Cache Contention-Aware System-Level Resource Optimization on Clustered Multicores using Machine Learning

Zur Erlangung des akademischen Grades eines

Doktors der Ingenieurwissenschaften

von der KIT-Fakultät für Informatik des
Karlsruher Instituts für Technologie (KIT)

genehmigte

Dissertation

von

Mohammed Bakr Sikal

aus Larache (Marokko)

Tag der mündlichen Prüfung: 25. Juli 2025
1. Referent/Referentin: Prof. Dr.-Ing. Jörg Henkel
2. Referent/Referentin: Prof. Dr. Sridevan Parameswaran

Hiermit erkläre ich an Eides statt, dass ich die von mir vorgelegte Arbeit selbstständig verfasst habe, dass ich die verwendeten Quellen, Internet-Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen – die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Mohammed Bakr Sikal

Acknowledgements

This dissertation is dedicated to my father, Ahmed Sikal, whose values, dreams, and belief in me have been a guiding light throughout my life. Though he is no longer here to witness this milestone, his presence was constant in every step of this journey, and I carry his memory in all that I do.

First and foremost, I want to thank Prof. Jörg Henkel for being far more than just a supervisor. His guidance, trust, and constant support shaped not only my research but also the way I think as a scientist. The environment he built, with its balance of research freedom and direction, gave me the needed space to grow and innovate.

I am grateful to Prof. Sri Parameswaran for agreeing to be my co-advisor. His input, help and generosity, especially in traveling all the way from Sydney to Karlsruhe just to attend my defense, is something I will never forget. That gesture meant a great deal to me.

My heartfelt thanks go to Dr.-Ing. Heba Khdr, who has been a constant in my PhD journey from day zero. For five years, her mentorship has been invaluable, with her incredible scientific precision and rigor, with genuine care for my growth as a researcher and as a person. She has challenged me to think critically, guided me with patience, and never hesitated to give honest, constructive feedback, even when it was difficult to hear. Working with her has not only shaped my research, but also left me with lessons in perseverance, humility, and dedication that I will definitely carry throughout my career.

I am grateful to all my colleagues and office mates for making my PhD an enjoyable journey at the office. My collaborations with Dr. -Ing. Heba Khdr, Prof. Dr. -Ing. Jeferson Gonzalez, Dr.-Ing. Hassan Nassar, Benedikt Dietrich, Dr. -Ing. Lars Bauer, Dr. Kostas Balaskas and Dr. Lokesh Siddhu made research an enjoyable effort. I would like to also thank Dr. -Ing. Martin Rapp for his valuable guidance and collaboration during the first two years of my

PhD, which greatly supported my early steps in this journey. Finally, a special thanks to Prof. Dr. Jeferson and Zeynep, whose friendship brought balance during even the most intense deadlines.

I am profoundly grateful to my mother, Najat Draina, who has been my anchor, my constant source of strength, and my greatest supporter. Her sacrifices have shaped who I am today. For more than three decades, she has been there through every success and every setback, offering unconditional love, encouragement, and faith in my abilities. This achievement is as much hers as it is mine, and it rests on the foundation she built with her dedication, hard work, and boundless care.

To my wife and my partner, Ilham Chekrad, thank you for standing beside me through it all. For celebrating my wins as if they were your own, for picking me up when I stumbled, and for never losing faith in me. You have been my anchor in moments of uncertainty and my greatest source of joy in moments of success. Your patience with the long working hours, your understanding of the sacrifices this journey demanded, and your quiet encouragement on the hardest days kept me moving forward. More than anyone, you have seen the full reality of this process, the stress, the setbacks, the self-questioning—and yet you remained true in your support. This achievement is not mine alone; it is ours, and I am endlessly grateful to share it with you.

To my family: my sister Abir, my brother-in-law Ahmed, to Tita, and to the joyful little newcomers to our lives—my nephew Sami and my niece Lyne Lilia—thank you for filling our family with light. Even in their own small ways, these little ones have brought me moments of pure joy and reminded me of what truly matters beyond work. Your love, encouragement, and pride in my work have meant more to me than I can say.

Finally, I am thankful to my family-in-law for their constant support and encouragement throughout this journey, Tonton Tahar, Tati Fatima, Ali, Khaoula and Soukaina, and I am grateful to be part of such a caring family.

Thank you.

Mohammed Bakr Sikal

List of Publications

The following list enumerates papers and book chapters published by the author of this dissertation while pursuing his doctorate.

First-author publications that present major contributions to this dissertation

[1] Mohammed Bakr Sikal, Heba Khdr, Martin Rapp, and Jörg Henkel. “Thermal- and Cache-Aware Resource Management based on ML-Driven Cache Contention Prediction”. In: *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 2022, pp. 1384–1388. DOI: `10.23919/DATE54114.2022.9774776`

[2] Mohammed Bakr Sikal, Heba Khdr, Martin Rapp, and Jörg Henkel. “Machine Learning-based Thermally-Safe Cache Contention Mitigation in Clustered Manycores”. In: *2023 60th ACM/IEEE Design Automation Conference (DAC)*. 2023, pp. 1–6. DOI: `10.1109/DAC56929.2023.10247708`

[3] Mohammed Bakr Sikal, Heba Khdr, Lokesh Siddhu, and Jörg Henkel. “ML-Based Thermal and Cache Contention Alleviation on Clustered Manycores With 3-D HBM”. in: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)* 43.11 (2024), pp. 3614–3625. DOI: `10.1109/TCAD.2024.3438998`

[4] Mohammed Bakr Sikal, Jeferson González-Gómez, Heba Khdr, and Jörg Henkel. “Contention-Aware Forecasting of Energy Efficiency through Sequence-Based Models in Modern Heterogeneous Processors”. In: *2025 62nd ACM/IEEE Design Automation Conference (DAC)*. 2025

[5] Mohammed Bakr Sikal, Jeferson González-Gómez, Osama Abboud, Xun Xiao, Heba Khdr, and Jörg Henkel. “PHASEL: Learning Phase-Level Application Sensitivities for Energy-Efficient Resource Management in Heterogeneous 6G Systems”. In: Under Review. 2025

[6] Mohammed Bakr Sikal, Jeferson González-Gómez, Heba Khdr, and Jörg Henkel. “ARDiS: A Portable and Unified Resource Management Framework in Real Hardware Systems”. In: *ACM Transactions on Design Automation of Electronic Systems (TODAES)* (2025). Under Review

First-author publications that present minor contributions to this dissertation

[7] Mohammed Bakr Sikal, Hassan Nassar, Heba Khdr, and Jörg Henkel. “A Dataset for LLM-Based Detection of Power-Wasters in Routed FPGA Netlists”. In: *2025 IEEE International Conference on LLM-Aided Design (ICLAD)*. 2025, pp. 235–241. DOI: 10.1109/ICLAD65226.2025.00023

Other co-authored publications

[8] Martin Rapp, Mohammed Bakr Sikal, Heba Khdr, and Jörg Henkel. “SmartBoost: Lightweight ML-Driven Boosting for Thermally-Constrained Many-Core Processors”. In: *2021 58th ACM/IEEE Design Automation Conference (DAC)*. 2021, pp. 265–270. DOI: 10.1109/DAC18074.2021.9586287

[9] Jeferson González-Gómez, Mohammed Bakr Sikal, Heba Khdr, Lars Bauer, and Jörg Henkel. “Balancing Security and Efficiency: System-Informed Mitigation of Power-Based Covert Channels”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 43.11 (2024), pp. 3395–3406. DOI: 10.1109/TCAD.2024.3438999

[10] Jeferson González-Gómez, Mohammed Bakr Sikal, Heba Khdr, Lars Bauer, and Jörg Henkel. “Smart Detection of Obfuscated Thermal Covert Channel Attacks in Many-core Processors”. In: *2023 60th ACM/IEEE Design Automation Conference (DAC)*. 2023, pp. 1–6. DOI: 10.1109/DAC56929.2023.10247844

[11] Heba Khdr, Mustafa Enes Batur, Kanran Zhou, Mohammed Bakr Sikal, and Jörg Henkel. “Multi-Agent Reinforcement Learning for Thermally-Restricted Performance Optimization on Manycores”. In: *2024 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 2024, pp. 1–6. DOI: 10.23919/DATE58400.2024.10546574

[12] Konstantinos Balaskas, Heba Khdr, Mohammed Bakr Sikal, Fabian Kreß, Kostas Siozios, Jürgen Becker, and Jörg Henkel. “Heterogeneous

Accelerator Design for Multi-DNN Workloads via Heuristic Optimization". In: *IEEE Embedded Systems Letters* 16.4 (2024), pp. 317–320. DOI: `10.1109/LES.2024.3443628`

[13] Heba Khdr, Mohammed Bakr Sikal, Benedikt Dietrich, and Jörg Henkel. "Towards the Optimization of Hardware Efficiency through Machine Learning". In: *2025 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. 2025

Abstract

In the past decade, clustered architectures have become the mainstream design of multi-core processors in mobile devices, data center servers, and HPC systems. Unlike monolithic multi-core chips, clustered architectures partition cores into clusters that share resources like the last-level cache, improving scalability and efficiency. Despite these advantages, they also introduce new challenges, particularly concerning cache contention, which becomes more complex to manage than in traditional monolithic designs. In monolithic systems, where all cores share a single last-level cache, application-to-core mapping does not affect cache contention patterns. In clustered architectures, however, an additional optimization dimension emerges: how to efficiently map applications to clusters to minimize contention. Moreover, modern clustered processors, especially heterogeneous ones, feature multi-level shared caches, leading to contention both within and across clusters. As application behavior evolves dynamically, continuous adjustments to mapping strategies are required at runtime.

Cache contention is not an isolated phenomenon. In fact, unmanaged cache contention impacts performance, temperature, and energy efficiency of clustered multicores, potentially leading to execution slowdowns and creating complex tradeoffs. Addressing these tradeoffs necessitates intelligent resource management, including task mapping, migration, and DVFS. To this end, machine learning-based techniques have emerged as a promising solution, enabling predictive and adaptive decision-making to mitigate cache contention in clustered multi-core systems. To this end, *this dissertation presents novel machine learning-based resource management techniques for tackling the cache contention problem and its implications for performance, temperature and energy efficiency, on both homogeneous and heterogeneous clustered multi-core processors.*

First, a contention-aware technique for minimizing the temperature of clustered homogeneous multi-core processors under performance constraints is introduced. The technique employs a neural network-based model to predict the impact of different application-to-cluster mappings and DVFS configurations on the performance of applications considering cache contention effects. After identifying all application-to-cluster mapping options at runtime and eliminating those that violate the performance constraint, the technique evaluates their potential thermal impacts, before applying the configuration that leads to the minimum overall system temperature.

Second, to further explore the implications of cache contention, a machine learning-based contention-aware task migration and DVFS technique to maximize the performance of clustered homogeneous multi-core processors under a thermal constraint is presented. The technique leverages the prediction accuracy of a trained neural network to continuously identify the migration options that would minimize cache contention effects on different clusters. Eventually, the application migration that would maximize the overall system performance without causing thermal violations is chosen.

Third, as clustered multicores started adopting 3D memories like the High-Bandwidth Memory (HBM), a machine learning-based contention-aware technique to maximize the overall system performance under temperature constraints of the processor and the HBM is presented. The technique orchestrates four neural network models at runtime, to continuously identify the migration options that minimize contention effects and maximize the overall system performance, while maintaining the thermal safety of both the processor and the HBM.

Finally, to address the complexity of cache contention in clustered heterogeneous multicores, a novel methodology for training energy efficiency forecasting models is presented. This methodology integrates structured data generation with LSTM- and Transformer-based models to predict future system states under dynamic and previously unseen runtime conditions. This work is made possible by a portable and open-source framework for system-level resource management on real hardware, also a contribution of this research. The framework enables the experimental validation of the proposed methodology on a real clustered heterogeneous multicore, ensuring its effectiveness in real-world deployments.

The evaluations of these techniques on both simulation and real hardware highlight improvements over the state of the art while maintaining a negligible runtime overhead, which demonstrates that machine learning-based approaches, when carefully integrated with system-level resource management, can successfully tackle the cache contention problem, and lead to significant improvement in terms of performance, temperature, and energy efficiency in modern homogeneous and heterogeneous multicores.

Zusammenfassung

In den letzten zehn Jahren haben sich Cluster-Architekturen als dominantes Design für Mehrkernprozessoren in mobilen Endgeräten, Rechenzentrumsservern und HPC-Systemen etabliert. Im Gegensatz zu monolithischen Mehrkernchips partitionieren Cluster-Architekturen die Kerne in Gruppen (Cluster), die Ressourcen wie den Last-Level-Cache (LLC) gemeinsam nutzen, was die Skalierbarkeit und Effizienz verbessert. Trotz dieser Vorteile führen Cluster-Architekturen jedoch auch neue Herausforderungen ein, insbesondere im Hinblick auf Cache-Konkurrenz, die sich komplexer gestaltet als in traditionellen monolithischen Designs. In monolithischen Systemen, in denen alle Kerne einen einzigen Last-Level-Cache teilen, hat die Zuordnung von Anwendungen zu Kernen keinen Einfluss auf das Muster der Cache-Konkurrenz. In Cluster-Architekturen hingegen entsteht eine zusätzliche Optimierungsdimension: die effiziente Zuordnung von Anwendungen zu Clustern zur Minimierung der Konkurrenz. Darüber hinaus weisen moderne Cluster-Prozessoren – insbesondere heterogene Prozessoren – mehrstufige geteilte Caches auf, was zu Konkurrenz sowohl innerhalb als auch zwischen Clustern führt. Da sich das Verhalten von Anwendungen dynamisch verändert, sind kontinuierliche Anpassungen der Zuordnungsstrategien zur Laufzeit erforderlich.

Cache-Konkurrenz ist kein isoliertes Phänomen. Tatsächlich beeinträchtigt eine unzureichend adressierte Cache-Konkurrenz die Leistung, Temperatur und Energieeffizienz von Cluster-Mehrkernsystemen erheblich. Dies kann zu Ausführungsverzögerungen, thermischen Hotspots und unnötig hohem Energieverbrauch führen. Zur Bewältigung dieser Probleme sind intelligente Ressourcenmanagement-Techniken wie Anwendungsverteilung, Thread-Migration und dynamische Spannungs- und Frequenzskalierung (DVFS) erforderlich. In diesem Zusammenhang haben sich Verfahren des maschinellen Lernens als vielversprechende Lösung etabliert, da sie eine vorausschauende und adaptive Entscheidungsfindung zur Minderung der Cache-Konkurrenz in Cluster-Mehrkernsystemen ermöglichen. Diese Dissertation stellt neuartige, auf maschinellem Lernen basierende Ressourcenmanagementtechniken zur Bewältigung des Cache-Konkurrenzproblems und seiner Auswirkungen auf Leistung, Temperatur und Energieeffizienz in homogenen und heterogenen Cluster-Mehrkernprozessoren vor.

Zunächst wird eine konkurrenzbewusste Technik zur Minimierung der Temperatur homogener Cluster-Mehrkernprozessoren unter Einhaltung von Leistungsanforderungen vorgestellt. Die Technik verwendet ein neuronales Netzwerk zur Vorhersage der Auswirkungen verschiedener Zuordnungen von Anwendungen zu Clustern sowie DVFS-Konfigurationen auf die Anwendungsleistung unter Berücksichtigung von Cache-Konkurrenz-Effekten. Nach der Identifizierung aller möglichen Zuordnungsoptionen zur Laufzeit und der Eliminierung derjenigen, die die Leistungsgrenze verletzen, bewertet die Technik deren potentielle thermische Auswirkungen und wählt schließlich die Konfiguration aus, die zur geringsten Gesamttemperatur des Systems führt.

Zweitens wird eine auf maschinellem Lernen basierende, konkurrenzbewusste Technik zur Anwendungsmigration und DVFS vorgestellt, mit dem Ziel, die Leistung homogener Cluster-Mehrkernprozessoren unter Einhaltung eines Temperaturgrenzwerts zu maximieren. Diese Technik nutzt die Vorhersagegenauigkeit eines trainierten neuronalen Netzwerks, um kontinuierlich die Migrationsoptionen zu identifizieren, die die Cache-Konkurrenz in verschiedenen Clustern minimieren. Letztlich wird diejenige Migrationsentscheidung getroffen, die die Systemleistung maximiert, ohne thermische Grenzwerte zu verletzen.

Drittens wird im Zuge der Einführung von 3D-Speichern wie dem High-Bandwidth Memory (HBM) in Cluster-Mehrkernprozessoren eine weitere maschinell lernbasierte Technik vorgestellt, die auf konkurenzbewusste Migration abzielt, um die Gesamtleistung des Systems unter Berücksichtigung thermischer Grenzen von Prozessor und HBM zu maximieren. Die Technik verwaltet zur Laufzeit vier neuronale Netzwerkmodelle, die kontinuierlich die besten Migrationsoptionen zur Minimierung der Konkurrenz und Maximierung der Systemleistung bei Einhaltung thermischer Sicherheitsgrenzen ermitteln.

Abschließend wird zur Bewältigung der Komplexität der Cache-Konkurrenz in heterogenen Cluster-Mehrkernsystemen eine neuartige Methodik zur Modellierung der Energieeffizienz vorgestellt. Diese Methodik kombiniert strukturierte Datengenerierung mit LSTM- und Transformer-basierten Modellen zur

Vorhersage zukünftiger Systemzustände unter dynamischen und im Voraus unbekannten Laufzeitbedingungen. Die Umsetzung dieser Methodik wurde durch ein portables, quelloffenes Framework für das Ressourcenmanagement auf Systemebene auf realer Hardware ermöglicht, welches ebenfalls ein Beitrag dieser Forschung. Das Framework ermöglicht die experimentelle Validierung der vorgeschlagenen Methodik auf einem realen heterogenen Cluster-Mehrkernsystem und gewährleistet so deren Praxistauglichkeit.

Die Evaluation der vorgestellten Techniken auf Simulationsplattformen sowie realer Hardware zeigt deutliche Verbesserungen gegenüber dem Stand der Technik bei vernachlässigbarem Laufzeit-Overhead und belegt, dass maschinelles Lernen ein entscheidender Ansatz zur Bewältigung des Cache-Konkurrenzproblems in homogenen wie heterogenen Cluster-Mehrkernprozessoren ist. Darüber hinaus zeigt sich, dass maschinelle Lernverfahren, wenn sie sorgfältig in das systemweite Ressourcenmanagement integriert werden, das Cache-Konkurrenzproblem erfolgreich adressieren und zu signifikanten Verbesserungen hinsichtlich Leistung, Temperatur und Energieeffizienz in modernen homogenen und heterogenen Mehrkernsystemen führen können.

Research at CES

The Chair for Embedded Systems (CES) at the Karlsruhe Institute of Technology (KIT) conducts research at the forefront of embedded and resource-constrained computing. Its work spans a wide spectrum of topics including system-level optimization, thermal and power management, and machine learning for embedded platforms.

Resource Management and Machine Learning

Resource management plays a critical role in enhancing energy efficiency, thermal behavior, performance, and reliability of multicore systems [14]. CES has built a rich legacy in this area through a series of doctoral research efforts, each advancing the state of the art with increasing granularity and sophistication. This line of research was initiated by CES researchers who pioneered thermal management strategies that operated independently of application-specific behavior. Their work focused on eco-efficient learning techniques for system-level thermal control [15], a foundational contribution that demonstrated the benefits of machine learning for thermal regulation without detailed application characterization. Building upon this, the next generation of research introduced application-awareness into the resource management process. In particular, CES researchers' effort focused on thermal-aware resource management by incorporating application characteristics into the decision-making process, relying primarily on design-time profiling to build predictive models for runtime control [16, 17, 18, 19, 20]. More recently, the research at CES advanced the field by employing machine learning techniques for dynamic resource management. These contributions included sophisticated runtime mechanisms for thermal and performance optimization, notably with *smart* resource management [21, 8, 22]. However, these efforts did not explicitly address the growing problem of cache contention in clustered multicore architectures.

