[49] A. Pahlevan, X. Qu, M. Zapater, and D. Atienza, "Integrating heuristic and machine-learning methods for efficient virtual machine allocation in data centers," *IEEE transactions on computer-aided design of integrated circuits and systems*, vol. 37, no. 8, pp. 1667–1680, 2017.

[50] K. Haghshenas, A. Pahlevan, M. Zapater, S. Mohammadi, and D. Atienza, "Magnetic: Multi-agent machine learning-based approach for energy efficient dynamic consolidation in data centers," *IEEE Transactions on Services Computing*, 2019.

[51] V. Hanumaiah, S. Vrudhula, and K. S. Chatha, "Performance optimal online DVFS and task migration techniques for thermally constrained multi-core processors," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 30, no. 11, pp. 1677–1690, 2011.

[52] M. Kamal, A. Iranfar, A. Afzali-Kusha, and M. Pedram, "A thermal stress-aware algorithm for power and temperature management of MPSoCs," in *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition*. EDA Consortium, 2015, pp. 954–959.

[53] S. Murali, A. Mutapcic, D. Atienza, R. Gupta, S. Boyd, and G. De Micheli, "Temperature-aware processor frequency assignment for MPSoCs using convex optimization," in *Proceedings of the 5th IEEE/ACM international conference on Hardware/software codesign and system synthesis*, 2007, pp. 111–116.

[54] C. Isci, A. Buyuktosunoglu, C.-Y. Cher, P. Bose, and M. Martonosi, "An analysis of efficient multi-core global power management policies: Maximizing performance for a given power budget," in *2006 39th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'06)*. IEEE, 2006, pp. 347–358.

[55] K. Neshatpour, W. Burleson, A. Khajeh, and H. Homayoun, "Enhancing power, performance, and energy efficiency in chip multiprocessors exploiting inverse thermal dependence," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 26, no. 4, pp. 778–791, 2018.

[56] M. G. Moghaddam and C. Ababei, "Dynamic lifetime reliability management for chip multiprocessors," *IEEE Transactions on Multi-Scale Computing Systems*, vol. 4, no. 4, pp. 952–958, 2018.

[57] Y. Xie and W.-L. Hung, "Temperature-aware task allocation and scheduling for embedded multiprocessor systems-on-chip (MPSoC) design," *Journal of VLSI signal processing systems for signal, image and video technology*, vol. 45, no. 3, pp. 177–189, 2006.

[58] A.-M. Rahmani, M.-H. Haghbayan, A. Kanduri, A. Y. Weldezion, P. Liljeberg, J. Plosila, A. Jantsch, and H. Tenhunen, "Dynamic power management for many-core platforms

in the dark silicon era: A multi-objective control approach," in *2015 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*. IEEE, 2015, pp. 219–224.

[59] A. M. Rahmani, M.-H. Haghbayan, P. Liljeberg, A. Jantsch, and H. Tenhunen, "Multi-objective power management for CMPs in the dark silicon age," in *The Dark Side of Silicon*. Springer, 2017, pp. 191–216.

[60] E. Del Sozzo, G. C. Durelli, E. Trainiti, A. Miele, M. D. Santambrogio, and C. Bolchini, "Workload-aware power optimization strategy for asymmetric multiprocessors," in *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2016, pp. 531–534.

[61] Y. Wang, K. Ma, and X. Wang, "Temperature-constrained power control for chip multiprocessors with online model estimation," *ACM SIGARCH computer architecture news*, vol. 37, no. 3, pp. 314–324, 2009.

[62] K. Skadron, T. Abdelzaher, and M. R. Stan, "Control-theoretic techniques and thermal-RC modeling for accurate and localized dynamic thermal management," in *Proceedings Eighth International Symposium on High Performance Computer Architecture*. IEEE, 2002, pp. 17–28.

[63] Y.-K. Kwok and I. Ahmad, "Efficient scheduling of arbitrary task graphs to multiprocessors using a parallel genetic algorithm," *Journal of Parallel and Distributed Computing*, vol. 47, no. 1, pp. 58–77, 1997.

[64] A. S. Pillai, K. Singh, V. Saravanan, A. Anpalagan, I. Woungang, and L. Barolli, "A genetic algorithm-based method for optimizing the energy consumption and performance of multiprocessor systems," *Soft Computing*, vol. 22, no. 10, pp. 3271–3285, 2018.

[65] L. Miao, Y. Qi, D. Hou, Y.-h. Dai, and Y. Shi, "A multi-objective hybrid genetic algorithm for energy saving task scheduling in CMP system," in *2008 IEEE International Conference on Systems, Man and Cybernetics*. IEEE, 2008, pp. 197–201.

[66] P. R. Kumar and S. Palani, "A dynamic voltage scaling with single power supply and varying speed factor for multiprocessor system using genetic algorithm," in *International Conference on Pattern Recognition, Informatics and Medical Engineering (PRIME-2012)*. IEEE, 2012, pp. 342–346.

[67] U. Gupta, S. K. Mandal, M. Mao, C. Chakrabarti, and U. Y. Ogras, "A deep Q-learning approach for dynamic management of heterogeneous processors," *IEEE Computer Architecture Letters*, vol. 18, no. 1, pp. 14–17, 2019.

[68] M. G. Moghaddam, "Dynamic lifetime reliability and energy management for network-on-chip based chip multiprocessors," Ph.D. dissertation, Marquette University, 2018.

[69] F. M. M. ul Islam, M. Lin, L. T. Yang, and K.-K. R. Choo, "Task aware hybrid DVFS for multi-core real-time systems using machine learning," *Information Sciences*, vol. 433, pp. 315–332, 2018.

[70] Q. Zhang, M. Lin, L. T. Yang, Z. Chen, and P. Li, "Energy-efficient scheduling for real-time systems based on deep Q-learning model," *IEEE transactions on sustainable computing*, vol. 4, no. 1, pp. 132–141, 2017.

[71] M. Otoom, P. Trancoso, M. A. Alzubaidi, and H. Almasaeid, "Machine learning-based energy optimization for parallel program execution on multicore chips," *Arabian Journal for Science and Engineering*, vol. 43, no. 12, pp. 7343–7358, 2018.

[72] D. Processor, "Power and thermal management in the intel® core tm," *Intel® Centrino® Duo Mobile Technology*, vol. 10, no. 2, p. 109, 2006.

[73] J. Iyer, C. L. Hall, J. Shi, and Y. Huang, "System memory power and thermal management in platforms build on intel centrino duo technology." *Intel Technology Journal*, vol. 10, no. 2, 2006.

[74] L. Buşoniu, R. Babuška, and B. De Schutter, "Multi-agent reinforcement learning: An overview," in *Innovations in multi-agent systems and applications-1*. Springer, 2010, pp. 183–221.

[75] T. Chantem, Y. Xiang, X. S. Hu, and R. P. Dick, "Enhancing multicore reliability through wear compensation in online assignment and scheduling," in *Proceedings of the Conference on Design, Automation and Test in Europe*. EDA Consortium, 2013, pp. 1373–1378.

[76] D. Brooks and M. Martonosi, "Dynamic thermal management for high-performance microprocessors," in *Proceedings HPCA Seventh International Symposium on High-Performance Computer Architecture*. IEEE, 2001, pp. 171–182.

[77] A. K. Coskun, D. Atienza, T. S. Rosing, T. Brunschwiler, and B. Michel, "Energy-efficient variable-flow liquid cooling in 3D stacked architectures," in *Proceedings of the Conference on Design, Automation and Test in Europe (DATE)*. European Design and Automation Association, 2010, pp. 111–116.

[78] C. J. Lasance, "Thermally driven reliability issues in microelectronic systems: status-quo and challenges," *Microelectronics Reliability*, vol. 43, no. 12, pp. 1969–1974, 2003.

[79] J. Koomey, "Growth in data center electricity use 2005 to 2010," *A report by Analytical Press, completed at the request of The New York Times*, vol. 9, 2011.

[80] J. Y. Kim, H. J. Chang, Y. H. Jung, K. M. Cho, and G. Augenbroe, "Energy conservation effects of a multi-stage outdoor air enabled cooling system in a data center," *Energy and buildings*, vol. 138, pp. 257–270, 2017.

[81] A. Pahlevan, X. Qu, M. Zapater, and D. Atienza, "Integrating heuristic and machine-learning methods for efficient virtual machine allocation in data centers," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2018.

[82] A. Shehabi, S. Smith, D. Sartor, R. Brown, M. Herrlin, J. Koomey, E. Masanet, N. Horner, I. Azevedo, and W. Lintner, "United states data center energy usage report," Lawrence Berkeley National Lab.(LBNL), Berkeley, CA (United States), Tech. Rep., 2016.

[83] M. Stansberry and J. Kudritzki, "Uptime institute 2012 data center industry survey," *white paper, Uptime Institute*, 2013.

[84] Cisco. (2017) Cisco unified computing system site planning guide: Data center power and cooling. "https://www.cisco.com/c/en/us /solutions/collateral/data-center-virtualization/unified-computing/white_ paper_c11-680202.pdf".