Thesis Alignment This dissertation extends CES's trajectory in resource management by addressing a previously underexplored yet critical challenge: **cache contention-aware system-level resource optimization**. This work focuses on clustered multi-core architectures, where dynamic contention for shared caches can significantly degrade system performance and energy efficiency. This dissertation introduces proactive resource management strategies driven by lightweight machine learning models that operate at runtime. These models predict system behavior under cache contention effects, which enable runtime management of workloads and contention scenarios.

Cross-Layer Security

Cross-layer security is also a key research area at CES, aimed at addressing vulnerabilities across the hardware-software stack. CES researchers develop techniques to protect embedded systems from threats ranging from physical tampering to sophisticated software-based attacks. This includes the use of lightweight cryptographic methods, countermeasures against side-channel attacks [23], attack-resilient hardware primitives, and real-time intrusion detection mechanisms [10]. By applying security measures [9] across abstraction layers, CES aims to significantly enhance the robustness and trustworthiness of embedded systems operating in adversarial environments.

Non-Volatile Memory (NVM)

CES also conducts extensive research on non-volatile memory (NVM) technologies, which offer persistent data storage without requiring power. This line of work explores high-density, low-latency, and durable NVM solutions [24] that are well-suited for emerging memory hierarchies. In particular, CES investigates how the varying retention characteristics of different NVM technologies can be exploited to accelerate machine learning workloads, optimize memory utilization, and improve overall system efficiency.

Reconfigurable Systems

In the area of reconfigurable computing, CES focuses on the development of adaptive, self-organizing architectures capable of dynamic resource allocation.

Leveraging hardware-software co-design, CES engineers systems that reconfigure in real time based on application demands [25]. This reconfigurability not only enables performance and energy efficiency improvements, but also enhances scalability and hardware longevity—particularly in domains such as approximate computing, where minor precision losses can be traded for substantial computational gains [26].

Contents

List of Figures

List of Tables

1. Introduction

The design philosophy of multi-core processors has undergone a fundamental shift from monolithic, i.e., non-clustered designs to *clustered* architectures in the recent years, exemplified by many commercial implementations such as AMD's EPYC [27], ARM's big.LITTLE [28], Intel's Alder Lake [29], and Apple's M1 processor [30]. This evolution has been primarily driven by the stagnation of transistor scaling predicted by Moore's Law [31, 32], the cessation of Dennard scaling [33], and the escalating manufacturing and packaging challenges associated with large monolithic dies [34, 35]. The resulting architectural paradigm, known as *clustered multi-core architectures*, partitions processor cores into multiple physically or logically grouped clusters, each typically having localized caches, memory controllers, and interconnect fabrics [28, 27]. An example of such an architecture is illustrated in the AMD Zen 3 microarchitecture on Fig. 1.1.

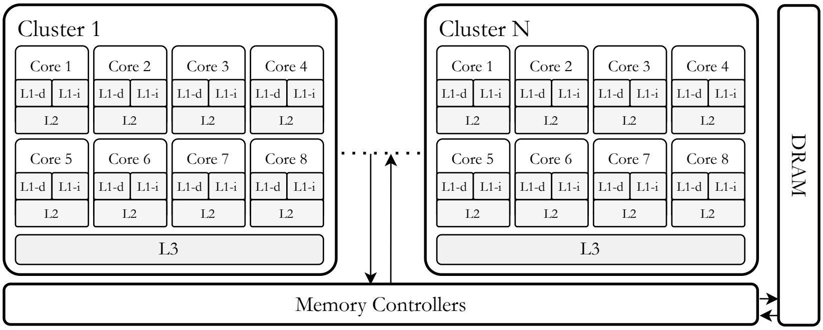


Figure 1.1.: The AMD Zen 3 microarchitecture [36], as an example of homogeneous clustered multicores, where contention for the last-level cache occurs only between concurrently-running applications on the same cluster.

The adoption of clustered multi-core designs has grown significantly across various computing domains. In mobile computing, clustered multi-core processors, exemplified by ARM's big.LITTLE architecture introduced in 2011, achieved widespread adoption due to their ability to dynamically balance high performance and low energy consumption [28]. Today, virtually all flagship mobile devices employ some form of clustered architecture, reflecting universal adoption within the industry [37].

In the realm of data centers, clustered architectures have similarly become the standard. AMD's EPYC and Ryzen server CPUs [38], leveraging a chiplet-based clustered approach, have rapidly gained market share, rising from less than 5% in 2017 to approximately 28% by late 2023, and to a 33% by mid-2024, mainly due to significant improvements in scalability, yield, and power efficiency compared to traditional monolithic server processors [39, 40]. Concurrently, Intel has adopted similar clustered strategies in its most recent server processor lines, e.g., Alder Lake, Sapphire Rapids, etc. to achieve higher core counts and efficiency [41, 29].

1.1. The Cache Contention Problem

Despite the clear advantages of clustered multi-core architectures, e.g., improved scalability, enhanced power efficiency, and greater design flexibility, they introduce significant challenges in the management of shared resources, particularly the cache and memory subsystems. Cache contention, a well-known problem in non-clustered designs, persists in these new clustered designs as well. Unlike monolithic processors, clustered architectures feature complex, hierarchical cache structures. They typically implement hierarchical cache subsystems that comprise multiple cache levels, structured in private caches, e.g., L1, L2, and shared caches, e.g., L3 or last-level cache (LLC), shared between the cores within a cluster or across clusters, as shown in Fig. 1.2. Consequently, applications running in parallel on the system compete for the limited available cache space. This competition results in *cache contention*, where concurrently-executing applications adversely slow down each other's execution, by evicting useful cache lines or saturating shared memory channels [42, 43]. While only chip-wide cache contention is observed on monolithic dies, i.e., all cores on the chip share the same LLC, the new memory hierarchies in modern clustered multicores introduced new forms of contention within clusters (Fig. 1.1) and across clusters (Fig. 1.2).

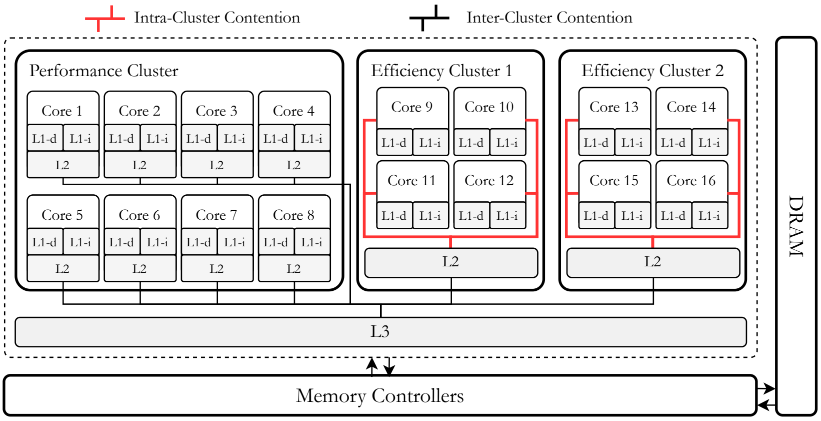


Figure 1.2.: The Intel Alder Lake microarchitecture[29], as an example of clustered heterogeneous multicores where both inter-cluster and intra-cluster contention can occur between concurrently-running applications.

Cache contention is a performance-degrading phenomenon that occurs when multiple concurrently-running applications or processes compete for limited cache resources within a processor [43]. It is primarily caused by two factors: limited cache capacity and conflicting access patterns among applications. In clustered architectures, the LLC remains insufficient relative to the aggregate working set sizes of co-running applications. When applications exhibit substantial spatial or temporal locality—that is, when they frequently access memory addresses that are either close to each other (spatial locality) or reused within short time intervals (temporal locality)— and share a cache level, they tend to compete aggressively for the same cache sets, thereby evicting each other's useful data. Such frequent evictions force the system to access main memory more often, resulting in significantly higher latency and reduced bandwidth [43].

Addressing the Cache Contention Problem

Addressing cache contention involves strategies that span both hardware and software domains. From a hardware perspective, designers employ several strategies to mitigate contention. These include increasing cache associativity, implementing advanced cache partitioning and isolation techniques, e.g., Cache Allocation Technology (CAT) by Intel [44] and developing adaptive cache replacement policies [45]. Such methods aim to manage cache utilization proactively, allocating cache partitions more efficiently or dynamically adapting policies to workload characteristics to minimize harmful interference. While these hardware-centric approaches offer viable methods for contention mitigation, they typically require significant hardware modifications and additional complexity, while being constrained by their inherent cost, limited flexibility, and relatively longer development cycles [46, 47].

In contrast, *system-level resource management (RM)* strategies provide a more flexible, readily-deployable, and cost-effective means to mitigate cache contention [47]. From a software perspective, system-level RM techniques mainly rely on cache-aware application placement, both statically and dynamically. RM techniques can identify applications with conflicting cache behaviors and strategically separate them across clusters, thus minimizing contention. These strategies can incorporate profiling tools to classify workloads based on cache sensitivity, enabling the system to balance cache demands by co-mapping complementary workloads.

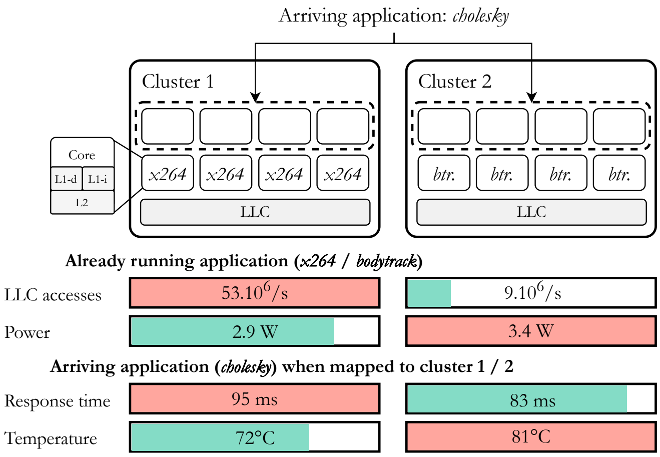
Cache contention remains an open critical performance bottleneck in clustered multi-core architectures, necessitating ongoing innovations in both hardware mechanisms and software management techniques to optimize resource utilization and achieve maximal performance efficiency. Moreover, **cache contention is not an isolated phenomenon**; it also has significant implications for power consumption, temperature management, and overall energy efficiency, creating *complex* tradeoffs between different optimization objectives at runtime, **which have not been observed nor tackled by the state of the art**.

1.2. Motivational Examples

To illustrate the complexity of the cache contention problem, the following subsections present two motivational examples that highlight the tradeoff between cache contention and system temperature, as exposed by application-to-cluster mapping and migration.

1.2.1. Application Mapping

An experiment is conducted on the HotSniper [48] simulator, where two applications from the *PARSEC* [49] and *SPLASH-2* [50] benchmark suites, *x264* and *bodytrack*, are executed on two individual clusters on a simulated clustered homogeneous processor, similar to the AMD Zen 3 microarchitecture (detailed in Section 3.1.1), as shown in Fig. 1.3. Both applications are multi-threaded, running 4 threads in parallel each, and occupying 4 cores on their corresponding cluster, following the one-thread-per-core model [51].



Arriving application: *cholesky*
Cluster 1
Cluster 2
Core
L1-d
L1-i
L2
x264
btr.
LLC
Already running application (*x264* / *bodytrack*)
LLC accesses
53.10^6/s
9.10^6/s
Power
2.9 W
3.4 W
Arriving application (*cholesky*) when mapped to cluster 1 / 2
Response time
95 ms
83 ms
Temperature
72°C
81°C

Figure 1.3.: Application-to-cluster mapping introduces a trade-off between mitigating cache contention and managing temperature, which has not been observed in the literature.

The *x264* application is memory-intensive, which is reflected in its high number of LLC accesses per second, compared to *bodytrack*, which is compute-intensive. The average power consumption of *bodytrack* is higher than the one of *x264*.

A new application, *cholesky*, arrives to the system, and needs to be mapped to one of the two clusters. *Cholesky* also requires 4 cores, as it has 4 threads. Once scheduled, a system-level RM technique shall decide to which cluster and cores to map the threads of the arriving application. As both clusters have 4 available/free cores each, we analyze the implications of the two possible mappings, in terms of performance and system temperature.

When co-mapped with *x264* on Cluster 1, the execution time of *cholesky* is 95 ms. That is 14% slower than its response time when co-mapped with *bodytrack* on Cluster 2, due to the slowdown induced by LLC contention on Cluster 1. From a performance perspective, mapping *cholesky* to Cluster 2 is the better option. However, considering the impact of these mappings on the temperature of the chip suggests otherwise. Co-mapping *cholesky* with *bodytrack* on Cluster 2 leads to an average cluster temperature of 81 °C, due to the high power consumption of *bodytrack*. On the other hand, co-mapping *cholesky* with *x264* on Cluster 1, leads to a 9 °C decrease in the average cluster temperature.

This example highlights that application-to-cluster mapping exposes a trade-off between cache contention and temperature. As will be discussed in Chapter 5, *this trade-off has not been observed in the literature.* Particularly, co-mapping an application with a memory-intensive application increases cache contention and results in an execution slowdown, but might decrease temperature due to the decreased computation on CPU cores. On the other hand, co-mapping with a compute-intensive application decreases cache contention but might increase temperature, as intensive computation typically results in higher power consumption. Depending on the optimization goal at runtime, these observations highlight two possible resource management directions.

Thermally-Constrained Optimization If the chip is thermally constrained to 80 °C, mapping to Cluster 2 leads to a runtime violation of the temperature constraint. Such a violation would trigger the Dynamic Thermal Management (DTM) on the chip to throttle down the cores to the minimum voltage/frequency (V/f) level, until the temperature of the chip is reduced below 80 °C. This would also nullify the intended performance gains from avoiding to map *cholesky* to Cluster 1, where the observed contention-induced slowdown was high. Obviously, if the goal is is to maximize system performance, and temperature is *not constrained* to 80 °C, mapping to Cluster 2 remains the best option.

Performance-Constrained Optimization If the goal is to minimize temperature, mapping *cholesky* to Cluster 1 is clearly the best option. However, if *cholesky* has a defined execution deadline of 90 ms, the achieved 95 ms execution time is a performance violation, making the mapping to Cluster 2 the only option. Obviously, canceling either constraint would result in different options.

Importantly, the analysis in this motivational example *requires* that the cache contention-induced slowdown between the concurrently-running applications on the cluster *and* the impacts on performance and temperature are *known*, based on which the RM can evaluate the implications of the different mapping options. However, at runtime, as applications arrive to the system, this information is *not available*.

1.2.2. Application Migration

In this second motivational example, we conduct experiments on the same HotSniper [48] simulator, simulating three clusters of the same clustered homogeneous processor in the previous example (detailed in Section 3.1.1). This platform operates under a critical temperature threshold of $T_{crit} = 80°C$. The example, illustrated in Fig. 1.4, highlights the consequences of application migration on system performance and thermal behavior, without the interference of emergency thermal management mechanisms such as the Thermal Control Circuitry (TCC) [52].

Initially, all three clusters operate uniformly at a V/f level of 2.6 GHz. Two applications from the PARSEC [49] and SPLASH-2 [50] benchmark suites—*lu.cont* and *x264*—are executed concurrently on cluster 1. Meanwhile, clusters 2 and 3 each host a single application: *water.sp* and *barnes*, respectively. Application performance is measured by execution time.

Our application of interest, *lu.cont*, experiences a performance degradation of 23% when co-executed with *x264* on cluster 1, relative to its standalone execution. This significant slowdown is indicative of severe cache contention

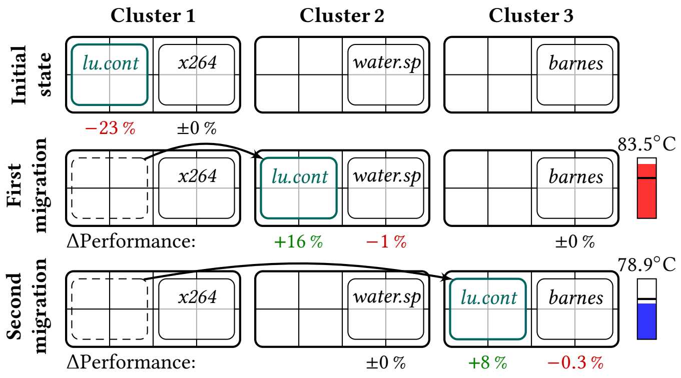


Figure 1.4.: The initial configuration reveals significant cache contention on cluster 1, evidenced by a 23% performance degradation in *lu.cont*'s execution relative to its standalone execution on a single cluster. Two migration strategies are applied to alleviate the contention and are evaluated against the initial setup. The results demonstrate that application migration introduces a trade-off between mitigating cache contention and managing temperature.

between the two applications. To mitigate this, we explore the potential benefits and trade-offs of migrating *lu.cont* to an alternative cluster.

Migrating *lu.cont* to cluster 2 yields a 16% performance improvement for the application, with only a minimal 1% performance impact on *water.sp*. However, this migration elevates the system's peak temperature to 83.5°C, surpassing T_{crit}. Such a violation would activate the TCC, triggering a system-wide downscaling of all core V/f levels to the minimum, which could negate or even reverse the performance gains achieved by mitigating cache contention. In contrast, migrating *lu.cont* to cluster 3 leads to a more moderate performance improvement of 8%, without any performance degradation to *barnes*, and crucially, without breaching the thermal constraint. This outcome preserves the migration-induced performance gains.

This example clearly highlights the necessity of jointly considering cache contention and thermal constraints when performing application migrations to optimize performance under thermal safety requirements.

1.3. Challenges of Cache Contention-Aware Resource Management

Building system-level resource management techniques that consider the cache contention problem *and* its implications faces several challenges.

1.3.1. Challenge 1: Complexity of Contention-Induced Slowdown Behavior

Cache contention-induced slowdown depends on the memory access characteristics, cache sensitivity, and execution behavior of concurrently-running applications. These interactions are highly non linear, asymmetric, application specific and time varying, making the modeling of slowdown effects challenging.

App *A* / App *B*	*streamclus.*	*lu.cont*	*cholesky*	*lu.ncont*
	Simultaneous Start | Shifted Start (40 ms)			
streamclus.	0.0% | 0.0%	20.8% | 19.4%	3.6% | 4.2%	22.0% | 26.5%
lu.cont	0.0% | 0.0%	28.9% | 0.7%	12.5% | 13.0%	20.8% | 17.7%
cholesky	0.0% | 0.0%	27.2% | 0.0%	8.3% | 10.1%	17.4% | 13.8%
lu.ncont	0.0% | 0.0%	26.7% | 0.0%	18.8% | 2.2%	18.8% | 8.9%

Table 1.1.: Comparison of application slowdowns when scheduled simultaneously and with a 40 ms shift in arrival time. Each cell reports the suffered slowdowns suffered by App A when co-mapped with App B, which highlights the asymmetry in the slowdown behavior. Shifting the arrival times of applications by only 40 ms leads to different execution slowdowns.

A concrete illustration is shown in Table 1.1, when representative benchmark applications *streamcluster*, *lu.cont*, *cholesky* and *lu.ncont* are executed in different pair combinations on a cluster, on a simulated homogeneous clustered processor, similar to the AMD Zen 3 microarchitecture. The table highlights multiple observations. First, *streamcluster*, while minimally affected by contention, significantly slows down its co-running applications. Second, *lu.cont* suffers notable performance degradation of 20.8% and 27.2% when co-executed with *streamcluster* and *cholesky*, respectively. Yet, *lu.ncont* exhibits the opposite sensitivity pattern, suffering greater degradation from *streamcluster* (22.0%) compared to *cholesky* (17.4%). Such scenarios illustrate

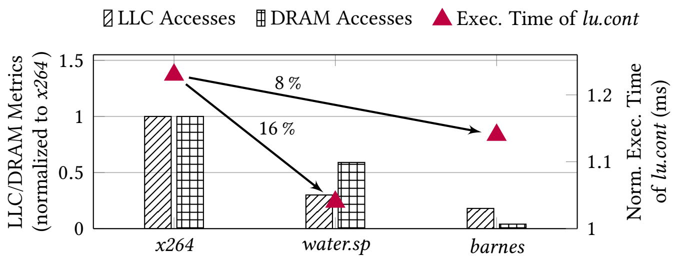


Figure 1.5.: The execution time of *lu.cont* is impacted differently when co-executed with different applications: *x264*, *water.sp* or *barnes* on one cluster. Depending only on DRAM or LLC accesses is not sufficient to estimate cache contention.

the highly asymmetric and application-specific nature of cache contention interactions, making them difficult to capture or generalize through simple heuristic or analytical models.

The complexity increases with higher concurrency. A single AMD Zen 3 cluster can run up to eight threads, and processors like Intel Alder Lake support up to 24 threads contending for a shared LLC. As the number of concurrent applications grows, the number of possible contention scenarios increases exponentially, making exhaustive characterization impractical. Moreover, this complexity is further amplified by temporal variation. Applications exhibit phase-level behavior, alternating between periods of high and low memory intensity. As a result, contention varies over time, even for the same application combination. Additionally, in open systems [53], the arrival order and overlap of applications affect the observed slowdown. Identical workloads arriving in different sequences may lead to different contention patterns due to varying cache states. As can be observed in Table 1.1, the same application pairs experience different slowdowns when their arrival times are shifted by just 40 milliseconds.

Finally, cache contention-induced slowdown cannot be reliably estimated using simple analytical models that are based on the memory access patterns of applications. In Fig. 1.5, we examine LLC and DRAM access counts (following LLC misses), the most common indicators of cache contention behavior in the literature [54, 55, 56], for each of the three applications that co-execute with *lu.cont* in the second motivational example in Section 1.2.2 and Fig. 1.4, along with the corresponding execution time of *lu.cont* in each case. As expected, *x264*, which causes the most significant slowdown for *lu.cont*, shows the highest levels of LLC and DRAM activity, confirming substantial cache contention. However, a surprising observation emerges with *water.sp*. Although it has resulted in the least slowdown for *lu.cont*, it does not exhibit the lowest memory access activity, contrary to expectations. This finding suggests that relying on one or two contention metrics, such as DRAM accesses or LLC accesses, is insufficient for accurately characterizing cache contention. Additionally, when *lu.cont* runs on cluster 3 with *barnes*, it experiences a 14% slowdown, whereas *barnes* itself is affected only minimally (3% slowdown). This highlights that the performance impact of cache contention is not uniform across co-running applications.

In summary, cache contention behaviors are influenced by application combinations, execution phases, and arrival timing. These factors create an exponentially-large and dynamic space of slowdown scenarios *that cannot be tackled with static or simple analytical modeling techniques.*

1.3.2. Challenge 2: Hardware Complexity and Diversity of Memory Hierarchies

Cache contention-aware resource management must account for the diversity and complexity of memory hierarchies in modern multi-core processors. These hierarchies differ significantly across microarchitectures and vendors in terms of cache level organization, capacity, and sharing granularity, i.e., which cores share which cache levels. Such differences influence both the scope and intensity of cache contention at runtime. Accurately modeling these effects requires detailed understanding of the memory structures specific to each platform. A key dimension of this complexity is the configuration of private and shared cache levels. Some architectures implement multiple private cache levels per core and a shared LLC at the cluster level, leading to *intra-cluster* contention between concurrently-running application on the cluster. For example, in AMD's Zen 3 microarchitecture, shown in Fig. 1.1, each core includes private L1 and L2 caches, while all cores within a cluster share a logically-unified L3 cache. In this configuration, contention arises *within the cluster* as co-running applications compete for the shared L3 cache.

Other architectures expose additional complexity by enabling cache sharing across clusters, leading to both intra- and *inter-cluster* contention. Intel's

Alder Lake architecture [29] exemplifies this design, as shown in Fig. 1.2. It combines Performance (P) cores and Efficiency (E) cores, each with different cache configurations. P cores have private L1 and L2 caches, whereas E cores have private L1 caches and a shared L2 per E cluster. All cores, regardless of type, share a chip-wide L3 cache. This structure induces intra-cluster contention within E clusters *and* inter-cluster contention across P and E cores at the L3 level.

These differences have direct implications for system-level resource management. On platforms line the AMD Zen 3-like microarchitecture, RM techniques must primarily address intra-cluster contention. In contrast, on architectures like Alder Lake, both intra- and inter-cluster contention must be considered jointly. Resource management decisions that optimize only for a single cache level may be ineffective if other levels remain contention bottlenecks. Moreover, this hardware diversity also constrains the *generalization* of contention models, as prediction-based approaches must be adapted to each memory hierarchy. Models that abstract away hardware-specific characteristics risk missing critical interactions between cores and caches, reducing their effectiveness for runtime management.

In summary, the diversity and complexity of memory hierarchies across platforms present a fundamental challenge. Resource management strategies must be explicitly tailored to each system's cache configuration to accurately model and mitigate cache contention.

1.3.3. Challenge 3: Implications for Performance, Temperature, and Energy Efficiency

Cache contention is not an isolated phenomenon; it impacts system performance, power consumption, and thermal behavior in interdependent ways, across different system components. While contention primarily manifests as increased application slowdown due to elevated cache miss rates and memory access latency, its consequences also affect power and thermal efficiency. Efforts to minimize contention at the cluster or chip level, by distributing memory-intensive workloads, must also consider co-running compute-intensive applications. These applications are not affected by and do not contribute to cache contention, but they generate high power density on the chip, leading to localized thermal hotspots. Thus, reducing contention alone does not guarantee thermal safety or energy efficiency at the system level, as demonstrated in Fig. 1.3

On the other hand, increased contention leads to higher memory traffic to off-chip DRAM. While it is not problematic for classical 2D DRAMs, this cache contention-induced memory traffic can increase thermal load on modern 3D memories, such as High-Bandwidth Memory (HBM), where vertical stacking constrains heat dissipation, especially in the bottom-most layers furthest away from the cooling system. In such scenarios, a thermal violation on any subsystem, i.e., processing cores or memory, triggers thermal control units, e.g., DTM [52], low-power mode [57], etc., which would throttle down cores or memory channels to the lowest frequency, in order to restore a thermally-safe operation. Therefore, omitting the thermal impacts of RM decisions on the different subsystems may nullify the intended optimization gains.

In summary, cache contention introduces multi-objective trade-offs that cannot be addressed independently. Effective resource management must rely on predictive models that jointly consider the performance impact, energy cost, and thermal footprint of resource management actions.

1.3.4. Challenge 4: The Need for Proactiveness at Runtime

In open systems [53], applications arrive at unpredictable times, requiring the resource management techniques to make immediate application-to-cluster mapping decisions. These decisions must anticipate the impact of cache contention on performance, power, and temperature, based only on the current system state and limited application information. This requires predictive models that are capable of estimating the consequences of *potential* application-to-cluster mappings before effectively applying the RM action. This proactiveness requirement is further complicated by application phase behavior. As applications transition between phases with different memory intensities, previously optimal mappings may become suboptimal. This also necessitates *application migration*. In this context, design-time techniques such as offline profiling or static analytical models are insufficient, as discussed in Section 1.3.1. They cannot adapt to dynamic arrivals, evolving workloads, or shifting contention patterns. Instead, runtime management must rely on lightweight, adaptive models capable of guiding both initial placement and subsequent migration decisions at runtime.

In summary, the dynamic nature of real systems requires proactive, prediction-driven resource management techniques that operate efficiently under time and information constraints.

1.3.5. Challenge 5: Stringent Runtime Requirements

System-level resource management must operate within strict timing constraints imposed by the operating system and underlying hardware. In practice, system-level RM techniques are invoked periodically, at intervals typically in the order of 1 to 10 milliseconds, e.g., for dynamic voltage and frequency scaling (DVFS), thread migration, or DTM [58]. Within each interval, the RM must evaluate the impact of contention, assess multiple alternative mappings, and apply the selected decision, without introducing prohibitive overheads. This requirement introduces significant computational constraints. Resource management decisions rely on predictive models for performance, power, and thermal behavior. When invoked together, these models can impose non-negligible cumulative overhead. If not explicitly optimized, this overhead may exceed the available runtime budget, making the models impractical for deployment. Machine learning (ML) models, while potentially more accurate and adaptive, introduce additional complexity. Unless designed carefully, models such as neural networks (NNs) often require substantial compute and memory resources for inference. Therefore, ML-based predictors must be computationally lightweight and support fast inference to be viable under the tight runtime constraints. Optimization itself must also be tractable. Exhaustively evaluating all mapping alternatives is infeasible due to the combinatorial nature of the search space. As a result, runtime resource management must rely on *smart* techniques to find acceptable solutions within the allowed time window.

In summary, runtime requirements constrain both the complexity of predictive models and the scope of optimization. Resource management techniques must be explicitly designed for low-latency execution, ensuring decisions are made promptly and with minimal overhead.

1.4. Tackling Cache Contention

The challenges presented in the previous section yield one conclusion: *the cache contention problem is too complex to be modeled using analytical models,* especially within the landscape of the increasingly-complex memory hierarchies that characterize modern clustered multicores. Beyond the underlying fixed architecture, the dynamic configuration space affecting the cache contention behavior is exponential and multi-dimensional, considering the number of possible co-running applications, the number of threads per application, their mapping to cores, the operating V/f levels (core-level or cluster-level) and the timing of application arrivals. With variations in any of these parameters substantially altering the contention dynamics at runtime, profiling these scenarios at design time *is therefore impractical* and *does not generalize to unseen applications and runtime scenarios* in open systems.

To address these challenges, the contributions in this dissertation employ machine learning and train highly-accurate prediction models, which learn complex contention patterns from a limited training dataset at design time and generalize to new and unseen runtime scenarios. These models use as input a set of relevant performance monitoring counters that are chosen to effectively capture cache contention effects between concurrently-running applications. The proposed NN models in the individual contributions estimate the impact of potential resource management decisions on one or multiple aspects of the system, including performance, temperature or energy efficiency.

1.5. Thesis Contributions

This dissertation presents novel machine learning-based resource management techniques for tackling the cache contention problem and its implications for performance, temperature and energy efficiency, on both homogeneous and heterogeneous clustered multi-core processors. The key contributions of this dissertation are:

1. *TcRM* [1], a machine learning-based cache contention-aware technique for minimizing the temperature of clustered homogeneous multi-core processors under performance constraints is introduced. The technique employs a neural network-based model to predict the impact of different application-to-cluster mappings and DVFS configurations on

the performance of applications considering cache contention effects. After identifying all application-to-cluster mapping options at runtime and eliminating those that violate the performance constraint, the technique evaluates their potential thermal impacts, before applying the configuration that leads to the minimum overall system temperature.

2. *SmartCM* [2], a machine learning-based contention-aware task migration and DVFS technique to maximize the performance of clustered homogeneous multi-core processors under a thermal constraint is presented. The technique leverages the prediction accuracy of a trained neural network to continuously identify the migration options that would minimize cache contention effects on different clusters. Eventually, the application migration that would maximize the overall system performance without causing thermal violations is chosen.

3. *MTCM* [3], a machine learning-based contention-aware technique to maximize the overall system performance under temperature constraints of the processor and an HBM is presented. The technique orchestrates four neural network models at runtime, to continuously identify the migration options that minimize contention effects and maximize the overall system performance, while maintaining the thermal safety of both the processor and the HBM.

4. PHASEL [5], the training data generation methodology that enabled the ML models in the three previous contributions to capture the complex runtime dynamics that impact cache contention behavior. It addresses the challenge of generating training data for training lightweight ML models, suitable for system-level RM at runtime, by extracting the phase-level behavior of applications through phase-level slicing of their execution traces.

5. ARDiS [6], the first open-source and portable framework for system-level resource management on real hardware. It provides a unified, architecture-agnostic platform for running system-level RM techniques directly on real hardware, by enabling the design, implementation, and evaluation of sophisticated RM strategies, including machine learning-based approaches, with minimal effort and maximum reproducibility.

6. EffiCast [4], a methodology for training energy efficiency forecasting models is presented. This methodology integrates structured data generation with LSTM- and Transformer-based models to forecast

future system states under dynamic and previously unseen runtime conditions. EffiCast is validated on the heterogeneous Intel® Core™ i9 12th generation Alder Lake processor using our proposed ARDiS framework.

1.6. Dissertation Outline

The next chapter discusses the background of cache contention and its mitigation, as well as the state of the art related to the contributions of this dissertation. Chapter 3 presents the system models and the experimental framework used to evaluate the different contributions, including simulation-based and real hardware setups, as well as our proposed ARDiS framework contribution. Chapters 4 to 8 introduce the other five contributions and techniques of this dissertation. Chapter 4 presents our ML training methodology, using which the subsequent techniques have trained their corresponding prediction models. Chapter 5 presents our first ML-based contention-aware application mapping and DVFS technique for temperature minimization. Chapter 6 presents our second ML-based contention-aware application migration and DVFS technique for performance maximization. Chapter 7 presents our third ML-based contention-aware application migration and DVFS technique for performance maximization under temperature constraints of both the processor and the 3D HBM. Chapter 8 presents our ML-based energy efficiency forecasting technique for heterogeneous clustered processors. Finally, Chapter 9 concludes this dissertation and discusses future research directions for tackling the cache contention problem.

2. The State of the Art

This chapter begins by formally defining the cache contention problem and then presents a comprehensive overview of the related literature to the specific contributions of this dissertation. In fact, the problem of shared cache contention has been extensively studied in the literature, resulting in a large body of state-of-the-art techniques that address the problem at both architectural and system levels. A common feature across these approaches is the use of predictive models to estimate the extent of cache contention or to assess its impact on overall system behavior. While earlier methodologies predominantly relied on heuristic or analytical models, which were often customized for particular workloads or hardware platforms, more recent research exhibits a clear shift towards the adoption of machine learning-based models for contention prediction, which have demonstrated improved generalizability across diverse applications and execution environments.

2.1. Cache Contention Background

Cache contention in clustered multi-core architectures emerges as a fundamental bottleneck that degrades application performance. As multi-core systems have evolved from monolithic shared-cache models to hierarchical and clustered cache organizations, the interaction between concurrently-executing workloads has become increasingly complex, as discussed in Section 1.3. In such architectures, different components of the memory hierarchy can be shared, e.g., DRAM, memory controllers, LLC, etc. This structural organization, while beneficial for scalability and power efficiency, introduces heterogeneous access latencies and exacerbates the problem of resource interference.

2.1.1. Mechanisms of Cache Contention

Cache contention manifests in various forms depending on the nature of the workload, the granularity of cache sharing, and the cache coherence protocol employed. The performance degradation due to cache contention is typically quantified through **slowdown**, defined as the ratio between an application's execution time when running concurrently with other applications and its execution time in isolation. Numerous studies have demonstrated that such slowdowns can reach 2–5× in memory-intensive scenarios [59, 60]. In latency-sensitive contexts, even minor contention-induced latency increases can lead to performance/Quality of Service (QoS) constraint violations or service-level agreement (SLA) violations, e.g., in cloud environments. The phenomenon is particularly acute in workloads with large working-set sizes that exceed the available cache capacity, causing frequent cache evictions and memory accesses.

2.1.2. Formulation of the Cache Contention Problem

The cache contention problem can be formulated as follows [43][1]. Let us denote a clustered multicore system as a set of clusters $C = C_1, C_2, \ldots, C_k$, where each cluster C_i contains n_i cores and a shared LLC of capacity S_i bytes. The LLC may be either inclusive or non-inclusive and is typically organized as a set-associative cache with A-way associativity and B sets.

Let $\mathcal{W} = w_1, w_2, \ldots, w_m$ be a set of concurrently executing workloads, each assigned to a subset of cores $\mathcal{K}_{w_j} \subseteq \bigcup C_i$. For a given workload w_j, we define:

- M_j: the LLC miss rate, e.g., misses per thousand instructions, MPKI, etc.
- R_j: the cache reuse distance (average number of distinct cache blocks between two accesses to the same block)
- B_j: memory bandwidth demand (typically bytes/sec)

[1] This problem formulation employs mathematical notations that are only used in the context of this chapter. Section 3.1 introduces the mathematical notations used to formulate the problems addressed in the individual contributions of this dissertation.

- L_j: LLC occupancy (bytes)

Contention-Induced Slowdown

Performance degradation due to contention can be modeled using the contention slowdown factor σ_j for workload w_j:

$$\sigma_j = \frac{T_j^{\text{shared}}}{T_j^{\text{alone}}} \tag{2.1}$$

where T_j^{shared} is the execution time when w_j runs concurrently with other workloads, and T_j^{alone} is the isolated execution time. In practical systems, $\sigma_j > 1$ due to contention.

Empirically, σ_j has been correlated with increased LLC misses and the resulting memory accesses. Let $\Delta M_j = M_j^{\text{shared}} - M_j^{\text{alone}}$ and $\Delta B_j = B_j^{\text{shared}} - B_j^{\text{alone}}$. Then:

$$\sigma_j \approx 1 + \alpha \cdot \Delta M_j + \beta \cdot \Delta B_j \tag{2.2}$$

where α, β are coefficients that are impacted by the characteristics of the workload, the underlying architecture, the memory hierarchy, etc.

Shared Cache Pressure and Evictions

Let the total cache footprint of all workloads sharing cluster C_i be:

$$\Phi_i = \sum_{w_j \in C_i} L_j \tag{2.3}$$

Cache contention occurs when $\Phi_i > S_i$, triggering evictions. The eviction rate E_j for workload w_j is defined as:

$$E_j = \frac{\text{Number of useful lines evicted due to other workloads}}{\text{Total number of lines reused}} \tag{2.4}$$

Existing models in the literature typically attempt to estimate, model or predict this metric, its proxies, or directly its impact on the observed performance of the applications, e.g., through the slowdown, instructions per second (IPS), execution time, etc.

Modeling Cache Contention

A central enabler of cache contention mitigation is the ability to *accurately* predict the occurrence and impact of contention on application performance, both at the architecture [61] and system levels [54]. Simplistic and heuristic-based models have traditionally formed the basis of such predictions [62, 63, 55]. For example, the *Animal* classification scheme proposed in [56] allows classifying applications, in terms of their mutual influence when sharing the same cache space. In this classification scheme, an application can be a turtle (low use of cache space), a sheep (small working set and low miss rate), a rabbit (large working set and low miss rate) or a devil (very high miss rate). Another example, the *Pain* model [55], attempts to characterize applications in terms of their intensity and sensitivity to cache space sharing, relying on the observed performance slowdown and stack distance profiles.

These models, while intuitive, typically assume prior knowledge of application behavior, and thus are ill-suited for open systems [53] where workloads are dynamic and not known a priori. As will be expanded further in the next subsections, recent techniques in the literature propose *cache contention-aware techniques*, that consider or predict *the impact of cache contention* on the different aspects of the system [64, 65, 54], e.g., performance, temperature, power, etc., rather than attempting to model the phenomenon itself through simplistic classification or heuristic-based schemes.

2.2. Architecture-Level Cache Contention Mitigation

Cache contention in multi-core processors has been addressed extensively in the literature, through architectural mechanisms that aim to restrict or regulate the use of shared LLCs. A prominent strategy involves *cache partitioning* [46], which statically or dynamically allocates exclusive portions of the LLC to different processing cores or application domains. Hardware-based solutions, such as *way-partitioning* (or set-partitioning) [66], have been incorporated in commercial processors by vendors including Intel, AMD, and ARM. These mechanisms are effective in reducing inter-application interference and support dynamic reconfiguration, but their granularity is restricted by the number of cache ways, which limits the total number of distinct partitions that can be formed. Consequently, this leads to the underutilization of the cache space in scenarios where many applications with heterogeneous and small working-set sizes are executed.

An alternative approach is *memory page coloring* [67, 68], which leverages control over physical memory allocation to guide how memory pages map to cache sets. By allocating memory such that different applications are mapped to disjoint cache set indices, software can enforce isolation at the level of cache sets. This technique does not require any hardware modifications and has been extensively studied in the context of real-time and embedded systems. Nonetheless, it suffers from significant memory management overheads, particularly in dynamic or general-purpose environments. In fact, page coloring complicates memory allocation due to fragmentation and reduced support for large pages, and incurs significant costs when recoloring is required at runtime.