[85] J. Srinivasan, S. V. Adve, P. Bose, and J. A. Rivers, "The case for lifetime reliability-aware microprocessors," *ACM SIGARCH Computer Architecture News*, vol. 32, no. 2, p. 276, 2004.

[86] J. Council, "Failure mechanisms and models for semiconductor devices," *JEDEC Publication JEP122-A*, 2002.

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

[88] A. K. Coskun, T. Š. Rosing, K. A. Whisnant, and K. C. Gross, "Static and dynamic temperature-aware scheduling for multiprocessor SoCs," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 16, no. 9, pp. 1127–1140, 2008.

[89] Z. Lu, W. Huang, S. Ghosh, J. Lach, M. Stan, and K. Skadron, "Analysis of temporal and spatial temperature gradients for ic reliability," *University of Virginia Technical Report CS-2004*, vol. 8, 2004.

[90] Y. Xiang, T. Chantem, R. P. Dick, X. S. Hu, and L. Shang, "System-level reliability modeling for MPSoCs," in *Proceedings of the eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*. ACM, 2010, pp. 297–306.

[91] S. D. Downing and D. Socie, "Simple rainflow counting algorithms," *International journal of fatigue*, vol. 4, no. 1, pp. 31–40, 1982.

[92] J. W. McPherson, J. McPherson, and Glaser, *Reliability physics and engineering*. Springer, 2010.

[93] Y. Joshi and P. Kumar, *Energy efficient thermal management of data centers*. Springer Science & Business Media, 2012.

[94] N. Rolander, J. Rambo, Y. Joshi, J. K. Allen, and F. Mistree, "An approach to robust design of turbulent convective systems," *Journal of Mechanical Design*, vol. 128, no. 4, pp. 844–855, 2006.

[95] A. H. Khalaj and S. K. Halgamuge, "A review on efficient thermal management of air-and liquid-cooled data centers: From chip to the cooling system," *Elsevier, Applied Energy*, vol. 205, pp. 1165–1188, 2017.

[96] D. Varma, "Air-based cooling vs. liquid-based cooling," Tech. Rep., 2020. [Online]. Available: https://www.grcooling.com/air-based-cooling-vs-liquid-based-cooling/

[97] M. Iyengar, M. David, P. Parida, V. Kamath, B. Kochuparambil, D. Graybill, M. Schultz, M. Gaynes, R. Simons, R. Schmidt *et al.*, "Server liquid cooling with chiller-less data center design to enable significant energy savings," in *SEMI-THERM, 2012 28th Annual IEEE*. IEEE, 2012, pp. 212–223.

[98] M. A. Kadhim, Y. T. Al-Anii, N. Kapur, J. L. Summers, and H. M. Thompson, "Performance of a mixed mode air handling unit for direct liquid-cooled servers," in *SEMI-THERM*. IEEE, 2017, pp. 172–178.

[99] S. Zimmermann, I. Meijer, M. K. Tiwari, S. Paredes, B. Michel, and D. Poulikakos, "Aquasar: A hot water cooled data center with direct energy reuse," *Elsevier, Energy*, vol. 43, no. 1, pp. 237–245, 2012.

[100] P. L. Leonard and A. Phillips, "The thermal bus opportunity-a quantum leap in data center cooling potential." *ASHRAE transactions*, vol. 111, no. 2, 2005.

[101] R. Hannemann, J. Marsala, and M. Pitasi, "Pumped liquid multiphase cooling," in *ASME 2004 International Mechanical Engineering Congress and Exposition*. American Society of Mechanical Engineers, 2004, pp. 469–473.

[102] H. Coles and M. Herrlin, "Immersion cooling of electronics in dod installations," CALIFORNIA UNIV BERKELEY BERKELEY United States, Tech. Rep., 2016.

[103] A. Bar-Cohen, M. Arik, and M. Ohadi, "Direct liquid cooling of high flux micro and nano electronic components," *Proceedings of the IEEE*, vol. 94, no. 8, pp. 1549–1570, 2006.

[104] N. Lamaison, J. B. Marcinichen, C. L. Ong, and J. R. Thome, "Two-phase mini-thermosyphon electronics cooling, Part 3: Transient modeling and experimental validation," in *2016 15th IEEE Intersociety Conference on Thermal and Thermomechanical Phenomena in Electronic Systems (ITherm)*. IEEE, 2016, pp. 589–598.

[105] ——, "Two-phase mini-thermosyphon electronics cooling, part 4: Application to 2u servers," in *2016 15th IEEE Intersociety Conference on Thermal and Thermomechanical Phenomena in Electronic Systems (ITherm)*. Ieee, 2016, pp. 599–609.

[106] C. L. Ong, R. L. Amalfi, J. B. Marcinichen, N. Lamaison, and J. R. Thome, "Two-phase mini-thermosyphon for cooling of datacenters: Experiments, modeling and simulations," in *ASME 2017 International Technical Conference and Exhibition on Packaging and Integration of Electronic and Photonic Microsystems collocated with the ASME 2017 Conference on Information Storage and Processing Systems*. American Society of Mechanical Engineers Digital Collection, 2017.

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

[108] J. A. Winter, D. H. Albonesi, and C. A. Shoemaker, "Scalable thread scheduling and global power management for heterogeneous many-core architectures," in *2010 19th International Conference on Parallel Architectures and Compilation Techniques (PACT)*. IEEE, 2010, pp. 29–39.

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

[110] B. Zhao and H. Aydin, "Minimizing expected energy consumption through optimal integration of DVS and DPM," in *Proceedings of the 2009 International Conference on Computer-Aided Design*. ACM, 2009, pp. 449–456.

[111] V. Devadas and H. Aydin, "On the interplay of voltage/frequency scaling and device power management for frame-based real-time embedded applications," *IEEE Transactions on Computers*, vol. 61, no. 1, pp. 31–44, 2012.

[112] A. Mutapcic, S. Boyd, S. Murali, D. Atienza, G. De Micheli, and R. Gupta, "Processor speed control with thermal constraints," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 56, no. 9, pp. 1994–2008, 2009.

[113] S. Zhang and K. S. Chatha, "Thermal aware task sequencing on embedded processors," in *Design Automation Conference*. IEEE, 2010, pp. 585–590.

[114] F. Mulas, D. Atienza, A. Acquaviva, S. Carta, L. Benini, and G. De Micheli, "Thermal balancing policy for multiprocessor stream computing platforms," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 28, no. 12, pp. 1870–1882, 2009.

[115] V. Hanumaiah and S. Vrudhula, "Temperature-aware DVFS for hard real-time applications on multicore processors," *IEEE Transactions on Computers*, vol. 61, no. 10, pp. 1484–1494, 2012.

[116] X. Zhou, J. Yang, M. Chrobak, and Y. Zhang, "Performance-aware thermal management via task scheduling," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 7, no. 1, pp. 1–31, 2010.

[117] M. Al Faruque, J. Jahn, T. Ebi, and J. Henkel, "Runtime thermal management using software agents for multi-and many-core architectures," *IEEE Design & Test of Computers*, vol. 27, no. 6, pp. 58–68, 2010.

[118] G. Liu, M. Fan, and G. Quan, "Neighbor-aware dynamic thermal management for multi-core platform," in *2012 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2012, pp. 187–192.

[119] G. Singla, G. Kaur, A. K. Unver, and U. Y. Ogras, "Predictive dynamic thermal and power management for heterogeneous mobile platforms," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2015*. IEEE, 2015, pp. 960–965.

[120] J. Choi, C.-Y. Cher, H. Franke, H. Hamann, A. Weger, and P. Bose, "Thermal-aware task scheduling at the system software level," in *Proceedings of the 2007 international symposium on Low power electronics and design*. ACM, 2007, pp. 213–218.

[121] J. Yang, X. Zhou, M. Chrobak, Y. Zhang, and L. Jin, "Dynamic thermal management through task scheduling," in *ISPASS 2008-IEEE International Symposium on Performance Analysis of Systems and software*. IEEE, 2008, pp. 191–201.

[122] I. Ukhov, M. Bao, P. Eles, and Z. Peng, "Steady-state dynamic temperature analysis and reliability optimization for embedded multiprocessor systems," in *Proceedings of the 49th Annual Design Automation Conference*. ACM, 2012, pp. 197–204.

[123] P. Mercati, A. Bartolini, F. Paterna, T. S. Rosing, and L. Benini, "Workload and user experience-aware dynamic reliability management in multicore processors," in *Design Automation Conference (DAC), 2013 50th ACM/EDAC/IEEE*. IEEE, 2013, pp. 1–6.

[124] A. K. Coskun, T. S. Rosing, and K. Whisnant, "Temperature aware task scheduling in MPSoCs," in *2007 Design, Automation & Test in Europe Conference & Exhibition*. IEEE, 2007, pp. 1–6.

[125] 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," in *Proceedings of the 51st Annual Design Automation Conference*. ACM, 2014, pp. 1–6.