Beyond partitioning, the design of *cache coherence protocols* [69] has a direct impact on cache contention. Inclusive cache hierarchies, as used in earlier Intel processors, propagate evictions across levels, exacerbating interference by enforcing redundant evictions. More recent processors adopt non-inclusive or *exclusive* cache policies, such as those employed in Intel's Skylake-SP and AMD's EPYC families, which alleviate this issue by decoupling private cache contents from LLC state. Additional mechanisms, such as coherence directories and snoop filters, help reduce coherence traffic and mitigate unnecessary cache pollution, although they typically consume a significant portion of the cache, for example in AMD's implementation where up to 16% of the L3 cache may be reserved for directory structures.

Cache replacement policies also play a significant role in mitigating contention [45], beyond the traditional least-recently-used (LRU) policies. Replacement aware techniques have been proposed to bias eviction decisions based on access patterns or application priorities. Policies with dynamically-tuned algorithms like AARIP [70] or TADRRIP [71] adaptively adjust insertion behavior to enforce fairness or reduce inter-core interference. *Bypassing* techniques [72], wherein certain data streams are excluded from cache insertion are often implemented through dynamic access pattern detection. While this can reduce cache pollution, it also risks cache underutilization if misapplied.

In summary, architectural techniques offer strong isolation and low runtime overhead but are constrained by hardware capabilities and implementation-specific limitations. Their deployment typically still necessitates careful coordination with resource management policies to avoid resource underutilization and ensure efficient cache contention management.

2.3. Pseudo Architecture-Level Cache Contention Mitigation

Recognizing the complexity of fully-architectural approaches, recent processor generations have introduced pseudo-architectural mechanisms that expose certain cache and memory management features to the operating system via software interfaces. Intel's Resource Director Technology (RDT) [73], AMD's Platform QoS [74], and ARM's MPAM [75] exemplify this trend. These interfaces allow software to reconfigure resource partitions at runtime, enabling application-aware control over cache allocation and memory bandwidth distribution. In Intel systems, CAT [76] permits the association of applications with specific compute classes, each of which is assigned a bitmask that controls access to selected LLC ways, which ensures cache space isolation.

These interfaces, though more flexible than hardwired mechanisms, also have significant limitations, as they are highly platform-specific and available only on certain processor models. For instance, only few Intel Xeon processors support the full RDT suite, and comparable features in AMD systems are restricted to more recent server-grade Zen architectures. Moreover, the level of abstraction is high, and the number of isolation domains is limited to the number of configurable compute classes, typically less than a few dozen. This restricts scalability in workloads that comprise many concurrently-running and heterogeneous applications. The implementation details of these interfaces also vary significantly across vendors, complicating cross-platform adoption and policy portability.

In summary, although pseudo-architectural techniques offer a middle ground between static hardware partitioning and fully dynamic system-level management, their constrained expressivity and limited availability hinder widespread deployment. In practice, they are most effective when integrated into coordinated runtime systems that manage cache, bandwidth, and scheduling decisions holistically [44, 61].

2.4. System-Level Cache Contention Mitigation

System-level techniques present an attractive alternative to hardware-centric approaches due to their inherent flexibility and hardware-agnostic nature. These methods rely on the operating system's ability to observe, schedule, and coordinate application behavior using mechanisms such as task mapping, task migration, processor affinity, scheduling, and DVFS. Through active monitoring of performance counters and system metrics, i.e., Performance Monitoring Counters (PMCs) [77], the OS can detect contention and respond adaptively by redistributing applications and threads across cores, adjusting execution priorities, or adjusting processor frequency to influence resource contention patterns. The primary strength of system-level RM techniques lies in their applicability across a broad range of systems and their ability to dynamically adapt to workload changes. While the presence of certain capabilities, such as per-core/per-cluster DVFS or hardware counters, may depend on the underlying hardware, the decision making logic and control mechanisms reside entirely in software. This makes them particularly suitable for open systems and general-purpose operating systems where the applications and runtime conditions are unpredictable and dynamic.

System-Level RM and Machine Learning The effectiveness of system-level strategies depends critically on the accuracy of contention prediction models and the efficiency of runtime management policies. Without accurate models, reactive mitigation may lag behind workload phase transitions, leading to suboptimal performance. As discussed in Section 2.1.2, recent state-of-the-art techniques do not rely anymore on classification or simplistic cache contention models like [55, 56, 78], which assume prior knowledge of application behavior and thus are ill-suited for open systems where workloads are dynamic and not known a priori. Instead, ML-based approaches have been increasingly employed in the recent years [79]. Leveraging hardware performance counters and system-level statistics as input features, ML models can be trained to capture cache contention behavior, and predict its impact on performance (e.g., application slowdown), power, temperature, energy, etc. with high accuracy. The primary advantage of these methods is their ability to operate in open environments without requiring exhaustive profiling or prior characterization.

While software-based techniques avoid the granularity limitations of hardware mechanisms, they face several challenges. Mainly, as discussed in Section 1.3, cache contention is not an isolated phenomenon, as it has implication for other aspects of the system at runtime. Consequently, all recent cache contention-aware works in the literature do consider the impact of contention on some aspect of the system in their optimization, e.g., performance, power or temperature. Related to the contributions of this dissertation, the following presents the state-of-the-art works that propose thermal or performance optimization techniques that considered the cache contention problem, all of which have only addressed *subsets* of the discussed challenges, to achieve their optimization goals, under different constraints.

Temperature Minimization Thermal management techniques have employed various strategies such as task migration, V/f level adjustment, and application mapping. A recent approach based on task migration is presented in [80], where tasks are migrated from thermally stressed cores to relatively cooler ones to reduce chip temperature. Minimizing temperature via DVFS has been widely adopted due to its significant influence on power consumption and, consequently, on temperature [81]. Depending on architectural support, DVFS can be applied at the core level [8] or at the cluster level for clustered multi-core systems [82]. Application mapping is another commonly employed strategy in thermal management, particularly in multi-core architectures. It involves selecting mapping locations such that active cores are surrounded by idle ones, facilitating heat dissipation [16]. For example, the technique in [83] first determines a suitable region on the chip for mapping, and subsequently selects specific cores within this region, aiming to place inactive cores adjacent to the active ones to reduce temperature. Similarly, the approach in [84] employs inter-cluster task migration to distribute thermal load more evenly across the chip. The additional thermal headroom is then utilized by a cluster-level DVFS scheme to enhance performance. Furthermore, reinforcement learning-based techniques have been proposed for optimizing temperature under performance constraints [85, 86], leveraging DVFS and task mapping. More recently, an imitation learning-based method has been proposed in [87] to minimize temperature while satisfying application latency constraints through task migration. However, despite their consideration of temperature, these methods overlook cache contention.

Several prior works have addressed cache contention by modeling the induced slowdowns it causes [63, 62], enabling contention-aware scheduling policies. For example, the technique in [88] integrates cache contention considerations to meet performance requirements. The co-mapping of memory-intensive with compute-intensive applications, shown in [89], helps to alleviate contention by leveraging the lower memory access rates of compute-intensive applications. The authors in [90] propose an ML-based mapping approach that predicts and minimizes the impact of cache contention by allocating applications to clusters with the lowest predicted interference. Nonetheless, these techniques neglect thermal concerns in multi-core systems.

To summarize, existing state-of-the-art solutions have not jointly addressed cache contention and thermal issues, and have not observed the trade-offs that may arise between these two factors.

Performance Maximization Numerous resource management solutions in the literature [8, 91, 92, 80], aim to maximize system performance while adhering to thermal constraints. These solutions employ various methods such as DVFS, application mapping, and task migration. For instance, the technique in [91] utilizes application mapping by configuring different active/inactive core patterns tailored to application behaviors, thereby achieving an improved thermal distribution. The resulting thermal headroom is exploited to elevate core V/f levels, thus boosting performance. However, static mapping approaches fail to adapt to dynamic application behavior. To enhance adaptability, several techniques incorporate dynamic application mapping via runtime thread migration, e.g., from hot to cold cores. For example, [92] migrates threads while accounting for intra-application communication patterns. The technique in [80] dynamically selects between migration and DVFS actions based on their relative performance benefits. Additionally, [84] implements adaptive power budgeting to periodically reassign threads to cores, enabling higher V/f levels for cooler cores. Nevertheless, *none of these techniques addresses the cache contention problem.*

A cache contention-induced slowdown estimation model, derived via design-time profiling, is presented in [62], although it lacks applicability for unseen workloads. More recent techniques have focused on mitigating cache contention through application mapping. For instance, [89] introduces a runtime application migration strategy that periodically monitors shared bus access, such as DRAM accesses during LLC cache misses, to estimate contention levels and migrate applications accordingly. This technique also integrates a neural network to predict the maximum V/f levels that meet power constraints.

In conclusion, *none of the existing contention mitigation techniques ensures thermal safety within the performance maximization framework*, as thermal management is often treated separately and enforced via hardware mechanisms such as TCC or DTM.

HBM Integration None of the aforementioned techniques considers main memory temperature, as they are primarily designed for systems equipped with traditional 2D DRAMs, which are not typically affected by thermal challenges [93]. With the increasing adoption of HBM in modern architectures, more recent studies have turned attention to its thermal limitations. Efforts in [94, 95] explore advanced embedded cooling technologies and architectural improvements to mitigate thermal issues in HBM stacks. The study in [96] integrates thermal awareness by considering memory-channel temperatures in memory-page allocation to prevent thermal hotspots in frequently accessed memory banks. Similarly, [97] employs reinforcement learning to manage performance while maintaining thermal constraints across both processor and HBM, combining DVFS and low-power modes. Another recent contribution, [98], proposes a comprehensive performance maximization framework for systems combining multi-core processors with HBM. By associating core groups with specific memory channels, their method dynamically adjusts application mappings to manage HBM temperatures, while DVFS and low-power states are utilized to uphold thermal safety.

In summary, with respect to architectures integrating HBMs and clustered multicores, a significant gap remains in the current literature: *no resource management solution jointly addresses cache contention in the context of performance maximization while ensuring thermally safe operation of both the multicore and the HBM.*

3. System Models and Experimental Framework

This chapter presents the system models considered in the different contributions of this dissertation, as well as the experimental framework used to validate them. In fact, the contributions of this dissertation are evaluated on three different experimental setups: 1) the HotSniper[48] simulator was used to validate the *TcRM* [1] and *SmartCM* [2] techniques on an AMD Zen 3-like 64-core homogeneous clustered processor - 2) the CoMeT[99] simulator to validate *MTCM* [3] on the same processor with a an integrated 3D HBM - 3) our proposed ARDiS framework[6] was used to validate EffiCast [4] on the heterogeneous Intel® Core™ i9 12th generation Alder Lake processor.

3.1. Processor and Memory System Models

The processor and memory systems in this dissertation are modeled as follows[1]. We consider a *clustered multi-core system* composed of N processing cores, organized into C clusters. Each cluster consists of $n = N/C$ cores that share one or many cache levels. If the architecture supports per-cluster DVFS, cluster-independent control of f_c is possible where all cores on the cluster share the same V/f level, denoted by $f_c \in [f_{\text{min}}, f_{\text{max}}]$, where $c \in \{1, \ldots, C\}$. The mapping of cores to clusters is defined by a binary matrix $\mathbf{Q} = [q_{i,c}] \in \{0, 1\}^{N \times C}$, where $q_{i,c} = 1$ if and only if core i belongs to cluster c. Each core consumes power p_i, comprising both dynamic and leakage components, and we define the power vector as $\mathbf{P}_{\text{cores}} = [p_1, \ldots, p_N]$. The thermal behavior of the processor subsystem is modeled using the standard RC-based

[1] Table 3.1 summarizes all mathematical notations used to describe the system model.

thermal model [100], denoted by $\mathrm{TM}_{\mathrm{cores}}$. Given $\mathbf{P}_{\mathrm{cores}}$, it estimates the steady-state temperature vector $\mathbf{T}_{\mathrm{cores}} = [t_1, \ldots, t_N]$, where t_i is the temperature of core i. Thermal safety requires that the maximum core temperature remains below a critical threshold T_{crit}, i.e., $\max_i t_i < T_{\mathrm{crit}}$.

Cache Hierarchy. Each core has access to a memory hierarchy of caches. This typically includes private caches (e.g., L1, L2) and shared caches (e.g., L3). Shared caches may be configured in different ways:

- **Intra-cluster sharing**: All cores within a cluster share a single LLC, e.g., AMD Zen3[36]. Cache contention on these platforms arises only between applications co-running on the same cluster.
- **Inter-cluster sharing**: A shared cache spans multiple clusters, e.g., chip-wide L3 on the Intel i9 Alder Lake[29]. In this case, cache contention may occur between applications mapped to different clusters but sharing the same LLC.
- **Hybrid configurations**: A combination of both intra- and inter-cluster sharing policies, e.g., P and E clusters on the Intel i9 Alder Lake[29], allowing complex contention behaviors and varied cache interference domains.

Application Model. We assume an *open system* [53], where applications arrive dynamically at unknown times. Let K denote the number of concurrently executing multi-threaded applications. Each application $k \in \{1, \ldots, K\}$ consists of h_k threads and follows the widely used one-thread-per-core execution model [51], thereby requiring h_k distinct cores.

Application mappings are specified using two binary matrices:

- $\mathbf{G} = [g_{c,k}] \in \{0,1\}^{C \times K}$: $g_{c,k} = 1$ if application k is assigned to cluster c,
- $\mathbf{V} = [v_{i,k}] \in \{0,1\}^{N \times K}$: $v_{i,k} = 1$ if a thread of application k is mapped to core i.

All threads of the same application are mapped to a single cluster, while multiple applications may share the same cluster. Application performance may be characterized either by execution time or by the achieved instructions

Table 3.1.: Summary of Mathematical Notations

Symbol	**Description**
System Structure	
N	Total number of cores
C	Number of clusters
$n = N/C$	Number of cores per cluster
$f_c \in [f_{\text{min}}, f_{\text{max}}]$	Voltage-frequency level of cluster c
$\mathbf{Q} = [q_{i,c}]$	Binary matrix: core i belongs to cluster c
$\mathbf{P}_{\text{cores}} = [p_i]$	Power consumption of core i
$\mathbf{T}_{\text{cores}} = [t_i]$	Steady-state temperature of core i
TM_{cores}	Thermal model for processing cores
T_{crit}	Maximum safe temperature threshold (cores and HBM)
Application Model	
K	Number of multi-threaded applications
h_k	Number of threads of application k
$\mathbf{G} = [g_{c,k}]$	Binary matrix: application k assigned to cluster c
$\mathbf{V} = [v_{i,k}]$	Binary matrix: thread of application k mapped to core i
IPS_k	Instructions per second of application k
R_k	Response time of application k
$\hat{R}_k$	Response time deadline for application k
HBM Memory Model	
L	Number of HBM memory layers
B	Number of memory banks per layer
S	Number of memory channels
B_{ch}	Set of banks assigned to channel ch
ch_c	Memory channel accessed by cluster c
MC_c	Memory controller associated with cluster c
$\mathbf{P}_{\text{banks}}$	Power consumption of HBM banks
$\mathbf{T}_{\text{banks}} = [t_{b,\ell}]$	Temperature of bank b in layer ℓ
TM_{hbm}	Thermal model for HBM subsystem
T_{ch}	Maximum bank temperature in channel ch

per second (IPS_k). If deadline constraints apply, the response time R_k of application k must satisfy $R_k \leq \hat{R}_k$, where $\hat{R}_k$ is the specified upper bound.

Memory Subsystem. To model the memory subsystem in our *MTCM* [3] contribution, we also consider a HBM architecture integrated with the processor. Each cluster c is associated with a dedicated memory controller MC_c, which accesses a unique HBM channel ch_c. Cores in cluster c issue memory requests exclusively through MC_c.

The HBM is composed of L vertically stacked memory layers, each containing B banks. These banks are distributed across S physical memory channels. Each channel $\mathrm{ch} \in \{1, \ldots, S\}$ manages a disjoint subset of banks $B_{\mathrm{ch}} \subseteq \{1, \ldots, B\}$, and the temperature of channel ch is defined as:

$$T_{\mathrm{ch}} = \max_{b \in B_{\mathrm{ch}}} t_b,$$

where t_b is the temperature of bank b in the assigned layer.

Thermal behavior in the memory subsystem is modeled using $\mathrm{TM}_{\mathrm{hbm}}$, which estimates the memory-bank temperature matrix $\mathbf{T}_{\mathrm{banks}} = [t_{b,\ell}] \in \mathbb{R}^{B \times L}$, based on the per-bank power dissipation map $\mathbf{P}_{\mathrm{banks}}$. Thermal safety in the memory is similarly governed by the threshold T_{crit}, as specified by JEDEC standards [101], i.e., $\max_{b,\ell} t_{b,\ell} < T_{\mathrm{crit}}$.

Fallback Memory Model. In the contributions where HBM is not considered, we assume a conventional 2D memory system with uniform access latency and bandwidth across all cores. In this case, the memory controller is shared.

3.1.1. Architectural Instances

In the following subsections, we instantiate this model for the platforms considered in the contributions of this dissertation, featuring different cache/memory configurations and core architectures:

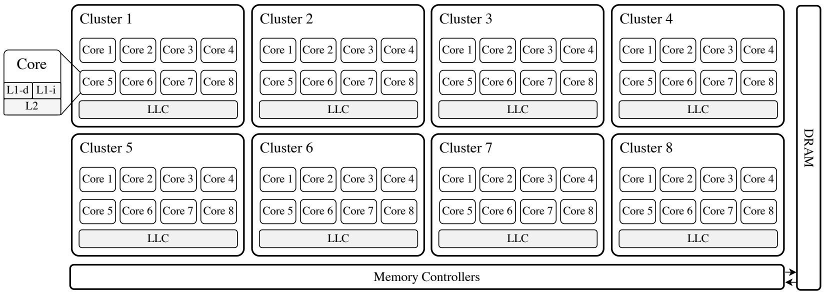


Figure 3.1.: The 64-core homogeneous clustered processor, following the AMD Zen 3 microarchitecture [36].

(A) 64-Core AMD Zen 3-Like Homogeneous Clustered Processor

We model a 64-core homogeneous clustered multicore, similar to AMD's Zen 3 microarchitecture Fig. 3.1. Each core has private 32KB L1 instruction and 32KB L1 data caches, and a private 256KB L2 cache. Each cluster on the multicore groups 8 cores, sharing one 8 MB LLC and one memory controller. The architecture supports per-cluster DVFS, where all cores on the cluster share the same V/f level, with frequency levels ranging from 1 GHz to 4 GHz in increments of 200 MHz.

(B) 16GB Samsung HBM2E

The HBM main memory is modeled based on the 16 GB Samsung HBM2E in [102], with 8 channels spanning 128 banks. The access latency is 6.73 ns latency and the per-access energy is 9.51 nJ, obtained using Cacti-3DD[103] based on [102]. The memory controller of each cluster on the multicore is mapped to one of the 8 channels of the HBM, as shown in Fig. 3.2.

(C) 24-Core Intel i9 12th Gen Heterogeneous Clustered Processor

We model the Intel i9 12th Generation processor as a heterogeneous clustered multi-core system comprising 24 physical cores: 8 high-performance (P) cores and 8 energy-efficient (E) cores, as shown in Fig. 3.3. The architecture

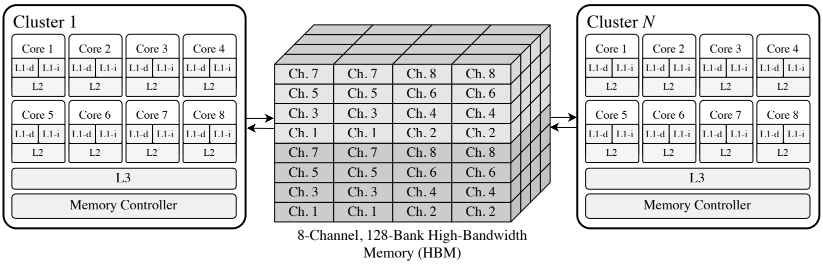


Figure 3.2.: The Samsung HBME2 [102] as the main memory of the simulated AMD Zen 3 homogeneous clustered processor.

is composed of two types of clusters: P-core clusters and E-core clusters. Each P-core has private 32 KB L1 instruction and 48 KB L1 data caches, and a private 1.25 MB L2 cache. The E-cores are grouped into clusters of four, with each cluster sharing a 2 MB L2 cache; each E-core also has private 32 KB L1 instruction and 32 KB L1 data caches.

All cores access a shared 30 MB LLC. This enables inter-cluster cache contention between concurrently running applications on both P and E-core clusters. Due to the architectural asymmetry, P and E cores operate at distinct voltage-frequency domains with independent DVFS control. All E cores in each E cluster share the same V/f level, while P cores are clocked independently with per-core DVFS.

3.2. Simulation Frameworks

The target 64-Core AMD Zen 3 homogeneous clustered processor[36] and the 16GB Samsung HBM2E[102] are simulated on the HotSniper[48] and CoMeT[99] simulators, respectively. In the following subsections, we present the technical details of the two simulators.

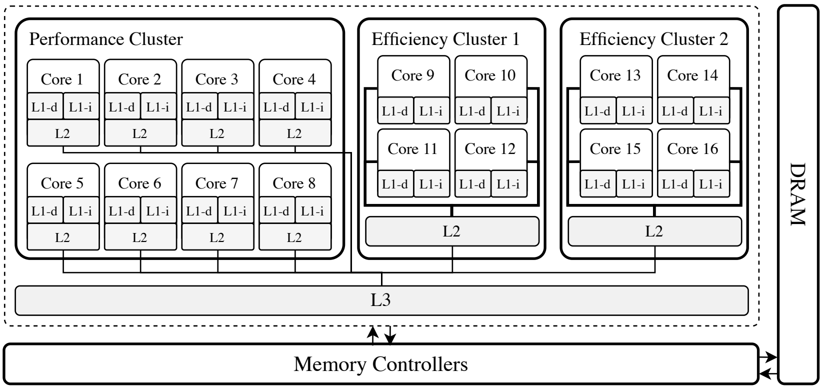


Figure 3.3.: The 24-core Intel i9 12th Gen Alder Lake heterogeneous clustered processor [29].

3.2.1. HotSniper

The HotSniper simulation framework [48] is used to evaluate the *TcRM* [1] and *SmartCM* [2] contributions in this dissertation, which target 64-Core AMD Zen 3 homogeneous clustered processor[36]. HotSniper integrates performance, power, and thermal simulation capabilities by combining Sniper [104] for cycle-accurate multi-core performance simulation, McPAT [105] for power estimation, and HotSpot [100] for temperature modeling. Sniper models multi-threaded and multi-program workloads on multi-core systems, with accurate representation of shared resource contention, such as caches and memory bandwidth. McPAT is invoked at regular simulation intervals to estimate dynamic and leakage power consumption, based on the underlying microarchitectural activity. The combination enables fine-grained analysis of the effect of resource management actions, e.g., DVFS, thread migration, etc. on system-wide performance and power. HotSniper also supports open-system simulation with configurable task arrival models (uniform, Poisson, or user-defined), enabling dynamic workload behavior analysis.

HotSniper integrates HotSpot [100] for interval-based thermal simulation of both the processor and its cooling system. HotSpot models the thermal behavior of the chip using an equivalent RC network, where temperatures correspond to electrical potentials and heat flow to currents. The full system, consisting of the processing cores and associated structures, e.g., private caches, LLC banks, NoC routers, etc. as well as the thermal interface material, heat spreader, and heat sink, is represented by a number of thermal nodes on a *floorplan*. The cooling system is modeled using HotSpot's default parameters, with an ambient temperature of 45°C and a critical thermal limit T_{crit} set to 80°C.

To support advanced resource management experimentation, HotSniper supports a modular resource management API and a simulation automation workflow. The API allows custom RM policies to be implemented through well-defined C++ interfaces, supporting actions such as thread migration, application mapping, and DVFS. The automation infrastructure supports different benchmarks (detailed in Appendix A) and enables the-execution of varied workloads and policies and provides a Python-based API for result collection and post-processing. The framework provides fine control over simulation parameters and observability of detailed runtime metrics, making it well-suited for validating the predictive accuracy and performance impact of the techniques proposed in this dissertation.

The architecture is modeled in 14 nm FinFET technology, with power models scaled from 22 nm data using established scaling factors [106]. DVFS frequencies range from 1.0 GHz to 4.0 GHz in 100 MHz increments.

3.2.2. CoMeT

The CoMeT simulation framework [99] is employed to validate the *MTCM* [3] technique in this dissertation, which employs the 16GB Samsung HBM2E[102]. CoMeT is an integrated simulation toolchain that supports interval-based performance, power, and thermal simulation of both cores and memory, specifically designed to address the thermal modeling challenges of high-density processors such as 2.5D and 3D-stacked core-memory architectures. It enables temperature-aware modeling of systems with various core-memory configurations, including off-chip 3D memories.

CoMeT also integrates the Sniper interval performance simulator [104], McPAT [105] for core power modeling, CACTI-3DD [103] for memory power modeling, and HotSpot [100] for thermal simulation. At each user-defined interval, performance counters are collected and used to estimate dynamic power for both core and memory. These power estimates are passed to

HotSpot along with a detailed floorplan to compute temperatures, accounting for both dynamic and temperature-dependent leakage power. This data is used to trigger dynamic DTM policies when thermal thresholds are violated, including the default low-power mode.

Similar to HotSniper, CoMeT also provides an interface to develop and evaluate custom thermal-aware resource management policies, including task mapping, thread migration, and DVFS. With regards to floor-planning, CoMeT includes a `floorplanlib` utility to generate regular or fine-grained floorplans for any core-memory layout, including multi-layer 3D stacks. Additionally, the `HeatView` module generates thermal videos that visually capture heating trends across simulation epochs, aiding qualitative understanding of spatiotemporal thermal behavior.

The `SimulationControl` module in CoMeT allows for batch execution of parameterized simulations, with Python APIs for defining workload variants, retrieving output metrics, and generating visualizations.

3.3. ARDiS: Experimentation on Real Hardware

To evaluate our EffiCast [4] on the heterogeneous Intel® Core™ i9 12th generation Alder Lake processor, we use our proposed ARDiS [6], the first open-source and portable framework to provide a unified, architecture-agnostic platform for running system-level RM techniques directly on real hardware. ARDiS eliminates the need to "reinvent the wheel," by enabling the design, implementation, and evaluation of sophisticated RM strategies—including machine learning-based approaches—with minimal effort and maximum reproducibility.

3.3.1. Motivation & Related Work

The increasing complexity of modern computing systems has elevated system-level RM to a critical area of research, with goals ranging from enhancing performance and energy efficiency to addressing security and reliability concerns. RM techniques focus on tuning system-level parameters to optimize resource allocation under diverse constraints. These knobs include DVFS, application-to-core mapping, task migration, and scheduling strategies.

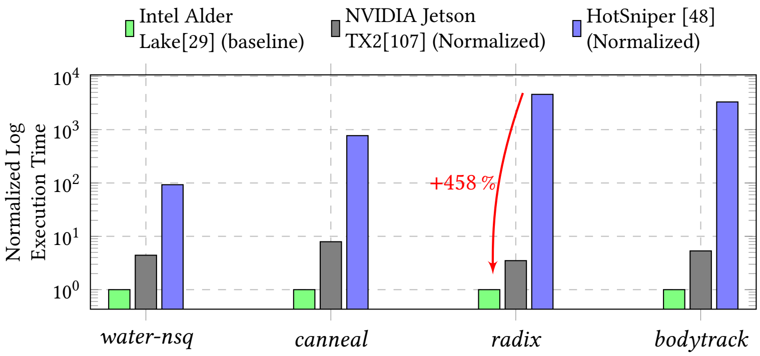


Figure 3.4.: Although simulators offer substantial advantages for architectural exploration, they typically incur considerably longer execution times than real hardware. This figure presents the normalized execution times of several benchmark applications, illustrating the substantial slowdown observed in the HotSniper [48] simulator—often several orders of magnitude greater than that of real hardware platforms like Intel Alder Lake and NVIDIA Jetson TX2. These prolonged simulation durations can pose a significant obstacle to system-level resource management studies.

As discussed in Section 1.3, effective resource management is essential for maintaining optimal system behavior in the presence of heterogeneous and dynamic workloads.

The Simulation Route Historically, simulation frameworks have been widely used to design and evaluate RM strategies [108, 109, 8, 22, 110, 111]. Simulation platforms generally fall into two categories: high-fidelity and low-fidelity simulators. High-fidelity simulators, such as those offering cycle-accurate modeling [112, 113], provide detailed system behavior but are often prohibitively slow for exploring complex RM schemes. Conversely, low-fidelity simulators [114, 115] offer faster simulation times but lack the precision needed to advance state-of-the-art RM research. This trade-off between fidelity and computational efficiency has emerged as a bottleneck, particularly in the context of ML-driven RM techniques [79].

To mitigate these limitations, interval simulators [116, 48, 99] have been proposed as a middle ground, offering a compromise between simulation accuracy and runtime performance. Unlike high-fidelity simulators that model execution at the instruction or cycle level, or low-fidelity simulators that rely on coarse abstractions, interval simulators divide execution into discrete time intervals [104]. Within each interval, performance is estimated using simplified analytical models, reducing simulation time while preserving reasonable accuracy. Despite these advantages, interval simulators still face a key limitation: their execution time remains several orders of magnitude higher than real hardware, as discussed in Chapter 4. Figure 3.4 illustrates this limitation, comparing the execution time of four single applications from the PARSEC [49] and SPLASH-2 [50] suites on three platforms: the heterogeneous Intel® Core™ i9 12th generation Alder Lake processor, an embedded NVIDIA Jetson TX2 board[107], and the HotSniper simulator [48] running on a server-class AMD Ryzen 7x host. Such extended run-times hinder the evaluation of complex RM strategies, especially for large-scale workloads or scenarios requiring real-time adaptivity.

An increasingly viable alternative is to implement and evaluate RM techniques directly on physical hardware. Unlike simulators, real systems offer precise measurements without relying on modeling assumptions. Recent studies have explored this direction [54, 117, 111, 118], including approaches based on ML. These works highlight several advantages of hardware-based RM evaluation, such as improved accuracy, reduced experimentation time, and applicability to real-world deployment scenarios.

The Literature Gap Despite these benefits, real-hardware-based RM research is hindered by several practical challenges. Modern hardware platforms are highly heterogeneous and lack standardized experimentation frameworks. As a result, researchers often develop *ad hoc* tools and methodologies, leading to solutions that are not portable across architectures and difficult to reproduce or scale. These limitations restrict broader adoption and hinder progress in the field. To address these shortcomings, we propose ARDiS, the first open-source[2] and *portable* framework for system-level resource management on real hardware platforms. ARDiS bridges this gap between simulation flexibility and hardware realism by offering a unified, architecture-agnostic, and extensible infrastructure.

Specifically, ARDiS enables researchers to:

[2] Will be made publicly available on GitHub upon acceptance (currently under review).

- Implement and evaluate advanced RM techniques, including those based on ML, with minimal setup overhead;
- Achieve high-fidelity performance evaluation by leveraging real hardware behavior;
- Promote reproducibility and portability across hardware platforms;
- Reduce development time through reusable and modular components.

3.3.2. Design Principles, Requirements and Challenges

To effectively address the limitations identified in existing literature and overcome practical constraints associated with real hardware-based RM, our proposed framework is guided by a set of design principles and must tackle several technical challenges. The following subsections outline the core principles and requirements that shape the framework's design, followed by a discussion of key implementation challenges and the strategies employed to resolve them.

Design Principles and Requirements

In designing a robust and adaptable framework capable of bridging gaps in current resource management research and practice, we identify the following core design principles and requirements: ① **Portability** – The framework must be hardware-agnostic, ensuring compatibility across diverse architectures and platforms. ② **Comprehensive RM Support** – It should provide integrated support for a range of resource management objectives and control knobs. ③ **Support for Training Data Generation** – The framework should facilitate efficient and flexible collection of high-quality training data for ML-based RM methods. ④ **Ease of Use and Extensibility** – It should be modular, open-source, and designed for straightforward extension to promote adoption within the research community.

To meet the goal of ① portability, we implement a hardware abstraction layer that defines a unified interface for interacting with different hardware targets. This abstraction allows resource management policies to be developed independently of hardware-specific constraints. The overall architecture of

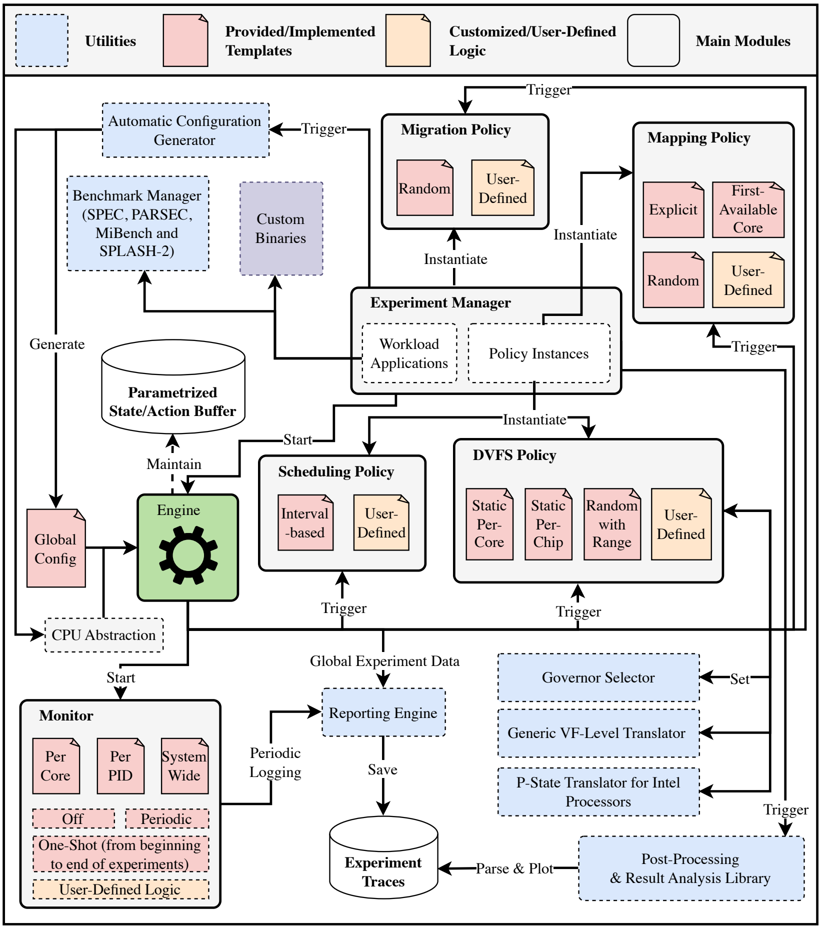


Figure 3.5.: High-level overview of ARDiS's architecture and modules.

the framework follows a layered design, promoting a clear separation of concerns and enabling modularity across decoupled components.

In support of (2) comprehensive RM, the framework is organized around key control modules responsible for scheduling, DVFS management, application mapping, and task migration. Each module is designed as a configurable component, allowing researchers to define, customize, and experiment with various RM strategies within a cohesive environment.

For ③ training data generation, we provide a highly flexible monitoring module that can be tailored to user needs. This module allows users to configure the scope of monitoring (per-core, per-process, or system-wide), the set of metrics to be collected (e.g., execution time, power usage, cache statistics), and the granularity of data collection. Additionally, it maintains a continuously-updated, parameterized buffer that supplies real-time statistics. This buffer can be queried periodically by RM policies and serves as input to machine learning models for runtime inference.

To fulfill ④ ease of use and extensibility, the entire framework is implemented in *Python*, maximizing accessibility and lowering the barrier to entry. All modules are parameterized, and OS-specific operations are abstracted into high-level function calls to improve portability and maintainability. Moreover, by making the framework open-source, we aim to foster community-driven development, encourage reuse, and facilitate collaborative advancements in hardware-based resource management research.

Design Challenges

Despite the modular and configurable structure of ARDiS, several technical challenges arise in its development and deployment.

Challenge 1: Balancing Generalization and Complexity One of the core design challenges is identifying the appropriate level of generalization for RM functionalities. The framework must be sufficiently flexible to accommodate a wide range of RM techniques, while avoiding unnecessary architectural complexity and redundant abstractions. To strike this balance, we conducted a comprehensive survey of existing RM methodologies in the literature, ensuring that our design aligns with common patterns while maintaining manageable complexity.

Challenge 2: Synchronization Across Multiple RM Knobs Advanced RM strategies frequently involve coordinating multiple control knobs—such as DVFS in conjunction with task migration—which introduces challenges in synchronization. Uncoordinated operations on shared system resources (e.g., frequency settings or thread affinities) may result in race conditions, inconsistent system states, or degraded performance. To address this, we implement an *epoch-based synchronization* mechanism. All RM policies are executed within predefined epochs, and updates are applied only at epoch boundaries, triggered by a global synchronization signal. This guarantees consistent and coordinated execution of RM decisions.

As illustrated in Fig. 3.6, our monitoring and reporting infrastructure uses *Queues* to distribute observed metrics concurrently to multiple consumers, allowing parallel access without interference. Execution priority is enforced through a structured policy order. Furthermore, we employ *Read-Write Locks* to allow concurrent read access to the system state, while ensuring exclusive write access when updates are applied. To prevent race conditions in multi-threaded environments, *Thread Locks* enforce mutual exclusion when modifying shared data structures, such as per-core frequencies or task mappings.

This combination of epoch coordination and fine-grained synchronization primitives enables multi-knob RM techniques to operate safely and efficiently in parallel.

Challenge 3: Ensuring Lightweight Operation For the framework to be viable in real-world scenarios, it must impose minimal overhead on the system. Excessive resource usage by the RM infrastructure itself could negatively impact the applications being managed. To minimize overhead, we optimize the implementation using Python's multi-threading capabilities to reduce blocking operations and avoid contention during RM policy execution. Buffered communication and asynchronous operations reduce latency and decouple monitoring from decision logic. Finally, the framework operates entirely in *user space*, avoiding any modifications to the operating system kernel, which further reduces complexity and runtime overhead.

3.3.3. The ARDiS Resource Management Framework

An overview of ARDiS is shown in Fig. 3.5. ARDiS is architected as a modular and extensible framework that enables the implementation and execution of system-level RM techniques in a unified, hardware-agnostic manner on real hardware platforms. The following subsections detail the core RM functionalities supported by the framework, discuss the design principles and associated challenges, and describe the execution workflow and internal architecture of its components.

Resource Management Policies

ARDiS supports the primary system-level resource management knobs, including: *scheduling*, application/thread *mapping*, application/thread *migration*, and DVFS. Users can select from default policy templates provided by the framework or define their own custom policies for each of these RM components.

The *Scheduling Policy* module governs application launch times and execution order across cores. It includes predefined scheduling strategies such as simultaneous launches, static sequential schedules, and randomized dynamic schedules based on probabilistic models.

The *Mapping Policy* controls how applications or threads are assigned to specific cores. Two main strategies are supported: *static mapping*, which assigns threads to fixed cores for the entire execution duration, and *dynamic mapping*, which defines only the initial placement of threads at application startup. Complementing this, the *Migration Policy* enables dynamic adjustments of thread-core affinities during runtime, either based on system conditions or user-defined heuristics.

The *DVFS Policy* adjusts core-level voltage and frequency settings to optimize performance or energy usage. This module supports multiple approaches, including static configurations, reactive schemes, and proactive strategies (e.g., ML-driven control). It also integrates with native Linux governors such as *performance*, *powersave*, and *schedutil*. DVFS control leverages the built-in *Translator* utilities that abstract hardware-specific configurations (e.g., Intel P-states) from user-defined V/f specifications. These default policy templates serve as starting points for experimentation and can be synchronized or combined to implement advanced multi-knob RM techniques.

Preparing for Experimentation

The **Experiment Manager** module serves as the primary interface for initiating experiments in ARDiS. It allows users to configure and execute experiments by specifying the set of applications, selecting appropriate scheduling, mapping, and DVFS policies, and optionally enabling migration to update thread affinities during execution. Prior to launching an experiment, users are required to define a **Global Configuration**. This configuration file includes hardware- and experiment-specific parameters such as the number of physical cores, accessible sensors, available PMCs, default monitoring interval, and DVFS epoch settings.

To facilitate rapid setup, the framework provides an **Automatic Configuration Generator**, which generates a baseline configuration tailored to the capabilities of the target platform. Users may then fine-tune this baseline to suit the needs of specific experiments. This configuration-centric approach ensures consistency across experimental runs and enhances reproducibility and comparability of results.