[126] A. Iranfar, S. N. Shahsavani, M. Kamal, and A. Afzali-Kusha, "A heuristic machine learning-based algorithm for power and thermal management of heterogeneous MPSoCs," in *2015 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*. IEEE, 2015, pp. 291–296.

[127] M. M. S. Aly and D. Atienza Alonso, "Temperature-aware design and management for 3D multi-core architectures," *Foundations and Trends in Electronic Design Automation*, vol. 8, pp. 117–197, 2014.

[128] J. C. Salinas-Hilburg, M. Zapater, J. L. Risco-Martín, J. M. Moya, and J. L. Ayala, "Unsupervised power modeling of co-allocated workloads for energy efficiency in data centers," in *Proceedings of the 2016 Conference on Design, Automation & Test in Europe*. EDA Consortium, 2016, pp. 1345–1350.

[129] M. J. Dousti and M. Pedram, "Power-aware deployment and control of forced-convection and thermoelectric coolers," in *2014 51st ACM/EDAC/IEEE Design Automation Conference (DAC)*. IEEE, 2014, pp. 1–6.

[130] W. Zheng, K. Ma, and X. Wang, "TECfan: Coordinating thermoelectric cooler, fan, and DVFS for CMP energy optimization," in *IPDPS*. IEEE, 2016.

[131] M. Zapater, J. L. Ayala, J. M. Moya, K. Vaidyanathan, K. Gross, and A. K. Coskun, "Leakage and temperature aware server control for improving energy efficiency in data centers," in *2013 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2013, pp. 266–269.

[132] V. Hanumaiah and S. Vrudhula, "Energy-efficient operation of multicore processors by dvfs, task migration, and active cooling," *IEEE Transactions on Computers*, vol. 63, no. 2, pp. 349–360, 2012.

[133] B. Acun, E. K. Lee, Y. Park, and L. V. Kale, "Neural network-based task scheduling with preemptive fan control," in *Proceedings of the 4th International Workshop on Energy Efficient Supercomputing*. IEEE Press, 2016.

[134] C. S. Chan, Y. Jin, Y.-K. Wu, K. Gross, K. Vaidyanathan, and T. Rosing, "Fan-speed-aware scheduling of data intensive jobs," in *Proceedings of the 2012 ACM/IEEE international symposium on Low power electronics and design*, 2012, pp. 409–414.

[135] C. S. Chan, A. S. Akyürek, B. Aksanli, and T. Š. Rosing, "Optimal performance-aware cooling on enterprise servers," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 9, pp. 1689–1702, 2018.

[136] J. Kim, M. M. Sabry, D. Atienza, K. Vaidyanathan, and K. Gross, "Global fan speed control considering non-ideal temperature measurements in enterprise servers," in *2014 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2014, pp. 1–6.

[137] Z. Liu, Y. Chen, C. Bash, A. Wierman, D. Gmach, Z. Wang, M. Marwah, and C. Hyser, "Renewable and cooling aware workload management for sustainable data centers," *ACM SIGMETRICS Performance Evaluation Review*, vol. 40, no. 1, pp. 175–186, 2012.

[138] L. Wang, G. Von Laszewski, J. Dayal, X. He, A. J. Younge, and T. R. Furlani, "Towards thermal aware workload scheduling in a data center," in *2009 10th International Symposium on Pervasive Systems, Algorithms, and Networks (ISPAN)*. IEEE, 2009, pp. 116–122.

[139] A. Banerjee, T. Mukherjee, G. Varsamopoulos, and S. K. Gupta, "Cooling-aware and thermal-aware workload placement for green HPC data centers," in *Green Computing Conference*. IEEE, 2010, pp. 245–256.

[140] T. Cao, W. Huang, Y. He, and M. Kondo, "Cooling-aware job scheduling and node allocation for overprovisioned HPC systems," in *Parallel and Distributed Processing Symposium (IPDPS), 2017 IEEE International*. IEEE, 2017, pp. 728–737.

[141] M. M. Sabry, A. K. Coskun, D. Atienza, T. Š. Rosing, and T. Brunschwiler, "Energy-efficient multiobjective thermal control for liquid-cooled 3-D stacked architectures," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2011.

[142] A. Kumar, L. Shang, L.-S. Peh, and N. K. Jha, "HybDTM: a coordinated hardware-software approach for dynamic thermal management," in *Proceedings of the 43rd annual Design Automation Conference*. ACM, 2006, pp. 548–553.

[143] S. Naffziger. (2014) Amd's commitment to accelerating energy efficiency.

[144] E. Rotem, A. Naveh, A. Ananthakrishnan, E. Weissmann, and D. Rajwan, "Power-management architecture of the Intel microarchitecture code-named sandy bridge," *IEEE micro*, vol. 32, no. 2, pp. 20–27, 2012.

[145] M. Ghasemazar, H. Goudarzi, and M. Pedram, "Robust optimization of a chip multiprocessor's performance under power and thermal constraints," in *2012 IEEE 30th International Conference on Computer Design (ICCD)*. IEEE, 2012, pp. 108–114.

[146] Y. Han, I. Koren, and C. M. Krishna, "Tilts: A fast architectural-level transient thermal simulation method," *Journal of Low Power Electronics*, vol. 3, no. 1, pp. 13–21, 2007.

[147] W. Huang, S. Ghosh, S. Velusamy, K. Sankaranarayanan, K. Skadron, and M. R. Stan, "HotSpot: A compact thermal modeling methodology for early-stage VLSI design," *IEEE Transactions on very large scale integration (VLSI) systems*, vol. 14, no. 5, pp. 501–513, 2006.

[148] W. Huang, K. Sankaranarayanan, K. Skadron, R. J. Ribando, and M. R. Stan, "Accurate, pre-RTL temperature-aware design using a parameterized, geometric thermal model," *IEEE Transactions on Computers*, vol. 57, no. 9, pp. 1277–1288, 2008.

[149] K. Van Craeynest, A. Jaleel, L. Eeckhout, P. Narvaez, and J. Emer, "Scheduling heterogeneous multi-cores through performance impact estimation (pie)," in *2012 39th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2012, pp. 213–224.

[150] T. E. Carlson, W. Heirman, and L. Eeckhout, "Sniper: Exploring the level of abstraction for scalable and accurate parallel multi-core simulation," in *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, 2011, pp. 1–12.

[151] K. Van Craeynest, S. Akram, W. Heirman, A. Jaleel, and L. Eeckhout, "Fairness-aware scheduling on single-ISA heterogeneous multi-cores," in *Proceedings of the 22nd international conference on Parallel architectures and compilation techniques*. IEEE, 2013, pp. 177–187.

[152] G. G. Faust, R. Zhang, K. Skadron, M. R. Stan, and B. H. Meyer, "ArchFP: Rapid prototyping of pre-RTL floorplans," in *2012 IEEE/IFIP 20th International Conference on VLSI and System-on-Chip (VLSI-SoC)*. IEEE, 2012, pp. 183–188.

[153] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The PARSEC benchmark suite: Characterization and architectural implications," in *Proceedings of the 17th international conference on Parallel architectures and compilation techniques*, 2008, pp. 72–81.

[154] S. Sharifi, A. K. Coskun, and T. S. Rosing, "Hybrid dynamic energy and thermal management in heterogeneous embedded multiprocessor SoCs," in *2010 15th Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2010, pp. 873–878.

[155] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, "McPAT: an integrated power, area, and timing modeling framework for multicore and manycore architectures," in *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, 2009, pp. 469–480.

[156] S. G. Johnson. (2014) The NLopt nonlinear-optimization package.

[157] K. Skadron, M. R. Stan, K. Sankaranarayanan, W. Huang, S. Velusamy, and D. Tarjan, "Temperature-aware microarchitecture: Modeling and implementation," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 1, no. 1, pp. 94–125, 2004.

[158] A. Iranfar, F. Terraneo, W. A. Simon, L. Dragic, I. Piljic, M. Zapater, W. Fornaciari, M. Kovac, and D. Atienza Alonso, "Thermal characterization of next-generation workloads on heterogeneous MPSoCs," in *International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation (SAMOS)*, 2017, pp. 1–6.

[159] A. Sridhar, A. Vincenzi, M. Ruggiero, T. Brunschwiler, and D. Atienza, "3D-ICE: Fast compact transient thermal modeling for 3D ICs with inter-tier liquid cooling," in *Proceedings of the International Conference on Computer-Aided Design*. IEEE Press, 2010, pp. 463–470.

[160] J. Treibig, G. Hager, and G. Wellein, "Likwid: A lightweight performance-oriented tool suite for x86 multicore environments," in *2010 39th International Conference on Parallel Processing Workshops*. IEEE, 2010, pp. 207–216.

[161] T. Gruber, J. Eitzinger, G. Hager, and G. Wellein. (2019) LIKWID 5: Lightweight performance tools.

[162] C. Bienia, "Benchmarking modern multiprocessors," Ph.D. dissertation, Princeton University, 2011.