The ARDiS Workflow

An overview of the ARDiS workflow is illustrated in Fig. 3.5. At the core of this workflow is the **Engine**, the central orchestration component responsible for managing the execution of experiments. The engine is initiated by the **Experiment Manager**, and it coordinates all phases of an experiment based on the parameters defined in the global configuration and the instantiated RM policies, including scheduling, mapping, DVFS, and migration. It also manages supporting functionalities such as benchmark management, binary execution, monitoring, and data reporting. For benchmark-driven workloads, e.g., from PARSEC, SPLASH-2, SPEC, or MiBench, the engine invokes the appropriate benchmark handlers through the **Benchmark Manager** utility. At the beginning of each experiment, the engine launches the **Monitor** module, which is responsible for collecting performance data throughout application execution. Monitoring behavior is defined in the global configuration, including the granularity and specific PMCs to be collected. Metrics include execution time, power consumption, and various CPU/cache events, which are gathered using *perf* [77]. Two modes of data collection are supported:

- *One-shot Monitoring*: *perf* is started once at the beginning of execution and stopped at the end. This mode is suitable for coarse-grained metrics like total energy consumed.
- *Periodic Monitoring*: Metrics are sampled at regular intervals (e.g., per-core, per-process, or system-wide) and saved in a *periodic log*.

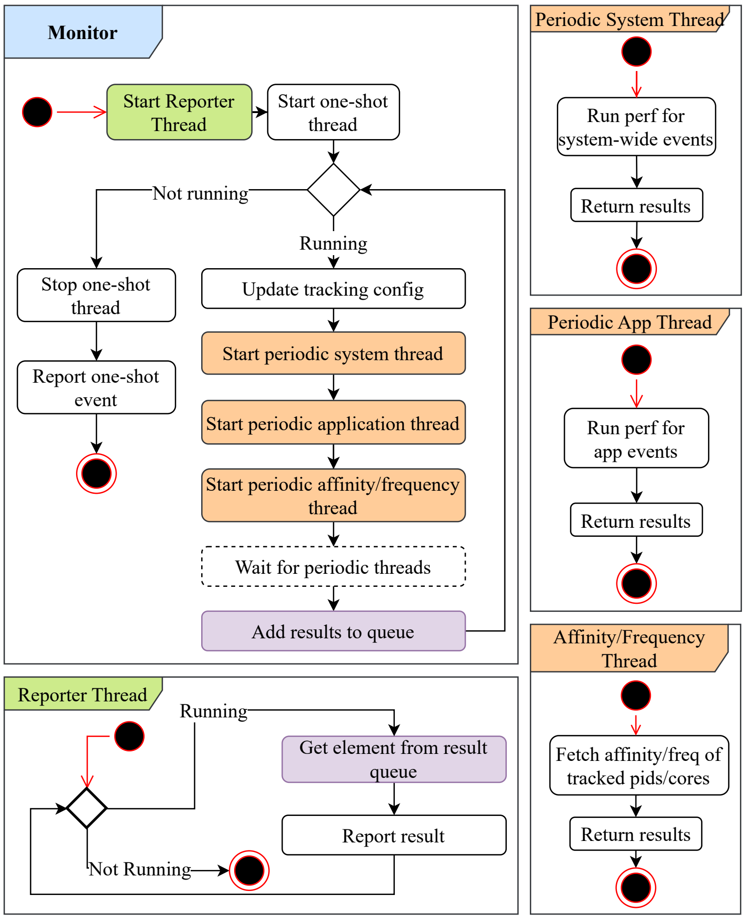


Figure 3.6.: Overview of ARDiS's multi-threaded monitoring system, where periodic and one-shot monitoring threads coordinate to track system-wide events, application-specific metrics, and affinity/frequency updates, ensuring efficient, low-overhead and synchronized data collection at runtime.

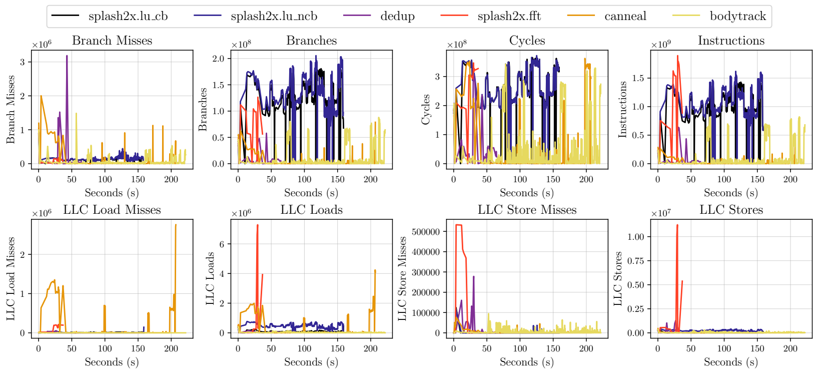


Figure 3.7.: At the end of each experiment, the result analysis library is automatically invoked to post-process the periodic execution logs and generate corresponding plots. This functionality is configurable and, by default, produces a plot for each monitored metric in ARDiS, thereby allowing researchers to conveniently analyze experimental results without additional setup.

Since ARDiS supports concurrent execution of multiple applications and parallel monitoring of various metrics, recording data sequentially can exceed the epoch duration and introduce overhead. To address this, the framework employs parallelization and queuing between the monitoring and **Reporting Engine** components, as shown in Fig. 3.6. This design ensures low-latency and synchronized data handling, contributing to the minimal overhead described in Section 3.3.2.

During runtime, the engine periodically invokes RM policies, collects updated execution metrics, and manages the entire experiment according to the configuration. To support decision-making by each policy module, users can define **Parametrized State/Action Buffers**, which store monitored metrics in memory. This eliminates the need for policies to read from log files during execution, reducing latency. After creating application threads, the engine continuously monitors their execution state, process IDs, core mappings, and V/f levels in a synchronized, thread-safe manner. This ensures safe access to shared runtime variables—such as process IDs, thread-to-core mappings, and core frequency levels—which may be concurrently accessed or modified by multiple policies. Importantly, the engine tracks not only the *intended* decisions of RM policies but also their *actual* effects on the system. For example, if a migration policy intends to remap a thread, or if a DVFS policy

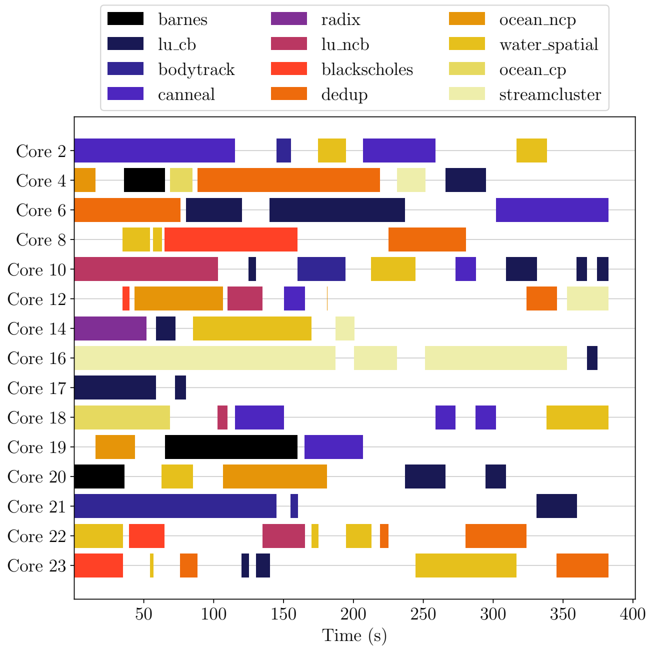


Figure 3.8.: The result analysis library produces plots depicting application-to-core and thread-to-core mappings. This functionality is particularly valuable for verifying the correctness of application-to-core affinity configurations over time, especially when task migration policies are activated in ARDiS.

attempts to adjust a core's frequency, the corresponding monitoring threads verify whether the operating system reflects these changes. This distinction ensures that policy outcomes are measured based on actual system behavior, not just theoretical intent.

Finally, upon completion of the workload, the engine clears the system caches to eliminate residual state from previous runs. This step is essential for ensuring reproducibility in experiments that require repeated execution under identical conditions. The engine also stops the monitor and triggers the post-processing phase, which is detailed in the subsequent subsection.

Post-Processing and the Result Analysis Library

Following a consumer/producer execution model, the **Reporting Engine** saves all experimental traces once the final application in the workload completes its execution. Subsequently, the **Result Analysis Library** performs automated post-processing of the collected data, generating visualizations and plots which are stored in the experiment's result directory.

As illustrated in Fig. 3.7, the post-processing utility automatically—unless explicitly configured otherwise—produces one time-series plot for each monitored *perf* event or performance metric, as well as a corresponding frequency-over-time plot. In addition, it generates an application-to-core and thread-to-core *mapping plot*, which visualizes the placement and migration of applications across the processor's cores over the duration of the experiment. This functionality, shown in Fig. 3.8, is particularly valuable when evaluating experiments that include migration policies, as it allows users to validate that thread affinities are correctly updated during runtime.

Furthermore, depending on the sensing capabilities of the underlying hardware, the post-processor can generate additional power and energy plots. As with other components of ARDiS, the post-processing module is fully configurable. Users may specify which visualizations to generate per experiment and can selectively enable plotting for specific *applications of interest*. This selective plotting is particularly beneficial when working with large workloads, such as those used to evaluate our EffiCast [4] (Chapter 8), as it reduces visual clutter and enhances the interpretability of the results.

4. Machine Learning-Based Prediction of Cache Contention Impacts

The contributions presented in this dissertation employ *supervised* ML to train prediction models at design time, which are then deployed at runtime to predict the impact of potential resource management actions on the system, before effectively applying them. Supervised ML requires labeled datasets, which makes the generation of training data is a crucial step in training such models, as the quality of the generated dataset is largely responsible for the quality of the predictions the model makes at runtime. In the context of this dissertation, supervised learning was chosen over alternative machine learning paradigms due to its practical alignment with the problem formulation and data availability. While reinforcement learning could also optimize resource management actions through interaction with the environment, it introduces significant overhead, complexity, and safety concerns when applied to real hardware or time-sensitive systems[79]. Similarly, unsupervised or self-supervised learning techniques may assist in discovering latent structure or augmenting limited datasets, but they are ill-suited for the direct modeling of action-effect relationships, which require clearly labeled outcomes. In contrast, the methodology adopted in this dissertation enables the collection of concise, phase-aware, labeled datasets that naturally fit the supervised learning paradigm, allowing for efficient training and evaluation of accurate predictive models.

This chapter is dedicated to our contribution, PHASEL [5], which addresses the challenge of generating training data for training lightweight ML models, suitable for system-level RM at runtime. The methodology extracts the phase-level behavior of applications through phase-level *slicing* of their execution traces. The slicing methodology introduced in PHASEL has been leveraged by the contributions in this thesis [1, 2, 3] to generate training datasets that enabled their corresponding ML models to capture the runtime dynamics impacting the cache contention behavior.

4.1. Machine Learning Problem Formulation

Let the state of our target platform at time t, i.e., pre-action state, be denoted as a feature vector $\mathbf{x}(t) \in \mathbb{R}^n$, where each element of $\mathbf{x}(t)$ represents a measurable PMC, e.g., IPC, cache miss rate, energy consumption, etc. Consider a RM action $a \in \mathcal{A}$, e.g., changing the V/f level or migrating a thread between cores. The objective is to learn a predictive function f such that:

$$f : (\mathbf{x}(t), a) \mapsto \mathbf{x}(t + \Delta t) \tag{4.1}$$

This function estimates the future, i.e., post-action system state $\mathbf{x}(t + \Delta t)$ resulting from applying action a in the current system state $\mathbf{x}(t)$, where Δt is a short time interval following the action application. To train such a model, we construct a dataset $\mathcal{D}$ consisting of samples of the form:

$$\mathcal{D} = \left\{ \left(\mathbf{x}_i, a_i, \mathbf{x}'_i \right) \right\}_{i=1}^{N} \tag{4.2}$$

where $\mathbf{x}_i$ represents the system state before action a_i is applied, a_i is the RM action applied during the profiling phase and $\mathbf{x}'_i = \mathbf{x}(t_i + \Delta t)$ is the observed system state after action a_i is applied. This formulation implicitly captures a state transition function:

$$\mathbf{x}' = T(\mathbf{x}, a) \tag{4.3}$$

which characterizes how a RM action a modifies the system state $\mathbf{x}$ in a phase-aware manner.

A key requirement in this formulation is that both the pre-action state $\mathbf{x}$ and the post-action state $\mathbf{x}'$ correspond to the same execution phase of the application. That is, the RM action must be applied and evaluated within a temporally localized region of execution to ensure that the observed state change is attributable to the action itself, rather than to natural phase transitions. To enable comparisons across different RM actions, the runtime conditions at the moment immediately preceding each action must be reproducible, such that each action can be applied to the same pre-action state.

However, reproducing such precise runtime conditions is inherently challenging on real hardware due to uncontrollable sources of variability and the difficulty of deterministic re-execution. In contrast, as will be discussed in this chapter, simulation environments offer the ability to *deterministically* replay execution contexts and apply different RM actions at well-defined points in time. This makes them theoretically suitable for controlled data generation. Nonetheless, this approach introduces two key challenges.

First, there is the problem of defining the set of valid and representative pre- and post-action state transitions to be collected, especially given the diversity of execution phases. Second, in interval simulators such as HotSniper [48] and CoMeT [99], even minimal real-time execution, e.g., one millisecond, can correspond to several minutes of simulation time. As a result, executing a large number of transitions across many application executions becomes computationally prohibitive. The simulation time elevates to *months* when large workloads are considered in multicores, e.g., 64-core simulated AMD Zen 3. In practice, *this limitation renders full-scale transition sampling infeasible in simulation.*

Illustrative Example: Combinatorial Complexity of DVFS Transitions

To illustrate the scale of the data required for fully characterizing the space of resource management actions, consider a representative application with a total execution time of 40 ms, which lies at the lower end of typical trace durations of the considered benchmarks applications in this dissertation, i.e., PARSEC[49] and SPLASH-2[50]. Assuming a RM granularity of 1 ms, the execution trace of the application would be divided into N execution slices:

$$N_{\text{slices}} = \frac{40\,\text{ms}}{1\,\text{ms}} = 40 \quad \text{time slices.} \tag{4.4}$$

For illustration purposes, we consider a typical DVFS policy with frequency levels ranging from 2.4 GHz to 4.0 GHz in 100 MHz increments:

$$\text{Frequency levels} = \{2.4, 2.5, 2.6, \ldots, 4.0\}\ \text{GHz} \tag{4.5}$$

yielding a total of:

$$N_{\text{freqs}} = \frac{4.0 - 2.4}{0.1} + 1 = 17 \quad \text{distinct frequency levels.} \tag{4.6}$$

At each execution slice, any transition between a pair of pre-action and post-action frequencies is theoretically possible. That is, the V/f level of the core(s) on which the application is running could be changed to any of 17 levels at each 1 ms, leading to:

$$N_{\text{transitions}} = N_{\text{freqs}} \times N_{\text{freqs}} = 17 \times 17 = 289 \tag{4.7}$$

unique DVFS transitions per slice. Over the 40 slices of the application, the total number of distinct pre-/post-action samples becomes:

$$N_{\text{samples}} = N_{\text{slices}} \times N_{\text{transitions}} = 40 \times 289 = 11{,}560 \tag{4.8}$$

This example highlights the exponential growth in the number of samples required to exhaustively represent the transition space, even for a single application and a single RM knob, i.e., DVFS. The problem becomes further exacerbated when considering longer traces, additional applications, complex contention scenarios, or multi-dimensional RM actions, i.e., DVFS and task migration, thus reinforcing the *impracticality* of exhaustive simulation-based training data generation.

In the context of system-level RM, actions are typically applied at fine temporal granularities—e.g., 1 ms epochs in Linux [58], where it is reasonable to assume that the application remains in the same phase immediately following an action. PHASEL leverages this observation to introduce a *slicing* methodology that operates on execution traces, generating pre- and post-action samples that preserve phase consistency. This enables the construction of high-quality training datasets that expose complex phase-level behavior to the learning model without the need for exhaustive simulation.

4.1.1. The Generalization Challenge

An essential property of any supervised machine learning model is its ability to *generalize*, i.e., to perform well not only on the training data $\mathcal{D}$, but also on previously unseen system states and contention scenarios. This is particularly critical in the context of RM, where the model must adapt to variations in application behavior, workload mixes, and system dynamics across different execution phases at runtime.

Let $\mathcal{P}$ denote the true, but unknown, distribution over the space of tuples $(\mathbf{x}, a, \mathbf{x}')$. While the training dataset $\mathcal{D}$ provides a finite sample from $\mathcal{P}$, the generalization objective is to minimize the expected prediction error over this full distribution:

$$\mathcal{L}_{\text{gen}}(f) = \mathbb{E}_{(\mathbf{x},a,\mathbf{x}')\sim\mathcal{P}}\left[\ell\left(f(\mathbf{x}, a), \mathbf{x}'\right)\right] \tag{4.9}$$

where $\ell(\cdot, \cdot)$ is a suitable loss function, e.g., mean squared error.

However, since $\mathcal{P}$ is not known explicitly, one typically minimizes the empirical loss over the training set:

$$\mathcal{L}_{\text{emp}}(f) = \frac{1}{N}\sum_{i=1}^{N} \ell\left(f(\mathbf{x}_i, a_i), \mathbf{x}_i'\right) \tag{4.10}$$

The generalization error is then defined as the difference between the expected and empirical losses:

$$\mathcal{E}_{\text{gen}}(f) = \mathcal{L}_{\text{gen}}(f) - \mathcal{L}_{\text{emp}}(f) \tag{4.11}$$

A model with low generalization error can reliably predict the impact of RM actions on new execution phases or system configurations that were not present in the training set. Achieving good generalization requires that the training dataset $\mathcal{D}$:

- covers a sufficiently diverse set of execution phases and system states;
- captures key dynamic behaviors affecting contention behavior and temporal variability;
- reflects the structural relationships between pre-action and post-action system states.

In this dissertation, we argue that our proposed PHASEL [5] phase-level slicing methodology implements all these requirements to enable such generalization. It allows to systematically capture pre- and post-action states at fine-grained phase intervals, leading to a dataset that enhances the model's ability to interpolate and extrapolate across unseen execution scenarios, as demonstrated in the experimental evaluation of the contributions of this dissertation.

4.1.2. Action-Induced and Natural State Transitions

A structuring challenge in this ML problem formulation arises in distinguishing system state transitions that are *causally* induced by the applied RM action, from those that occur *naturally* due to inherent changes in application behavior, e.g., phase transitions, workload fluctuations, consequent contention behavior, etc. If this distinction is not properly accounted for, the predictive model may mix correlation with causation, ultimately affecting its reliability and generalization.

To address this, we define two types of state transitions:

1. **Natural transitions:** $\mathbf{x}(t) \rightarrow \mathbf{x}(t+\Delta t)$, observed when **no RM action is applied**;
2. **Action-induced transitions:** $\mathbf{x}(t) \xrightarrow{a} \mathbf{x}'(t+\Delta t)$, resulting from the application of an RM action a at time t.

To isolate the causal effect of an RM action a, we require that:

$$\mathbb{E}[\mathbf{x}'(t+\Delta t) \mid \mathbf{x}(t), a] \neq \mathbb{E}[\mathbf{x}(t+\Delta t) \mid \mathbf{x}(t)] \tag{4.12}$$

This condition implies that the distribution of post-action states differs significantly from the distribution of naturally evolved states when conditioned on the same pre-action state.

In practice, we ensure this by enforcing that both pre-action and post-action states belong to the same data sample, along with their corresponding action, thereby controlling for temporal evolution within the application. Also, for each training sample in which an action a is applied, a matched control sample is collected in the same phase under identical system conditions, but without applying a. This facilitates counterfactual estimation of natural transitions. These controls ensure that the training dataset accurately reflects the causal impact of RM actions, allowing the learning model to distinguish between *natural* and *action-induced* transitions.