[163] C. Delimitrou and C. Kozyrakis, "Optimizing resource provisioning in shared cloud systems," *Tech. Rep.*, 2014.

[164] Intel. (2016) Intel xeon processor E5 v4 product family datasheet, volume one: Electrical. https://www.intel.com/content/dam/www/public/us/en/documents/datasheets/xeon-e5-v4-datasheet-vol-1.pdf.

[165] R. Cochran, C. Hankendi, A. K. Coskun, and S. Reda, "Pack & cap: adaptive DVFS and thread packing under power caps," in *MICRO*. IEEE, 2011, pp. 175–185.

[166] D. Sopic, A. Aminifar, and D. Atienza, "e-glass: A wearable system for real-time detection of epileptic seizures," in *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2018, pp. 1–5.

[167] Q. Zhang, L. T. Yang, Z. Yan, Z. Chen, and P. Li, "An efficient deep learning model to predict cloud workload for industry informatics," *IEEE transactions on industrial informatics*, vol. 14, no. 7, pp. 3170–3178, 2018.

[168] M. Amiri, L. Mohammad-Khanli, and R. Mirandola, "A sequential pattern mining model for application workload prediction in cloud environment," *Journal of Network and Computer Applications*, vol. 105, pp. 21–62, 2018.

[169] C. Liu, C. Liu, Y. Shang, S. Chen, B. Cheng, and J. Chen, "An adaptive prediction approach based on workload pattern discrimination in the cloud," *Journal of Network and Computer Applications*, vol. 80, pp. 35–44, 2017.

[170] Y. Hu, B. Deng, F. Peng, and D. Wang, "Workload prediction for cloud computing elasticity mechanism," in *2016 IEEE International Conference on Cloud Computing and Big Data Analysis (ICCCBDA)*. IEEE, 2016, pp. 244–249.

[171] D. Sopic, A. Aminifar, A. Aminifar, and D. Atienza, "Real-time classification technique for early detection and prevention of myocardial infarction on wearable devices," in *2017 IEEE Biomedical Circuits and Systems Conference (BioCAS)*. IEEE, 2017, pp. 1–4.

[172] J. Flich, G. Agosta, P. Ampletzer, D. A. Alonso, C. Brandolese, E. Cappe, A. Cilardo, L. Dragić, A. Dray, A. Duspara *et al.*, "MANGO: Exploring manycore architectures for next-generation HPC systems," in *DSD*. IEEE, 2017.

[173] I. Intel, "IA-32 architectures software developer's manual," *Volume 3A: System Programming Guide, Part*, vol. 1, no. 64, p. 64, 64.

[174] Cisco Systems, Inc. (2016) Cisco visual networking index: Forecast and methodology 2015-2020. cisco whitepaper.

[175] S. ULC, "Sandvine global internet phenomena report-1h2012," Technical report, Tech. Rep., 2012.

[176] J. V. Team, "Advanced video coding for generic audiovisual services," *ITU-T Rec. H*, vol. 264, pp. 14 496–10, 2003.

[177] F. Bossen, B. Bross, K. Suhring, and D. Flynn, "HEVC complexity and implementation analysis," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 22, no. 12, pp. 1685–1696, 2012.

[178] G. J. Sullivan, J.-R. Ohm, W.-J. Han, and T. Wiegand, "Overview of the high efficiency video coding (HEVC) standard," *IEEE Transactions on circuits and systems for video technology*, vol. 22, no. 12, pp. 1649–1668, 2012.

[179] P. Bordes, P. Andrivon, F. Hiron, P. Salmon, and R. Boitard. (2016) Joint Collaborative Team on Video Coding (JCT-VC) of ITU-T SG 16 WP 3 and ISO/IEC JTC 1/SC 29/WG 11. [Online]. Available: https://{HEVC}.hhi.fraunhofer.de

[180] ——. (2018) Kvazaar HEVC encoder. [Online]. Available: https://github.com/ultravideo/kvazaar

[181] F. Bossen and H. Common, "test conditions and software reference configurations," *JCT-VC Doc*, 2013. [Online]. Available: https://hevc.hhi.fraunhofer.de/

[182] F. De Simone, L. Goldmann, J.-S. Lee, and T. Ebrahimi, "Performance analysis of VP8 image and video compression based on subjective evaluations," in *Applications of Digital Image Processing XXXIV*, vol. 8135. International Society for Optics and Photonics, 2011, p. 81350M.

[183] A. S. Motra, A. Gupta, M. Shukla, P. Bansal *et al.*, "Fast intra mode decision for HEVC video encoder," in *Software, Telecommunications and Computer Networks (SoftCOM), 2012 20th International Conference on*. IEEE, 2012, pp. 1–5.

[184] T. K. Tan, R. Weerakkody, M. Mrak, N. Ramzan, V. Baroncini, J.-R. Ohm, and G. J. Sullivan, "Video quality evaluation methodology and verification testing of HEVC compression performance," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 26, no. 1, pp. 76–90, 2016.

[185] J. Levon and P. Elie. (2004) Oprofile: A system profiler for linux. [Online]. Available: http://oprofile.sourceforge.net/download/

[186] S. Sankaraiah, L. H. Shuan, C. Eswaran, and J. Abdullah, "Performance optimization of video coding process on multi-core platform using GOP level parallelism," *International Journal of Parallel Programming*, vol. 42, no. 6, pp. 931–947, 2014.

[187] F. Henry and S. Pateux, "Wavefront parallel processing," *Joint Collaborative Team on Video Coding (JCT-VC), Document JCTVC-E196, Geneva*, 2011.

[188] M. Shafique, M. U. K. Khan, and J. Henkel, "Power efficient and workload balanced tiling for parallelized high efficiency video coding," in *2014 IEEE International Conference on Image Processing (ICIP)*. IEEE, 2014, pp. 1253–1257.

[189] M. U. K. Khan, M. Shafique, and J. Henkel, "Power-efficient workload balancing for video applications," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 24, no. 6, pp. 2089–2102, 2015.

[190] R. N. Calheiros, E. Masoumi, R. Ranjan, and R. Buyya, "Workload prediction using ARIMA model and its impact on cloud applications' qos," *IEEE Transactions on Cloud Computing*, vol. 3, no. 4, pp. 449–458, 2015.

[191] D. Yi, J. Su, C. Liu, and W.-H. Chen, "New driver workload prediction using clustering-aided approaches," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 49, no. 1, pp. 64–70, 2019.

[192] Y. Tan, P. Malani, Q. Qiu *et al.*, "Workload prediction and dynamic voltage scaling for MPEG decoding," in *Asia and South Pacific Conference on Design Automation, 2006.* IEEE, 2006, pp. 6–pp.

[193] G. Wu, J. L. Greathouse, A. Lyashevsky, N. Jayasena, and D. Chiou, "GPGPU performance and power estimation using machine learning," in *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA).* IEEE, 2015, pp. 564–576.

[194] R. Jayaseelan and T. Mitra, "Dynamic thermal management via architectural adaptation," in *2009 46th ACM/IEEE Design Automation Conference.* IEEE, 2009, pp. 484–489.

[195] 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, 2014.

[196] M. F. Reza, T. T. Le, B. De, M. Bayoumi, and D. Zhao, "Neuro-NoC: Energy optimization in heterogeneous many-core NoC using neural networks in dark silicon era," in *2018 IEEE International Symposium on Circuits and Systems (ISCAS).* IEEE, 2018, pp. 1–5.

[197] Inchoon Yeo, C. C. Liu, and E. J. Kim, "Predictive dynamic thermal management for multicore systems," in *2008 45th ACM/IEEE Design Automation Conference*, 2008, pp. 734–739.

[198] D. Li, B. R. de Supinski, M. Schulz, D. S. Nikolopoulos, and K. W. Cameron, "Strategies for energy-efficient resource management of hybrid programming models," *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, no. 1, pp. 144–157, 2013.

[199] A. Das, A. Kumar, B. Veeravalli, R. Shafik, G. Merrett, and B. Al-Hashimi, "Workload uncertainty characterization and adaptive frequency scaling for energy minimization of embedded systems," in *2015 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2015, pp. 43–48.

[200] M. F. Akay and I. Abasıkeleş, "Predicting the performance measures of an optical distributed shared memory multiprocessor by using support vector regression," *Expert Systems with Applications*, vol. 37, no. 9, pp. 6293–6301, 2010.

[201] S. Sinaei, A. D. Pimentel, and O. Fatemi, "Run-time resource allocation for embedded multiprocessor system-on-chip using tree-based design space exploration," in *2017 12th International Conference on Design & Technology of Integrated Systems In Nanoscale Era (DTIS)*. IEEE, 2017, pp. 1–6.

[202] S. Browne, J. Dongarra, N. Garner, K. London, and P. Mucci, "A scalable cross-platform infrastructure for application performance tuning using hardware counters," in *SC'00: Proceedings of the 2000 ACM/IEEE Conference on Supercomputing*. IEEE, 2000.

[203] Y. Lv, B. Sun, Q. Luo, J. Wang, Z. Yu, and X. Qian, "CounterMiner: Mining big performance data from hardware counters," in *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2018, pp. 613–626.

[204] C. V. Li, V. Petrucci, and D. Mossé, "Exploring machine learning for thread characterization on heterogeneous multiprocessors," *ACM SIGOPS Operating Systems Review*, vol. 51, no. 1, pp. 113–123, 2017.

[205] J. Nomani and J. Szefer, "Predicting program phases and defending against side-channel attacks using hardware performance counters," in *Proceedings of the Fourth Workshop on Hardware and Architectural Support for Security and Privacy*. ACM, 2015, p. 9.

[206] L. Sethumadhavan, A. Tang, and S. Stolfo. (2018) Unsupervised anomaly-based malware detection using hardware features. US Patent App. 15/982,229.

[207] X. Zheng, L. K. John, and A. Gerstlauer, "Accurate phase-level cross-platform power and performance estimation," in *Proceedings of the 53rd Annual Design Automation Conference*, 2016, pp. 1–6.

[208] A. Borghesi, A. Bartolini, M. Lombardi, M. Milano, and L. Benini, "Predictive modeling for job power consumption in HPC systems," in *HiPC*. Springer, 2016.

[209] K. Singh, M. Bhadauria, and S. A. McKee, "Real time power estimation and thread scheduling via performance counters," *ACM SIGARCH Computer Architecture News*, vol. 37, no. 2, pp. 46–55, 2009.

[210] M. Curtis-Maury, J. Dzierwa, C. D. Antonopoulos, and D. S. Nikolopoulos, "Online power-performance adaptation of multithreaded programs using hardware event-based prediction," in *Proceedings of the 20th annual international conference on Supercomputing*, 2006, pp. 157–166.

[211] V. Podolskiy, A. Jindal, M. Gerndt, and Y. Oleynik, "Forecasting models for self-adaptive cloud applications: A comparative study," in *2018 IEEE 12th International Conference on Self-Adaptive and Self-Organizing Systems (SASO)*. IEEE, 2018, pp. 40–49.

[212] A. C. De Melo, "The new linux'perf'tools," in *Slides from Linux Kongress*, vol. 18, 2010.

[213] J. Benesty, J. Chen, Y. Huang, and I. Cohen, "Pearson correlation coefficient," in *Noise reduction in speech processing*. Springer, 2009, pp. 1–4.

[214] D. Arthur and S. Vassilvitskii, "k-means++: The advantages of careful seeding," in *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*. Society for Industrial and Applied Mathematics, 2007, pp. 1027–1035.

[215] P. J. Rousseeuw, "Silhouettes: a graphical aid to the interpretation and validation of cluster analysis," *Journal of computational and applied mathematics*, vol. 20, pp. 53–65, 1987.

[216] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[217] R. G. Gallager, *Stochastic processes: theory for applications*. Cambridge University Press, 2013.

[218] D. Hanrahan, "Fan-speed control techniques in PCs," *Analog Dialogue*, 2000.

[219] Z. Wang, C. Bash, N. Tolia, M. Marwah, X. Zhu, and P. Ranganathan, "Optimal fan speed control for thermal management of servers," in *ASME 2009 InterPACK Conference*. American Society of Mechanical Engineers, 2009.

[220] B. Bross, "High efficiency video coding (HEVC) text specification draft 9 (sodis)," in *11th JCT-VC meeting*, Oct 2012.

[221] (2017) DMR youtube report. [Online]. Available: http://expandedramblings.com/index.php/youtube-statistics/#

[222] A. Vetro, C. Christopoulos, and H. Sun, "Video transcoding architectures and techniques: an overview," *IEEE Signal processing magazine*, vol. 20, no. 2, pp. 18–29, 2003.

[223] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[224] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[225] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[226] M. Z. Alom, T. M. Taha, C. Yakopcic, S. Westberg, P. Sidike, M. S. Nasrin, B. C. Van Esesn, A. A. S. Awwal, and V. K. Asari, "The history began from alexnet: A comprehensive survey on deep learning approaches," *arXiv preprint arXiv:1803.01164*, 2018.

[227] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[228] L. Bottou, "Large-scale machine learning with stochastic gradient descent," in *Proceedings of COMPSTAT'2010*. Springer, 2010, pp. 177–186.

[229] S. Qamar, H. Jin, R. Zheng, P. Ahmad, and M. Usama, "A variant form of 3D-UNet for infant brain segmentation," *Future Generation Computer Systems*, 2019.

[230] B. H. Menze, A. Jakab, S. Bauer, J. Kalpathy-Cramer, K. Farahani, J. Kirby, Y. Burren, N. Porz, J. Slotboom, R. Wiest *et al.*, "The multimodal brain tumor image segmentation benchmark (BRATS)," *IEEE transactions on medical imaging*, vol. 34, no. 10, pp. 1993–2024, 2014.

[231] S. Bakas, H. Akbari, A. Sotiras, M. Bilello, M. Rozycki, J. S. Kirby, J. B. Freymann, K. Farahani, and C. Davatzikos, "Advancing the cancer genome atlas glioma MRI collections with expert segmentation labels and radiomic features," *Scientific data*, vol. 4, p. 170117, 2017.

[232] S. Bakas, M. Reyes, A. Jakab, S. Bauer, M. Rempfler, A. Crimi, R. T. Shinohara, C. Berger, S. M. Ha, M. Rozycki *et al.*, "Identifying the best machine learning algorithms for brain tumor segmentation, progression assessment, and overall survival prediction in the BRATS challenge," *arXiv preprint arXiv:1811.02629*, 2018.

[233] P. Tschandl, C. Rosendahl, and H. Kittler, "The HAM10000 dataset, a large collection of multi-source dermatoscopic images of common pigmented skin lesions," *Scientific data*, vol. 5, p. 180161, 2018.

[234] N. Codella, V. Rotemberg, P. Tschandl, M. E. Celebi, S. Dusza, D. Gutman, B. Helba, A. Kalloo, K. Liopyris, M. Marchetti *et al.*, "Skin lesion analysis toward melanoma detection 2018: A challenge hosted by the international skin imaging collaboration (ISIC)," *arXiv preprint arXiv:1902.03368*, 2019.

[235] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press Cambridge, 1998, vol. 1.

[236] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[237] E. Even-Dar and Y. Mansour, "Learning rates for q-learning," *Journal of Machine Learning Research*, vol. 5, no. Dec, pp. 1–25, 2003.

[238] Q. Huang, K. Zhou, S. You, and U. Neumann, "Learning to prune filters in convolutional neural networks," in *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*. IEEE, 2018, pp. 709–718.

[239] B. Baker, O. Gupta, N. Naik, and R. Raskar, "Designing neural network architectures using reinforcement learning," *arXiv preprint arXiv:1611.02167*, 2016.

[240] C.-H. Hsu, S.-H. Chang, J.-H. Liang, H.-P. Chou, C.-H. Liu, S.-C. Chang, J.-Y. Pan, Y.-T. Chen, W. Wei, and D.-C. Juan, "Monas: Multi-objective neural architecture search using reinforcement learning," *arXiv preprint arXiv:1806.10332*, 2018.

[241] U. A. Khan and B. Rinner, "Online learning of timeout policies for dynamic power management," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 13, no. 4, p. 96, 2014.

[242] J.-H. Hu, W.-H. Peng, and C.-H. Chung, "Reinforcement learning for HEVC/H. 265 intra-frame rate control," in *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2018, pp. 1–5.

[243] Y.-G. Chen, W.-Y. Wen, T. Wang, Y. Shi, and S.-C. Chang, "Q-learning based dynamic voltage scaling for designs with graceful degradation," in *Proceedings of the 2015 Symposium on International Symposium on Physical Design*, 2015, pp. 41–48.

[244] H. Shen, Y. Tan, J. Lu, Q. Wu, and Q. Qiu, "Achieving autonomous power management using reinforcement learning," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 18, no. 2, pp. 1–32, 2013.

[245] Z. Chen and D. Marculescu, "Distributed reinforcement learning for power limited many-core system performance optimization," in *2015 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2015, pp. 1521–1526.

[246] D. Biswas, V. Balagopal, R. Shafik, B. M. Al-Hashimi, and G. V. Merrett, "Machine learning for run-time energy optimization in many-core systems," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*. IEEE, 2017, pp. 1588–1592.

[247] Z. Wang, Z. Tian, J. Xu, R. K. Maeda, H. Li, P. Yang, Z. Wang, L. H. Duong, Z. Wang, and X. Chen, "Modular reinforcement learning for self-adaptive energy efficiency optimization in multicore system," in *2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2017, pp. 684–689.

[248] F. Terraneo, A. Leva, and W. Fornaciari, "Event-based thermal control for high power density microprocessors," in *Harnessing Performance Variability in Embedded and High-performance Many/Multi-core Platforms*. Springer, 2019.