4.2. Training Data Generation Methodology

We introduce **PH**ase-level **A**pplication **S**ensitivity **E**xtraction and **L**earning (PHASEL), a non-intrusive methodology for extracting and learning phase-level application behavior to different hardware and runtime configurations. Prior work have typically employed source code instrumentation techniques [119, 118, 120, 121, 122] to enable the comparison between execution phases across runtime configurations. On the other hand, PHASEL operates directly on binary executables using a slicing technique, based solely on performance counter readings, without requiring source code access. An overview of PHASEL is illustrated in Fig. 4.1 and Fig. 4.2.

Our approach starts by profiling applications across various configurations, e.g., thread-to-core mappings, V/f levels, etc., followed by phase identification and alignment to enable the comparison of slices across different configurations. Our methodology is validated on the two target platforms introduced in Chapter 3, i.e., the simulated 64-core homogeneous AMD Zen 3 clustered processor and the 24-core heterogeneous Intel Alder Lake clustered processor. The knowledge extracted from this data is then used to train ML models, as will be demonstrated in the different contributions in this thesis.

Application Profiling The first step in PHASEL is to comprehensively profile applications across various configurations on the target platform. The goal of this phase depends on the ML problem formulation. For instance, to train a model that predicts the impact of task migration decisions on system performance, applications should be profiled across different configurations of application-to-cluster and thread-to-core mappings. If the goal is to predict the impact of DVFS decisions, different configurations of cluster-level or core-level V/f settings should also be considered. Similarly, for joint RM decisions, the profiling should consider combined configurations, e.g., thread-to-core mappings while sweeping different V/f levels. To explore the impact of cache contention contention, different numbers of background applications should be mapped on the system. The ultimate goal is **to collect execution traces that are representative of the runtime conditions the model will exposed to**.

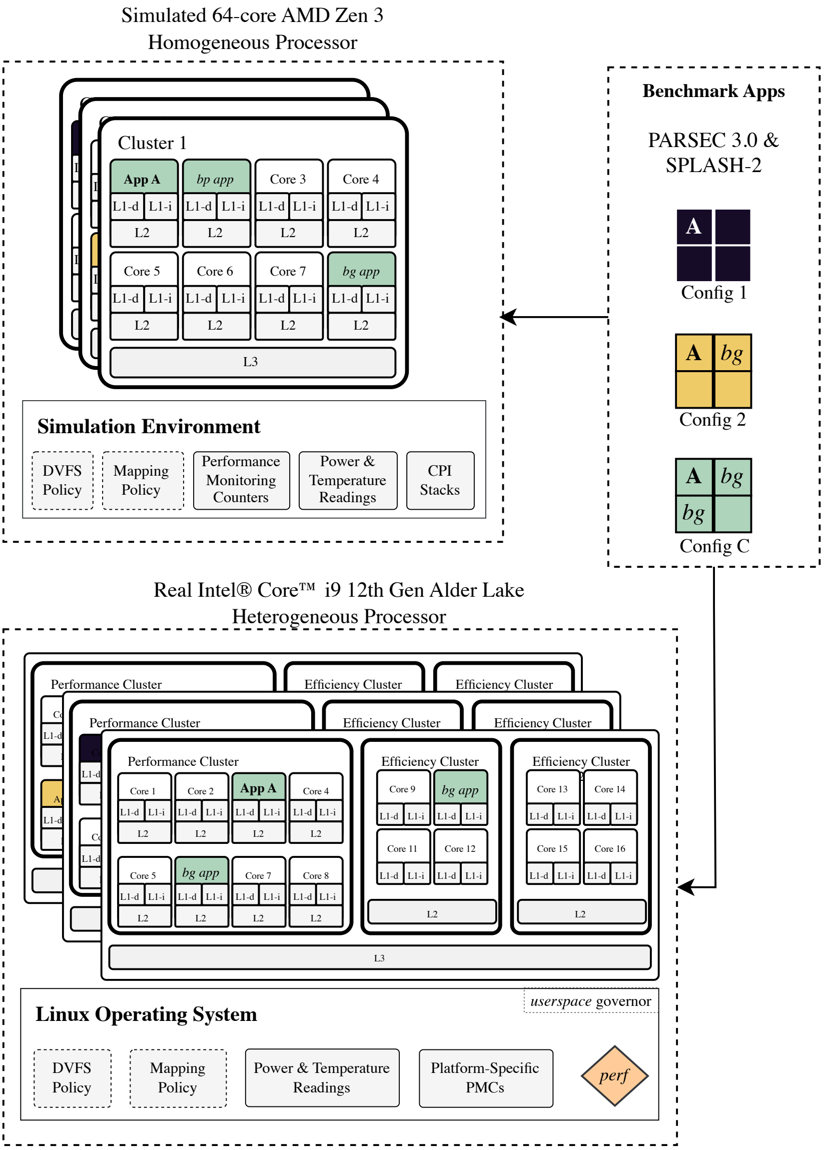
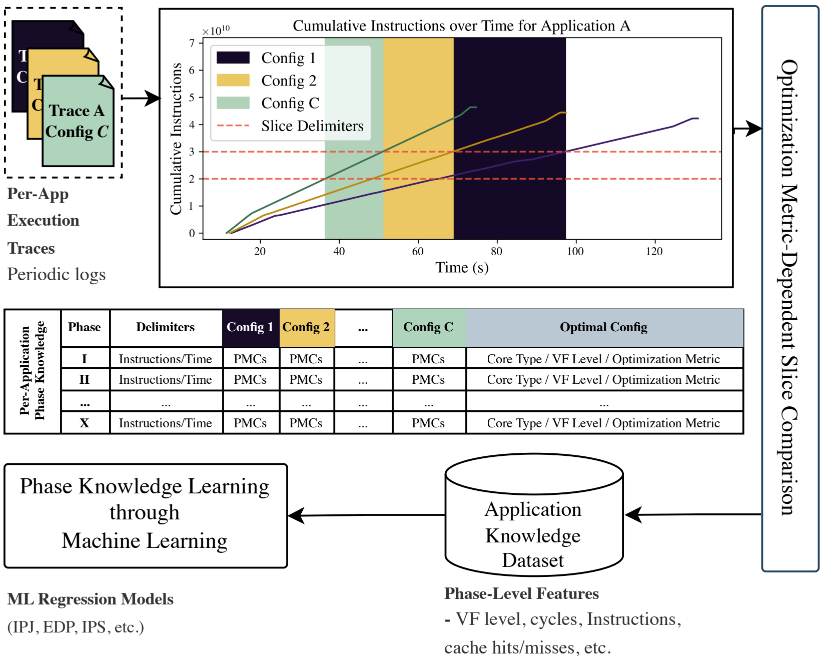


Figure 4.1.: Our methodology starts by executing the different benchmark applications in multiples representative configurations and collecting their periodic execution traces, which serve as input to the slicing process.


Trace A
Config C
Per-App
Execution
Traces
Periodic logs
Cumulative Instructions over Time for Application A
×10^10
Cumulative Instructions
Time (s)
Config 1
Config 2
Config C
Slice Delimiters
Optimization Metric-Dependent Slice Comparison
Per-Application Phase Knowledge
Phase Knowledge Learning through Machine Learning
Application Knowledge Dataset
ML Regression Models
(IPJ, EDP, IPS, etc.)
Phase-Level Features
- VF level, cycles, Instructions,
cache hits/misses, etc.

Phase	Delimiters	Config 1	Config 2	...	Config C	Optimal Config
I	Instructions/Time	PMCs	PMCs	...	PMCs	Core Type / VF Level / Optimization Metric
II	Instructions/Time	PMCs	PMCs	...	PMCs	Core Type / VF Level / Optimization Metric
...	...	...	...	...	...	...
X	Instructions/Time	PMCs	PMCs	...	PMCs	Core Type / VF Level / Optimization Metric

Figure 4.2.: The workflow of our slicing methodology for extracting phase-level knowledge about applications across different runtime configurations. The extracted knowledge is readily learnable by different ML models, which enable our RM models to perform system-level optimization at runtime.

Trace Data Collection In each executed experiment, the system and application behaviors are monitored periodically, collecting PMCs, at a granularity that is similar to the intended RM periodicity, e.g., 1 ms for DVFS. The mechanism to record the PMCs depends on the target platform. HotSniper[48] and CoMeT[99] support the recording of a wide variety of PMCs, including CPI stacks[123], memory-related metrics, e.g., cache accesses and misses, memory reads and writes, etc., branches, branch misses, retired instructions, instructions types (float and integer operations), etc. For power and temperature data, McPat[105] and Hotspot[100] are employed, respectively. For real hardware platforms, *perf*[77] can be used as the back-end monitoring tool, to periodically monitor the hardware and software events exposed by the target hardware and the running operating system, as described in Section 3.3. Power data can be gathered directly via sensors or estimated through interfaces like Running Average Power Limit (RAPL), while temperature data can be read through the sensors available on the target hardware. At the end of this phase, each application is profiled across the different configurations, and the periodic performance, power and temperature characteristics are collected.

Phase Identification and Alignment The next step in PHASEL identifies and aligns similar pre-action execution phases across different configurations. The goal is to form a collection of training samples, where each sample consists of a pre-action state, an action, i.e., the configuration change, and the resulting post-action state. The action is optional, as to cover the distribution of natural phase transitions in the execution of applications, as discussed in Section 4.1. Since the number of retired instructions remains *constant* for a given application, PHASEL aligns phases based on instruction counts. In this **instruction-based slicing**, an execution trace is divided into fixed slices of I instructions. We slide a window of I instructions over the entire execution trace in one configuration, defining each slice by its start and end instruction, $[I_{start}, I_{end}]$, and extract performance counters within this window. As performance data is monitored periodically and is timestamped, we first plot the cumulative retired instructions over time. This step allows the identification of the time step corresponding to the execution of each instruction within the execution of the application. With each slice now identified with time and instruction boundaries, it is possible to extract the metrics for each slice using linear interpolation. At this stage, each slice corresponds to a pre-action state that is labeled with a pre-action configuration. Using the same delimiting instructions $[I_{start}, I_{end}]$, all profiled post-action states can be extracted.

4.3. Illustrative Example

The following illustrative example demonstrates the benefits of our PHASEL slicing methodology to unveil counter-intuitive trends regarding the sensitivity of different applications to the heterogeneity of the processor in experimental setup (C) (Section 3.1.1). The machine has performance (P) and

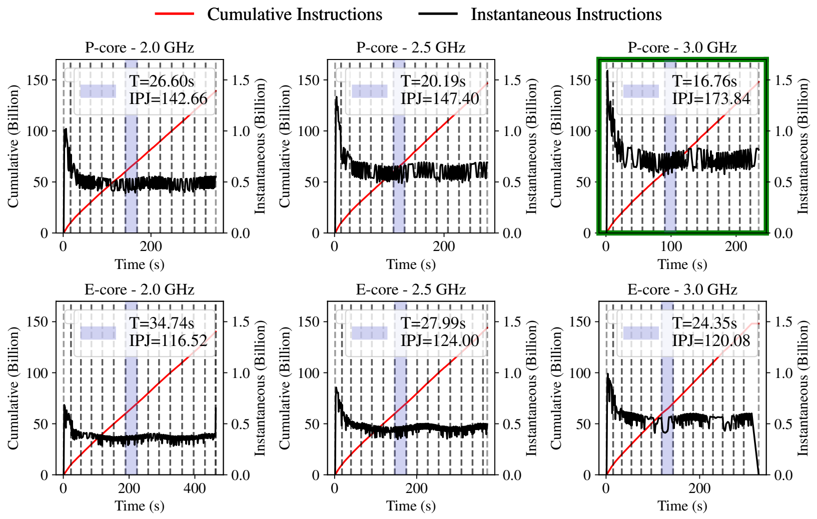


Figure 4.3.: *fluidanimate* - 10-billion-instruction slices Instruction slicing for *fluidanimate* confirms that the most energy-efficient execution can be achieved on a P core at the highest VF level.

efficiency (E) cores. As will be discussed in detail in Chapter 8, contrary to intuition, the most energy-efficient execution can be achieved on P-cores rather than on E-cores. Moreover, the most energy-efficient execution is always achieved when the cores are running at the highest V/f level. Consequently, from a resource management perspective, maximizing energy efficiency on such platforms is trivial: map application to P-cores and boost them to the highest V/f level[1]. To investigate this, we perform slicing on the execution traces of applications at a fine granularity using our methodology, as shown in Fig. 4.3 and Fig. 4.4. In each figure, one slice is highlighted—in blue, for comparison across the 6 different configurations of V/f levels and core types, and its achieved performance and energy efficiency are shown in the legend of the corresponding subfigure. Although the highlighted slices are of equal

[1] We demonstrate in Chapter 8 that, due to cache contention, this statement is invalidated, and such runtime decisions become too complex and non trivial.

instruction window sizes, their execution times vary significantly across executions, due to the different achieved IPS with each configuration.

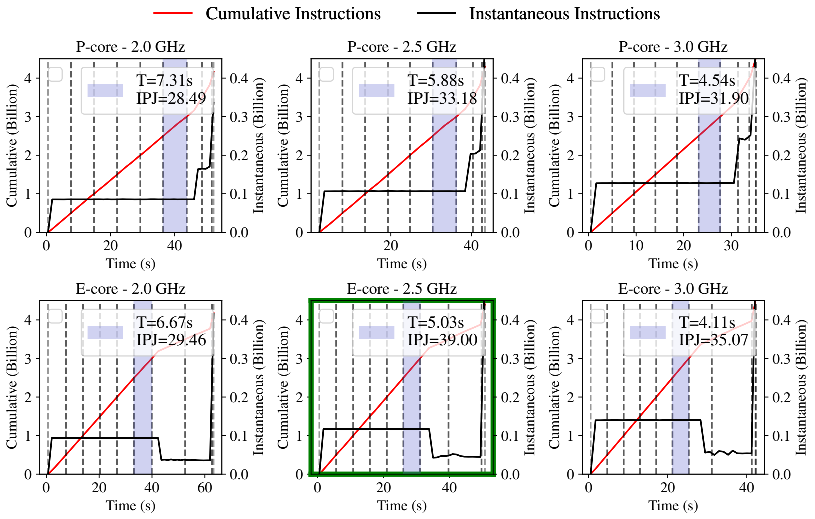


Figure 4.4.: *radix* - 500-million-instruction slices Instruction slicing for *radix* reveals trends that contradict initial observations where P cores at the highest VF level delivered the most energy-efficient execution. Instead, radix's slice is most energy-efficient on an E-core at a medium VF level.

In Fig. 4.3, we slice the execution of *fluidanimate*, with a 10-billion-instruction slice window. The highlighted slice confirms that the most energy-efficient execution is achieved on a P-core at the highest V/f level of 3.0 GHz, which is aligned with the previous statement. In Fig. 4.4, we slice the execution of *radix*, with a finer-grained window size of 500 million instructions. This experiment showed that *radix* exhibits a different sensitivity compared to *fluidanimate*. In fact, the most energy-efficient execution of the highlighted slice was not achieved on a P-core at the highest frequency, but rather on an E-core at 2.5 GHz. This emphasizes *the importance of selecting the appropriate slicing granularity to uncover energy efficiency gains*. The same experiment is conducted for all applications in the different benchmark suites, and highlights that applications exhibit different sensitivity to core types and V/f level throughout their execution.

It is important to note that PHASEL only provides the core slicing mechanism for analysis and training data generation. As will be demonstrated across this dissertation, the exact structure of the generated dataset is adapted to the problem formulations in each individual contribution, and will be detailed in their corresponding chapters.

5. Thermal Optimization using ML-Based Contention-Aware Task Mapping and DVFS

This chapter presents *TcRM* [1], the first work in the literature to consider the trade-off between cache contention and temperature on clustered multicores through a novel thermal- and cache-aware RM technique, by means of task mapping and DVFS. It aims at minimization the overall chip temperature under performance constraints of running applications.

As discussed in Section 2.4, recent RM strategies, e.g., [89, 65], have started to reduce cluster-level cache contention through application mapping. These approaches often co-map memory-bound and compute-bound applications on the same cluster to reduce pressure on the LLC, as compute-intensive tasks typically impose less demand on shared caches. Nevertheless, this strategy has the side effect of increasing chip temperatures, given that compute-heavy workloads tend to consume more power than memory-intensive ones. Due to the heat dissipation between simultaneously-operating cores in multi-core processors, thermal hotspots can form, which jeopardizes chip reliability and long-term functionality [17]. The trade-off between reducing cache contention and controlling temperature is illustrated in the motivational example in Section 1.2.1 and Fig. 1.3, and *has not been observed in the literature.*

5.1. Challenges and Novel Contributions

To realize the objective of minimizing temperature under performance constraints at runtime, two principal challenges must be addressed:

First, it is essential to estimate, at application arrival time, the slowdown resulting from cache contention between potentially co-mapped applications.

As discussed in Challenge 1 (Section 1.3.1) and 4 (Section 1.3.4), addressing this at design time through exhaustive profiling is impractical due to two main reasons.

1. The combinatorial explosion in the number of possible application pairings renders comprehensive profiling infeasible.
2. Cache contention is highly dynamic, varying with the memory intensity of applications during their overlapping execution intervals—a factor that cannot be predicted in open systems [53]. Furthermore, contention arises from various low-level mechanisms, including limited cache capacity and bandwidth, which are affected differently by diverse applications and have heterogeneous impacts on their execution slowdown.

As a result, the analytical modeling of such complex interactions becomes intractable. To overcome this, we adopt a machine learning-based solution using a NN model, which learns complex contention patterns from a limited training dataset at design time and generalizes to new application behaviors at runtime.

Second, determining the appropriate V/f level that compensates for the contention-induced performance degradation while simultaneously minimizing thermal dissipation is non-trivial. We address this by including the V/f level as an input feature to the NN model. This enables *TcRM* to *jointly* determine both the application-to-cluster assignments and the V/f levels that satisfy performance constraints despite contention, while reducing chip temperature. Additionally, *TcRM* refines resource management by selecting thread-to-core mappings, further leveraging thermal optimization opportunities.

In summary, our novel contributions are:

- We design, train, and employ an NN model to predict the slowdown in the application execution induced by cache contention between co-mapped applications in a cluster at a given V/f level.
- We introduce a thermal- and contention-aware resource management technique to determine application-to-cluster and thread-to-core mapping, as well as the required V/f levels of the clusters that compensate for the contention-induced slowdown predicted by the NN model, such that the performance constraints of all applications are satisfied, while minimizing temperature.

5.2. Problem Formulation

Based on the system model and mathematical notations defined in Section 3.1, the objective of *TcRM* is to find for each arriving application the application-to-cluster and thread-to-core mapping, and to select the V/f levels of all clusters so that the peak temperature of the chip $\max T$ is minimized while the performance constraints of all applications are satisfied, considering the cache-contention-induced slowdown. Mathematically, we can express this problem as finding matrices G, V, and f_c $\forall c$, in order to:

$$\text{Minimize } \max T \text{ s.t.: } R_k \leq \hat{R}_k \forall k$$

Our proposed approach to solve this problem is illustrated in Fig. 5.1. At design time, we train an NN model (Section 5.3) on some application scenarios at different V/f levels. At runtime, our proposed runtime resource management technique (Section 5.4) employs this model to predict contention-induced slowdowns in application executions even for unseen scenarios, and uses this prediction to select application-to-cluster, thread-to-core mappings, and the V/f levels so that the aforementioned goal/constraint are satisfied.

5.3. Slowdown Prediction Model

As discussed in Section 1.3, predicting the slowdown induced by cache contention remains a complex task. To address this, *TcRM* utilizes a an NN model specifically designed to estimate the slowdown experienced by co-mapped applications due to cache contention. The following subsections describe the methodology used to generate the training data, the selection of relevant features, and the architecture of the predictive model.