[249] F. Paterna, A. Acquaviva, and L. Benini, "Aging-aware energy-efficient workload allocation for mobile multimedia platforms," *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, no. 8, pp. 1489–1499, 2012.

[250] X. Nan, Y. He, and L. Guan, "Optimization of workload scheduling for multimedia cloud computing," in *2013 IEEE International Symposium on Circuits and Systems (ISCAS2013)*. IEEE, 2013, pp. 2872–2875.

[251] C. Li, L. Zhu, Y. Liu, and Y. Luo, "Resource scheduling approach for multimedia cloud content management," *The Journal of Supercomputing*, vol. 73, no. 12, pp. 5150–5172, 2017.

[252] S. Sahoo, I. Parida, S. K. Mishra, B. Sahoo, and A. K. Turuk, "Resource allocation for video transcoding in the multimedia cloud," in *Recent Findings in Intelligent Computing Techniques*. Springer, 2019, pp. 525–532.

[253] H. R. Mendis, N. C. Audsley, and L. S. Indrusiak, "Dynamic and static task allocation for hard real-time video stream decoding on NoCs," *Leibniz Transactions on Embedded Systems*, vol. 4, no. 2, pp. 01–1, 2017.

[254] D. Lee, J. Lee, and M. Song, "Video quality adaptation for limiting transcoding energy consumption in video servers," *IEEE Access*, vol. 7, pp. 126 253–126 264, 2019.

[255] M. Song, Y. Lee, and J. Park, "Scheduling a video transcoding server to save energy," *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, vol. 11, no. 2s, pp. 1–23, 2015.

[256] C. Jiang and S. Nooshabadi, "Parallel multiview video coding exploiting group of pictures level parallelism," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 8, pp. 2316–2328, 2016.

[257] X. Li, M. A. Salehi, M. Bayoumi, N. Tzeng, and R. Buyya, "Cost-efficient and robust on-demand video transcoding using heterogeneous cloud services," *IEEE Transactions on Parallel and Distributed Systems*, vol. PP, no. 99, pp. 1–1, 2017.

[258] I. Rec, "H. 264 advanced video coding for generic audiovisual services," *ITU-T Rec. H. 264-ISO/IEC 14496-10 AVC*, 2005.

[259] H. Kim and Y. Altunhasak, "Low-complexity macroblock mode selection for H. 264-AVC encoders," in *2004 International Conference on Image Processing, 2004. ICIP'04.*, vol. 2. IEEE, 2004, pp. 765–768.

[260] D. S. Turaga, M. van der Schaar, and B. Pesquet-Popescu, "Complexity scalable motion compensated wavelet video encoding," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 15, no. 8, pp. 982–993, 2005.

[261] M. Viitanen, A. Koivula, A. Lemmetti, A. Ylä-Outinen, J. Vanne, and T. D. Hämäläinen, "Kvazaar: Open-source HEVC/H. 265 encoder," in *Proceedings of the 2016 ACM on Multimedia Conference*. ACM, 2016, pp. 1179–1182.

[262] E. Kalali, Y. Adibelli, and I. Hamzaoglu, "A high performance and low energy intra prediction hardware for high efficiency video coding," in *Field Programmable Logic and Applications (FPL), 2012 22nd International Conference on*. IEEE, 2012, pp. 719–722.

[263] E. Raffin *et al.*, "Low power HEVC software decoder for mobile devices," *Journal of Real-Time Image Processing*, pp. 1–13, 2015.

[264] E. Nogues, S. Holmbacka, M. Pelcat, D. Menard, and J. Lilius, "Power-aware HEVC decoding with tunable image quality," in *2014 IEEE Workshop on Signal Processing Systems (SiPS)*. IEEE, 2014, pp. 1–6.

[265] E. Raffin, E. Nogues, W. Hamidouche, S. Tomperi, M. Pelcat, and D. Menard, "Low power HEVC software decoder for mobile devices," *Journal of Real-Time Image Processing*, vol. 12, no. 2, pp. 495–507, 2016.

[266] M. Abeydeera, M. Karunaratne, G. Karunaratne, K. De Silva, and A. Pasqual, "4K real-time HEVC decoder on an FPGA," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 26, no. 1, pp. 236–249, 2016.

[267] Y. He, M. Kunstner, S. Gudumasu, E.-S. Ryu, Y. Ye, and X. Xiu, "Power aware HEVC streaming for mobile," in *Visual Communications and Image Processing (VCIP)*. IEEE, 2013, pp. 1–5.

[268] G. Correa, P. Assuncao, L. Agostini, and L. A. da Silva Cruz, "Complexity scalability for real-time HEVC encoders," *Journal of Real-Time Image Processing*, vol. 12, no. 1, pp. 107–122, 2016.

[269] G. Tian and S. Goto, "Content adaptive prediction unit size decision algorithm for HEVC intra coding," in *Picture Coding Symp.* IEEE, 2012, pp. 405–408.

[270] M. U. K. Khan, M. Shafique, and J. Henkel, "Software architecture of high efficiency video coding for many-core systems with power-efficient workload balancing," in *2014 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2014, pp. 1–6.

[271] G. Corrêa, P. Assuncao, L. Agostini, and L. A. da Silva Cruz, "Complexity control of high efficiency video encoders for power-constrained devices," *IEEE Transactions on Consumer Electronics*, vol. 57, no. 4, pp. 1866–1874, 2011.

[272] G. Correa, P. Assuncao, L. A. da Silva Cruz, and L. Agostini, "Dynamic tree-depth adjustment for low power HEVC encoders," in *2012 19th IEEE International Conference on Electronics, Circuits, and Systems (ICECS 2012)*. IEEE, 2012, pp. 564–567.

[273] D. Zhou, L. Guo, J. Zhou, and S. Goto, "Reducing power consumption of HEVC codec with lossless reference frame recompression," in *Image Processing (ICIP), 2014 IEEE International Conference on*. IEEE, 2014, pp. 2120–2124.

[274] Z. Ma and A. Segall, "Frame buffer compression for low-power video coding," in *IEEE International Conference on Image Processing*. IEEE, 2011, pp. 757–760.

[275] D. Palomino, M. Shafique, H. Amrouch, A. Susin, and J. Henkel, "hevcDTM: Application-driven dynamic thermal management for high efficiency video coding," in *2014 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2014, pp. 1–4.

[276] D. Palomino, M. Shafique, A. Susin, and J. Henkel, "TONE: Adaptive temperature optimization for the next generation video encoders," in *Proceedings of the 2014 international symposium on Low power electronics and design*. ACM, 2014, pp. 33–38.

[277] ——, "Thermal optimization using adaptive approximate computing for video coding," in *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2016, pp. 1207–1212.

[278] M. Shafique and J. Henkel, "Low power design of the next-generation high efficiency video coding," in *Design Automation Conference (ASP-DAC), 2014 19th Asia and South Pacific*. IEEE, 2014, pp. 274–281.

[279] M. Shafique, B. Molkenthin, and J. Henkel, "An HVS-based adaptive computational complexity reduction scheme for H. 264/AVC video encoder using prognostic early mode exclusion," in *Proceedings of the Conference on Design, Automation and Test in Europe*. European Design and Automation Association, 2010, pp. 1713–1718.

[280] Y. Wang, J.-G. Kim, S.-F. Chang, and H.-M. Kim, "Utility-based video adaptation for universal multimedia access (UMA) and content-based utility function prediction for real-time video transcoding," *IEEE Transactions on Multimedia*, vol. 9, no. 2, pp. 213–220, 2007.

[281] S. Atapattu, N. Liyanage, N. Menuka, I. Perera, and A. Pasqual, "Real time all intra HEVC HD encoder on FPGA," in *IEEE 27th International Conference on Application-specific Systems, Architectures and Processors*, 2016, pp. 191–195.

[282] K. Miyazawa, H. Sakate, S.-i. Sekiguchi, N. Motoyama, Y. Sugito, K. Iguchi, A. Ichigaya, and S.-i. Sakaida, "Real-time hardware implementation of HEVC video encoder for 1080p hd video," in *Picture Coding Symposium (PCS), 2013*. IEEE, 2013, pp. 225–228.

[283] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *Journal of machine learning research*, vol. 13, no. Feb, pp. 281–305, 2012.

[284] K. Kandasamy, W. Neiswanger, J. Schneider, B. Poczos, and E. P. Xing, "Neural architecture search with bayesian optimisation and optimal transport," in *Advances in Neural Information Processing Systems*, 2018, pp. 2016–2025.

[285] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, "Regularized evolution for image classifier architecture search," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 33, 2019, pp. 4780–4789.

[286] H. Jin, Q. Song, and X. Hu, "Auto-keras: An efficient neural architecture search system," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2019, pp. 1946–1956.

[287] T. Elsken, J. H. Metzen, and F. Hutter, "Neural architecture search: A survey," *arXiv preprint arXiv:1808.05377*, 2018.

[288] G. F. Miller, P. M. Todd, and S. U. Hegde, "Designing neural networks using genetic algorithms." in *ICGA*, vol. 89, 1989, pp. 379–384.

[289] M. Suganuma, S. Shirakawa, and T. Nagao, "A genetic programming approach to designing convolutional neural network architectures," in *Proceedings of the Genetic and Evolutionary Computation Conference*, 2017, pp. 497–504.

[290] R. S. Olson and J. H. Moore, "TPOT: A tree-based pipeline optimization tool for automating machine learning," in *Automated Machine Learning*. Springer, 2019, pp. 151–160.

[291] H. Cai, T. Chen, W. Zhang, Y. Yu, and J. Wang, "Efficient architecture search by network transformation," in *Thirty-Second AAAI conference on artificial intelligence*, 2018.

[292] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," *arXiv preprint arXiv:1611.01578*, 2016.

[293] Z. Zhong, J. Yan, W. Wu, J. Shao, and C.-L. Liu, "Practical block-wise neural network architecture generation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 2423–2432.

[294] C. Liu, B. Zoph, M. Neumann, J. Shlens, W. Hua, L.-J. Li, L. Fei-Fei, A. Yuille, J. Huang, and K. Murphy, "Progressive neural architecture search," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 19–34.

[295] H. Pham, M. Y. Guan, B. Zoph, Q. V. Le, and J. Dean, "Efficient neural architecture search via parameter sharing," *arXiv preprint arXiv:1802.03268*, 2018.

[296] F. M. Carlucci, P. Esperanca, R. Tutunov, M. Singh, V. Gabillon, A. Yang, H. Xu, Z. Chen, and J. Wang, "Manas: multi-agent neural architecture search," *arXiv preprint arXiv:1909.01051*, 2019.

[297] H. Liu, K. Simonyan, and Y. Yang, "Darts: Differentiable architecture search," *arXiv preprint arXiv:1806.09055*, 2018.

[298] S. Xie, H. Zheng, C. Liu, and L. Lin, "Snas: stochastic neural architecture search," *arXiv preprint arXiv:1812.09926*, 2018.

[299] F. Terraneo, A. Leva, and W. Fornaciari, "An Open-Hardware Platform for MPSoC Thermal Modeling," in *International Conference on Embedded Computer Systems*. Springer, 2019.

[300] H. Amrouch and J. Henkel, "Lucid infrared thermography of thermally-constrained processors," in *2015 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*. IEEE, 2015, pp. 347–352.

[301] Y. Lu, W. Cheng, L. Huang, X. Zeng, and Y. Fan, "A flexible HEVC intra mode decision hardware for 8kx4k real time encoder," in *2015 IEEE 11th International Conference on ASIC (ASICON)*. IEEE, 2015, pp. 1–4.

[302] P. Sjövall, J. Virtanen, J. Vanne, and T. D. Hämäläinen, "High-level synthesis design flow for HEVC intra encoder on SoC-FPGA," in *DSD, 2015 Euromicro Conference on*. IEEE, 2015, pp. 49–56.

[303] J. Brandenburg and B. Stabernack, "Simulation-based HW/SW co-exploration of the concurrent execution of HEVC intra encoding algorithms for heterogeneous multi-core architectures," *Journal of Systems Architecture*, vol. 77, pp. 26–42, 2017.

[304] N. Nethercote and J. Seward, "Valgrind: A program supervision framework," *Electronic notes in theoretical computer science*, vol. 89, no. 2, pp. 44–66, 2003.

[305] Intel. Intel video transcode solutions: Simple, fast, efficient webinar. [Online]. Available: "https://software.intel.com/en-us/videos/intel-video-transcode-solutions-simple-fast-efficient-webinar"

[306] V. K. Adhikari, Y. Guo, F. Hao, V. Hilt, Z.-L. Zhang, M. Varvello, and M. Steiner, "Measurement study of Netflix, Hulu, and a tale of three CDNs," *IEEE/ACM Transactions on Networking (TON)*, vol. 23, no. 6, pp. 1984–1997, 2015.

[307] P. Vamplew, J. Yearwood, R. Dazeley, and A. Berry, "On the limitations of scalarisation for multi-objective reinforcement learning of pareto fronts," in *Australasian Joint Conference on Artificial Intelligence*. Springer, 2008, pp. 372–378.

[308] K. Van Moffaert, M. M. Drugan, and A. Nowé, "Hypervolume-based multi-objective reinforcement learning," in *International Conference on Evolutionary Multi-Criterion Optimization*. Springer, 2013, pp. 352–366.

[309] K. Van Moffaert and A. Nowé, "Multi-objective reinforcement learning using sets of pareto dominating policies," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 3483–3512, 2014.

[310] S. T. Welstead, *Fractal and wavelet image compression techniques*. SPIE Optical Engineering Press, Bellingham, WA, 1999.

[311] T. Ebi, D. Kramer, W. Karl, and J. Henkel, "Economic learning for thermal-aware power budgeting in many-core architectures," in *Proceedings of the seventh IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*. ACM, 2011, pp. 189–196.

[312] T. Ebi, A. Faruque, M. Abdullah, and J. Henkel, "TAPE: thermal-aware agent-based power economy for multi/many-core architectures," in *Proceedings of the 2009 International Conference on Computer-Aided Design*. ACM, 2009, pp. 302–309.

[313] D. Belson. (2017) Akamai's state of the internet. [Online]. Available: https://www.akamai.com/us/en/multimedia/documents/state-of-the-internet/q4-2017-state-of-the-internet-security-report.pdf

[314] M. Grellert, M. Shafique, M. U. K. Khan, L. Agostini, J. C. Mattos, and J. Henkel, "An adaptive workload management scheme for HEVC encoding," in *2013 20th IEEE International Conference on Image Processing (ICIP)*. IEEE, 2013, pp. 1850–1854.

[315] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1–9.

[316] P. F. Christ, M. E. A. Elshaer, F. Ettlinger, S. Tatavarty, M. Bickel, P. Bilic, M. Rempfler, M. Armbruster, F. Hofmann, M. D'Anastasi *et al.*, "Automatic liver and lesion segmentation in ct using cascaded fully convolutional neural networks and 3d conditional random fields," in *International Conference on Medical Image Computing and Computer-Assisted Intervention*. Springer, 2016, pp. 415–423.

[317] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*. IEEE, 2009, pp. 248–255.

[318] A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images," 2009.

[319] G. Bjontegaard, "Calculation of average PSNR differences between RD-Curves," in *Proceedings of the ITU-T Video Coding Experts Group (VCEG) Thirteenth Meeting*, 2001.

[320] A. Iranfar, M. Kamal, A. Afzali-Kusha, M. Pedram, and D. Atienza, "TheSPoT: Thermal stress-aware power and temperature management for multiprocessor systems-on-chip," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2018.

[321] A. Iranfar, A. Pahlevan, M. Zapater, and D. Atienza, "Enhancing Two-Phase Cooling Efficiency through Thermal-Aware Workload Mapping for Power-Hungry Servers," in *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2019, pp. 66–71.

[322] A. Iranfar, W. S. De Souza, M. Zapater, K. Olcoz, S. X. de Souza, and D. Atienza, "A machine learning-based framework for throughput estimation of time-varying applications in multi-core servers," in *2019 IFIP/IEEE 27th International Conference on Very Large Scale Integration (VLSI-SoC)*. IEEE, 2019, pp. 211–216.

[323] A. Iranfar, A. Pahlevan, M. Zapater, M. Žagar, M. Kovač, and D. Atienza, "Online efficient bio-medical video transcoding on MPSoCs through content-aware workload allocation," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2018*. IEEE, 2018, pp. 949–954.

[324] W. Silva-de Souza, A. Iranfar, A. Bráulio, M. Zapater, S. Xavier-de Souza, K. Olcoz, and D. Atienza, "Containergy—a container-based energy and performance profiling tool for next generation workloads," *Energies*, vol. 13, no. 9, p. 2162, 2020.

[325] A. Iranfar, F. Terraneo, G. Csordas, M. Zapater Sancho, W. Fornaciari, and D. Atienza Alonso, "Dynamic thermal management with proactive fan speed control through reinforcement learning," in *[Proceedings Design, Automation and Test in Europe Conference and Exhibition]*. IEEE, 2020.

[326] A. Iranfar, W. A. Simon, M. Zapater, and D. Atienza, "A machine learning-based strategy for efficient resource management of video encoding on heterogeneous MPSoCs," in *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2018, pp. 1–5.

[327] A. Iranfar, M. Zapater, and D. Atienza, "Machine learning-based quality-aware power and thermal management of multistream HEVC encoding on multicore servers," *IEEE Transactions on Parallel and Distributed Systems*, vol. 29, no. 10, 2018.

[328] L. Costero, A. Iranfar, M. Zapater, F. D. Igual, K. Olcoz, and D. Atienza, "MAMUT: multi-agent reinforcement learning for efficient real-time multi-user video transcoding," in *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2019, pp. 558–563.

[329] L. Costero, A. Iranfar, M. Zapater, F. D. Igual, K. Olcoz, and D. Atienza, "Resource management for power-constrained hevc transcoding using reinforcement learning," *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 12, pp. 2834–2850, 2020.

Arman Iranfar

+41 76 271 7237 • arman.iranfar@epfl.ch
https://people.epfl.ch/arman.iranfar?lang=en

Machine Learning researcher skilled in applied supervised, unsupervised and reinforcement learning with 5 years of experience in creating machine learning solutions for complex real-life problems. Knowledgeable about Deep Learning and Convolutional Neural Networks.

Education

- **Swiss Federal Institutes of Technology Lausanne (EPFL)** — **Lausanne, Switzerland**
  PhD Candidate, *Embedded Systems Laboratory, Prof. David Atienza* — *2016 - 2020*
- **University of Tehran** — **Tehran, Iran**
  M.Sc., *Nano-Systems Laboratory, Prof. Ali Afzali-Kusha* — *2013 - 2016*
- **Isfahan University of Technology** — **Isfahan, Iran**
  Undergraduate Student, *Electrical Engineering* — *2009 - 2013*

Research Interests

- **Applied Machine Learning and Deep Learning**
- **Power and Thermal Management**
- **Multi-Objective Design Optimization**

Skills

- **Programming:** Python, MATLAB, Java, C/C++, LaTeX, Bash, Verilog, VHDL, Android
- **Machine Learning Frameworks and Libraries:** Tensorflow, Keras, PyTorch, Scikit-Learn
- **Deep Learning:** Convolutional Neural Networks for Classification, Object Detection, Semantic Segmentation and Recurrent Neural Networks for Language Models.
- **Languages:** English (proficient), Persian (Native), French (B1).

Relevant Projects

- **MANGO: Exploring Manycore Architecture for Next-Generation HPC Systems**
  - **Description:** H2020 European Project FETHPC-2014. Enabling deep customization of architectures to HPC applications with power, performance, and predictability objectives.
  - **Role:** Developing two-phase cooling system and ML-based thermal-aware application scheduling on heterogeneous platforms.
- **DeepHealth: Deep-Learning and HPC to Boost Bio-medical Applications for Health**
  - **Description:** H2020 European Project ICT-2018-2: Combining High-Performance Computing (HPC) infrastructures with Deep Learning and Artificial Intelligence techniques to support bio-medical applications that require the analysis of large and complex bio-medical data sets
  - **Role:** Optimizing CNNs for higher accuracy in power-constrained platforms.
- **Face Authentication with AT&T Database**
  - **Description:** Course Project: Applying Machine Learning (K-Means, GMM, PCA, LDA, etc.)

for face recognition

Publications

- Luis Costero, **Arman Iranfar**, Marina Zapater, Francisco Igual, Katzalin Olcoz, David Atienza, **"Resource Management for Power-Constrained HEVC Transcoding Using Reinforcement Learning**," IEEE Transactions on Parallel and Distributed Systems (TPDS), 2020
- Wellington Silva-de-Souza, **Arman Iranfar**, Anderson Bráulio, Marina Zapater, Samuel Xavier-de-Souza, Katzalin Olcoz, David Atienza, **"Containergy—A Container-Based Energy and Performance Profiling Tool for Next Generation Workloads**," Energies, 2020
- **Arman Iranfar**, Federico Terraneo, Gabor Csordas, Marina Zapater Sancho, William Fornaciari, David Atienza Alonso,**"Dynamic Thermal Management with Proactive Fan Speed Control Through Reinforcement Learning**," In Design, Automation and Test in Europe Conference and Exhibition (DATE), 2020
- Wellington Silva de Souza, **Arman Iranfar**, Anderson Silva, Marina Zapater, Samuel Xavier de Souza, Katzalin Olcoz, David Atienza, **"A QoS and Container-Based Approach for Energy Saving and Performance Profiling in Multi-Core Servers**," In IFIP/IEEE 27th International Conference on Very Large Scale Integration (VLSI-SoC), 2019
- **Arman Iranfar**, Wellington Silva De Souza, Marina Zapater, Katzalin Olcoz, Samuel Xavier de Souza, David Atienza, **"A Machine Learning-Based Framework for Throughput Estimation of Time-Varying Applications in Multi-Core Servers**," In VLSI-SoC, 2019
- Luis Costero, **Arman Iranfar**, Marina Zapater, Francisco Igual, Katzalin Olcoz, David Atienza, **"MAMUT: Multi-Agent Reinforcement Learning for Efficient Real-time Multi-user Video transcoding**," In DATE, 2019
- **Arman Iranfar**, Ali Pahlevan, Marina Zapater, David Atienza, "Enhancing Two-Phase Cooling Efficiency through Thermal-Aware Workload Mapping for Power-Hungry Servers," In DATE, 2019
- José Flich, Giovanni Agosta, Philipp Ampletzer, David Atienza Alonso, Carlo Brandolese, Etienne Cappe, Alessandro Cilardo, Leon Dragić, Alexandre Dray, Alen Duspara, William Fornaciari, Edoardo Fusella, Mirko Gagliardi, Gerald Guillaume, Daniel Hofman, Ynse Hoornenborg, **Arman Iranfar**, *et al.*, **"Exploring manycore architectures for next-generation HPC systems through the MANGO approach**," Journal of Microprocessors and Microsystems, Elsavier, 2018
- **Arman Iranfar**, William Andrew Simon, Marina Zapater, David Atienza, **"A Machine Learning-based Strategy for Efficient Resource Management of Video Encoding on Heterogeneous MPSoCs**," IEEE International Symposium on Circuits and Systems (ISCAS), 2018
- **Arman Iranfar**, Marina Zapater, David Atienza, **"Machine Learning-based Quality-Aware Power and Thermal Management of Multistream HEVC Encoding on Multicore servers**," IEEE Transactions on Parallel and Distributed Systems, 2018
- **Arman Iranfar**, Ali Pahlevan, Marina Zapater, Martin Zagar, Mario Kovac, David Atienza,**"Online efficient bio-medical video transcoding on MPSoCs through content-aware workload allocation**," In DATE 2018
- **Arman Iranfar**, Mehdi Kamal, Ali Afzali-Kusha, Massoud Pedram, David Atienza, **"Thespot: Thermal Stress-Aware Power and Temperature Management for Multiprocessor Systems-on-Chip**," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD), 2017
- José Flich, Giovanni Agosta, Philipp Ampletzer, David Atienza Alonso, Carlo Brandolese, Etienne Cappe, Alessandro Cilardo, Leon Dragić, Alexandre Dray, Alen Duspara, William Fornaciari,

Gerald Guillaume, Ynse Hoornenborg, **Arman Iranfar**, *et al.*, "**MANGO: Exploring manycore architectures for next-generation HPC systems**," Euromicro Conference on Digital System Design (DSD), 2017

- **Arman Iranfar**, Federico Terraneo, William Andrew Simon, Leon Dragić, Igor Piljić, Marina Zapater, William Fornaciari, Mario Kovač, David Atienza, "**Thermal Characterization of Next-Generation Workloads on Heterogeneous MPSoCs**," International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS), 2017
- **Arman Iranfar**, Soheil Nazar Shahsavani, Mehdi Kamal, Ali Afzali-Kusha, "**A heuristic Machine Learning-based Algorithm for Power and Thermal Management of Heterogeneous MPSoCs**," IEEE/ACM International Symposium on Low Power Electronics and Design, 2015
- Mehdi Kamal, **Arman Iranfar**, Ali Afzali-Kusha, Massoud Pedram, "**A Thermal Stress-Aware Algorithm for Power and Temperature Management of MPSoCs**," In DATE, 2015

Teaching Experience

- Micro-programmed Embedded Systems (BS Course), EPFL, 2019
  - **Role:** Lab Supervision
- Design and Optimization of Internet-of-Things Systems (PhD Course), EPFL, EPFL, 2018
  - **Role:** Lab Supervision and Exercises Design
- Digital Electronic Design (BS), University of Tehran, 2014 - 2015
  - **Role:** Lab Supervision and Exercise Design
- Very Large Integrated Circuit Design (MS), University of Tehran, 2014 - 2015
  - **Role:** Lab Supervision and Exercise Design

Honors & Awards

- Awarded 3^{rd} prize on International Low-Power Design Contest, Lausanne, Switzerland, 2019
- Awarded Fellowship of University of Tehran for MS, Iran, 2013
- Awarded Fellowship of Isfahan University of Technology for BS, Iran, 2009

References

- **Prof. David Atienza**, Associate Professor, Embedded Systems Laboratory, EPFL
  - **Email:** david.atienza@epfl.ch
  - **Telephone:** +41 21 693 11 31
- **Prof. Marina Zapater**, Associate Professor, Haute école d'Ingénierie et de Gestion du Canton de Vaud
  - **Email:** marina.zapater@heig-vd.ch
  - **Telephone:** +41 24 557 73 59
- **Prof. Ali Afzali-Kusha**, Professor, Nanosystems Laboratory, University of Tehran
  - **Email:** afzali@ut.ac.ir
  - **Telephone:** +98 21 82 08 49 20