5.3.1. Training Data Generation

The upper part of Fig. 5.1 outlines the procedure used for generating training data, which is based on the methodology proposed in PHASEL [5] and detailed in Chapter 4. The initial step involves identifying representative execution scenarios from which trace data can be collected to extract the relevant

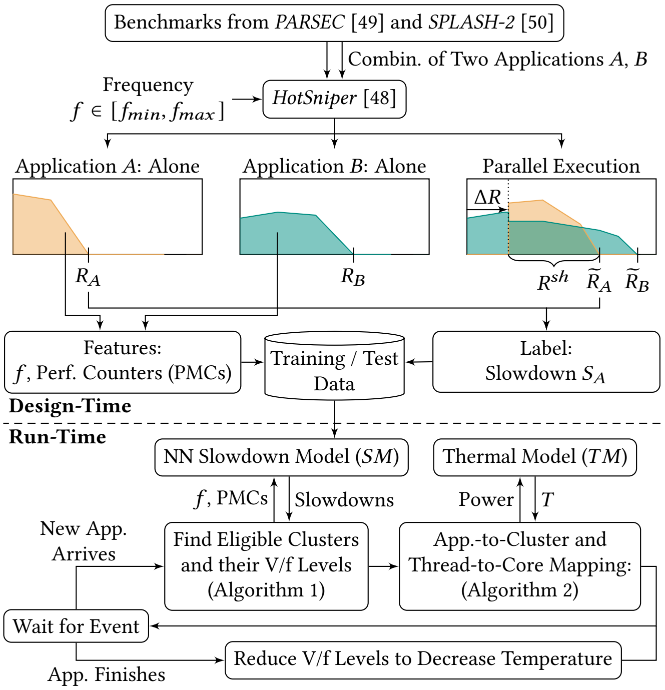


Figure 5.1.: *TcRM* flow. During design time, traces are collected and training data is generated following the our PHASE methodology [5] to train the predictive NN model. At runtime, application-to-cluster mapping and corresponding V/f levels are determined to fulfill the objectives of *TcRM*.

model features. Due to the exponential number of potential application combinations, we restrict our analysis to pairs of applications, denoted as A and B. As illustrated in Fig. 5.1, the difference in arrival times, ΔR, determines the shared execution interval R^{sh} of the co-executing applications. Varying ΔR values result in distinct cache contention patterns, as shown in Table 1.1. For training purposes, we select two representative ΔR values. Crucially, our experimental evaluation confirms that the trained model generalizes well to arbitrary application counts and ΔR values, since their effects on contention are encapsulated within the selected features.

In addition to these *parallel execution* scenarios, we also conduct *single execution* runs, in which each application is executed alone on a cluster. These are essential for both feature extraction and calculating the target slowdown metric used for training labels. All selected execution cases are evaluated across a range of V/f levels, which are also included as input features for the model.

5.3.2. Feature Selection and Model Architecture

The model features comprise a subset of cache and memory-related PMCs collected from the single execution traces of applications, specifically from the segments that overlap with the shared execution window R^{sh} in the parallel execution. To identify a compact yet effective subset of PMCs that accurately reflects cache contention behavior and minimizes prediction error, we apply two techniques: Lasso Regression to rank features by their predictive utility for S_A, and the Pearson correlation matrix to detect feature interdependencies. Following extensive analysis of trace data, we selected four features: LLC accesses, LLC misses, DRAM reads, and DRAM accesses. Consequently, each row in the training dataset includes the 4-tuple of PMCs for application A and the corresponding 4-tuple for application B.

The model's prediction target (label) is the slowdown experienced by application A due to co-mapping with application B, computed as:

$$S_A = \frac{\widetilde{R}_A}{R_A} - 1$$

Here, R_A represents the standalone response time of application A, while $\widetilde{R}_A$ denotes its response time when co-executed with another application.

While training focuses on scenarios with a single co-runner B, the model remains applicable to cases involving multiple co-mapped applications. In such cases, the PMCs corresponding to B are accumulated over all co-runners to represent aggregate contention effects. We employ a lightweight NN model architecture consisting of four fully connected hidden layers, with 64, 32, 16, and 8 neurons, respectively. Each hidden layer uses ReLU activation, and the output layer consists of a single neuron with linear activation. This model achieves a root-mean-square error (RMSE) of 2% and a 0.99-quantile prediction error of $\varepsilon = 5\%$. Moreover, the runtime inference overhead on our target platform is minimal, averaging only 3 µs.

Algorithm 1 Find Eligible Clusters and their V/f Levels

```
Input: arriving app. $A$, slowdown model $SM$ with error $\varepsilon$
Output: set of eligible clusters $C^E$ and V/f levels
  $C^E \leftarrow \{c : \forall k\ g_{c,k} = 0, f_{min}\}$                ▷ completely free clusters
  if $C^E \neq \{\}$ then return
  for each $c \in \{1, \ldots, C\}$ do                                     ▷ iterate over clusters
      $X \leftarrow \{i : q_{i,c} \wedge \forall k\ v_{i,k} = 0\}$        ▷ free cores on cluster $c$
      if $|X| < h_A$ then continue                                      ▷ not enough free cores on $c$
      $f \leftarrow$ BinarySearch($[f_{min}, f_{max}], \lambda f.$Satisfies$(f, c)$)
      $C^E \leftarrow C^E \cup \{(c, f)\}$
  procedure Satisfies($f, c$)                                           ▷ deadline violation check
      $\Gamma \leftarrow \{A\} \cup \{B_1, \ldots, B_Z\}$                 ▷ set of $A$ and other apps
      for each $\gamma \in \Gamma$ do                                   ▷ violation check for $\gamma$
          $R^{sh} \leftarrow$ shared time of $\gamma$ and other apps $\Gamma \setminus \{\gamma\}$
          $S_\gamma \leftarrow SM(f, \mathrm{PMC}(\gamma, R^{sh}), \sum_{\omega \in \Gamma \setminus \{\gamma\}} \mathrm{PMC}(\omega, R^{sh}))$
          $R'_\gamma \leftarrow (S_\gamma + 1 + \varepsilon) \cdot R^{sh} + (R_\gamma - R^{sh})$
          if $R'_\gamma > \hat{R}_\gamma$ then return false
      return true
```

5.4. Contention-Aware Mapping

As shown in the bottom part of Fig. 5.1, *TcRM* takes runtime decisions at two events; when a new application arrives in the system, and when an application finishes its execution. Upon arrival of an application A, *TcRM* executes two steps:

1. Finding *eligible* clusters for hosting application A and obtaining the minimum V/f level that satisfies the performance constraints of the application A and the already-running applications B on the cluster.
2. Selecting the application-to-cluster and thread-to-core mapping that minimizes the temperature.

Finding Eligible Clusters The first step is illustrated in Algorithm 1. First, *TcRM* looks for free clusters on the chip. If found, they will be passed to the second step. Otherwise, the search for eligible clusters starts. An eligible cluster for hosting application A must have enough free cores to execute it, and there must be at least one V/f level, f_c, that can compensate for the predicted slowdowns of all potential co-mapped applications on that cluster, denoted as Γ.

To find f_c, a binary search is performed. At each iteration, the slowdown suffered by each application, $\gamma \in \Gamma$, is estimated, considering the tested V/f level in that iteration. To do that, we pass to the NN model the relevant PMCs of the application γ, and the accumulated PMCs of all other applications in $\Gamma \setminus \{\gamma\}$. The PMCs are extracted from the parts of the single execution traces that correspond to the shared period R^{sh}. Considering the predicted slowdown, the response time of γ, i.e., R'_γ, is estimated. To account for errors in the NN predictions, the 0.99-quantile of the model error ε is added to slowdown predictions, to avoid underestimations which may result in performance constraint violations. If $\hat{R}_\gamma$ of each application is satisfied on cluster c at the current frequency f_c, then, (c,f) is added to the set of eligible clusters C^E with their corresponding frequencies.

Minimizing Temperature In the second step, illustrated in Algorithm 2, the goal is to find a hosting cluster c_{opt} and the set of cores $M_{c_{\text{opt}}}$ for application A, so that the peak steady-state temperature $\max T$ on the chip is minimized. Testing all thread-to-core mapping combinations has an exponential complexity. Instead, we employ a greedy search with a reduced complexity, that traverses all eligible clusters to select the mapping that reduces the temperature. On each eligible cluster, the thread mapping to each free core is tested, and the estimated peak chip temperature is stored. After testing all thread-to-core mapping options on a cluster, we select the mapping that results in the lowest $\max T$. The best thread-to-core mapping M_c on this cluster and its corresponding estimated T_c are stored. Finally, we select c_{opt}, the eligible cluster with the lowest $T_{c_{\text{opt}}}$, and we map the application threads to $M_{c_{\text{opt}}}$.

Upon departure of an application hosted by a cluster c, f_c is reset to the minimum V/f level that satisfies the performance constraint of the remaining applications.

Algorithm 2 App-To-Cluster and Thread-to-Core Mapping

```
Input: eligible clusters $C^E$, arriving app. $A$
Output: mapping and corresponding frequency of app. $A$
  for each $(c, f) \in C^E$ do
      $P^o \leftarrow \{1, \ldots, c-1, c+1, \ldots, C\}$          ▷ powers of other clusters
      $X \leftarrow \{i : q_{i,c} \wedge \forall k\ v_{i,k} = 0\}$          ▷ free cores on cluster $c$
      $P^B \leftarrow$ power of already running applications on $c$ at $f$
      $M_c \leftarrow \{\}$          ▷ start with empty mapping on $c$
      while $|M_c| < h_A$ do
          for each core $i \in X$ do
              $P \leftarrow P^o + P^B + P^A(M_c \cup \{i\}, f)$          ▷ full chip power
              $T \leftarrow TM(P)$          ▷ steady-state temperature with $p$
              $\hat{T}_i \leftarrow \max T$          ▷ peak temperature when using core $i$
          $i_{\text{opt}} \leftarrow \arg\min_i \hat{T}_i$          ▷ select core with min. peak temp.
          $M_c \leftarrow M_c \cup \{i_{\text{opt}}\}$          ▷ add core to mapping
          $X \leftarrow X \setminus \{i_{\text{opt}}\}$          ▷ core is not available any more
          if $|M_c| = h_A$ then
              $\hat{T}_c \leftarrow \min_i \hat{T}_i$          ▷ temp. when mapping on cluster $c$
              $f_c' \leftarrow f$          ▷ remember frequency for cluster $c$
  $c_{\text{opt}} \leftarrow \arg\min_c \hat{T}_c$          ▷ select cluster with min. peak temperature
  $g_{c_{\text{opt}},A} \leftarrow 1,\ v_{i,k} \leftarrow 1\ \forall i \in M_{c_{\text{opt}}},\ f_{c_{\text{opt}}} \leftarrow f_c'$          ▷ map $A$, set V/f level
```

Illustrative example

To illustrate the work of *TcRM*, we run an experiment on one cluster only, where two instances of *cholesky* are co-mapped with *fmm* at different execution times. We consider that an application can meet its performance constraint while running alone on the cluster at 2.4 GHz. Figure 5.2 shows the arrival times of the three applications, the LLC accesses and misses of *fmm* and the selected frequencies by *TcRM*.

At first, *fmm* is running alone on the cluster at 2.4 GHz. Then, the first instance of *cholesky* arrives, while *fmm* is heavily accessing the LLC with a high miss rate. As a response, *TcRM* chooses 3 GHz as the minimum frequency that will meet the performance constraints of both applications. At t=104 ms, *cholesky* finishes its execution and *TcRM* reduces the frequency of the cluster to 2.4 GHz.

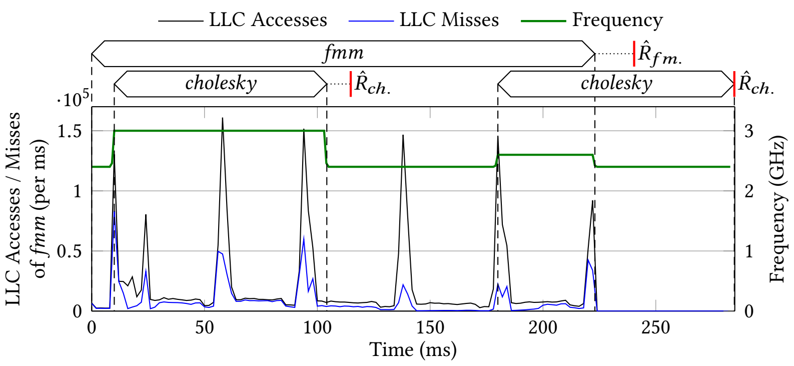


Figure 5.2.: An example illustrating the runtime actions of our resource management techniques. The DVFS decisions of our *TcRM* depend on the execution phases of applications and the resulting contention behavior on the cluster.

After a while, a new instance of *cholesky* arrives, but this time, *fmm* is accessing the LLC less heavily. This time, *TcRM* chooses alower V/f level of 2.6 GHz. At t=223 ms, *fmm* finishes its execution, and the cluster is throttled down to 2.4 GHz, since *cholesky* is running alone.

This example demonstrates how *TcRM* selects the required minimum V/f level to meet the performance constraints of all applications, while considering the variation in cache-contention behavior throughout execution time.

5.5. Experimental Evaluation

To evaluate *TcRM*, we run experiment on the *HotSniper* [48] simulator (detailed in Section 3.2.1), with the clustered 64-core homogeneous processor in experimental setup (A) in Section 3.1.1. The experiments are using benchmarks that show cache-contention behavior at the design profiling phase, to guarantee real testing for our prediction model. These applications include: *bodytrack*, *streamcluster*, *x264*, *cholesky*, *raytrace*, *radix*, *fmm*, *fft*, *lu.cont*, *lu.ncont*, *water.nsq* and *water.sp*, each with 4 parallel threads[1]. We use a mixed workload that combines 20 randomly-selected applications from this set of

[1] Detailed descriptions of each application are included in Appendix A

benchmarks, with medium and large input sizes. The arrival times of the applications are sampled from a Poisson distribution with varying arrival rates, to consider various system utilization values and importantly, to allow testing our technique for unseen scenarios. The performance constraint of an application is defined as its *alone* execution time at 2.4 GHz.

5.5.1. Comparison Technique

Our *TcRM* is the first to *jointly* consider the temperature increase problem and the cache-contention one, while the state of the art tackles each problem separately. Hence, we cannot compare *TcRM* directly against any state-of-the-art technique. We therefore develop a baseline, *CoRM*, as a combination of the two state-of-the-art works: the thermal-aware application mapping proposed in [83] and the contention-aware V/f level selection proposed in [54]. At runtime, when a new application arrives, *CoRM* first selects the cluster with the lowest average temperature of all its cores, and then finds the task-to-core mapping that reduces the temperature, using the approach in [83]. This is done by positioning idle cores between active ones. Moreover, temperature will be reduced further, as the NN-based approach in [54] selects the minimum V/f level of the cluster that satisfies performance constraints of the co-mapped applications, based on the current cluster frequency and readings from the PMCs. Their approach also selects the maximum V/f level as the starting operating point for the applications, then calls the model to select the minimum V/f level that satisfies the constraints. The model will not be called again during the application execution, unless a violation of performance constraint is predicted.

5.5.2. Experimental Results

We compare *TcRM* and *CoRM* in terms of the response times of the applications relative to their performance constraints and the resulting temperatures.

Performance Slack Fig. 5.3 shows that both techniques satisfy all performance constraints. However, *CoRM* results in a significant slack time, up to 38%. These results are explained by the selected frequencies by both techniques illustrated in Fig. 5.4. *CoRM* always selects higher frequencies, since it

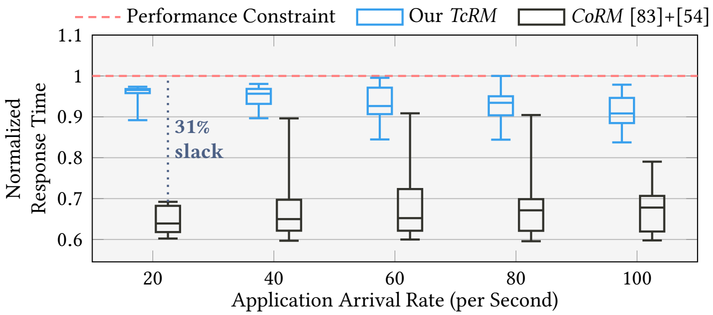


Figure 5.3.: *TcRM* minimizes excessive slack in application execution times to effectively reduce temperature. As a result, all applications complete execution near their respective deadlines. In comparison, *CoRM* exhibits substantial slack.

starts the application execution at f_{max} as mentioned in [54]. Then, the model is called based on the current performance counter readings, which reflect only the instantaneous cache-contention behavior of the running applications, but this might vary throughout the execution, as shown in Fig. 5.2.

In contrast, our *TcRM* considers the cache-contention behavior during the whole shared execution time of the co-mapped applications. This allows our model to provide a more accurate prediction of the slowdown, thus selecting lower V/f levels, ultimately yielding minimal slack times.

Temperature Minimization Figure 5.5 shows the resulting temperatures in our comparison experiments. *TcRM* reduces the temperature by 30% on average compared to *CoRM*. There are two reasons. First, *TcRM* selects lower V/f levels as shown in Fig. 5.4. Second, our mapping Algorithm 2 tests much more mapping possibilities compared to the ones tested by the mapping policy [83] used in *CoRM*. Specifically, *CoRM* only selects the coldest cluster, while *TcRM* first estimates the resulting temperatures of mapping the application to each available cluster, then selects the one that results in the minimum temperature.

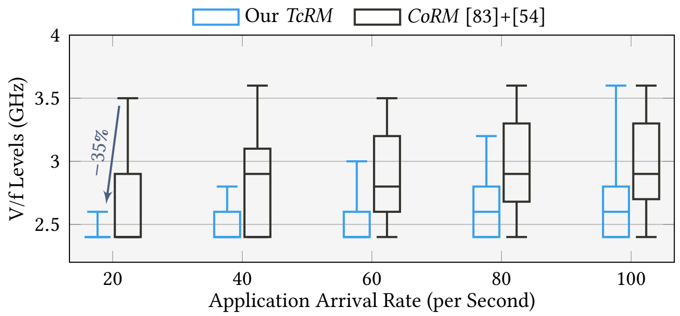


Figure 5.4.: *TcRM* sets cores to significantly lower V/f levels compared to *CoRM*, thereby leading to larger reduction in the overall chip temperature.

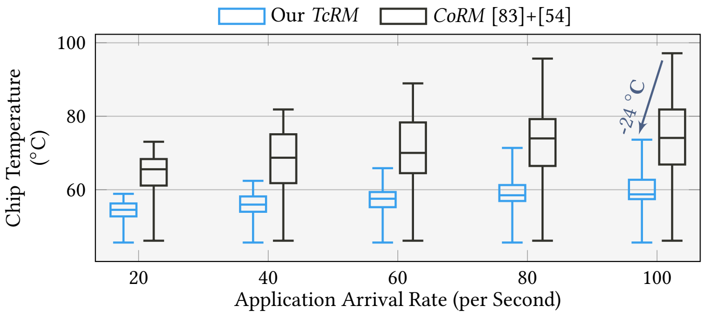


Figure 5.5.: *TcRM* significantly reduces the chip temperature over *CoRM*, thanks to the highly-accurate slowdown predictions of our trained NN network.

5.5.3. Runtime Overhead

The execution overhead of *TcRM* depends on the utilization of the chip as new applications arrive. We measure the overhead induced by *TcRM* in the extreme scenario, i.e., when none of the clusters is neither free nor fully utilized. In this scenario, *TcRM* has to traverse all clusters to find the eligible ones. This requires a binary search to find the minimum V/f level that satisfies the performance constraints of the co-mapped applications. This scenario is the worst-case scenario for our technique, and yields an overhead of 1% of the average execution time of the applications.

5.6. Summary

This chapter presented our *TcRM* contribution, which has investigated, for the first time in the literature, the trade-off between temperature and cache contention that arises on clustered multi-core processors.

Tackled Challenges In this work, the challenges in Section 1.3 were first identified and then successfully tackled. The most prominent challenge was the first one in Section 1.3.1, about the complexity of predicting the cache contention-induced slowdown. Tackling this challenge necessitated a robust formulation of the machine learning problem, along with the conception of our training data generation methodology, PHASEL [5], which we presented in Chapter 4. Moreover, the fact that ML was used to predict the cache contention-induced slowdown necessitated that the trained models are very lightweight, to be suitable for runtime management, fitting within the 1 ms DVFS epoch.

Achievements and Takeaways We exploited the observed trade-off by determining the application-to-cluster, thread-to-core mapping and the V/f levels of the clusters, in order to minimize the overall chip temperature while satisfying the performance constraints of applications. To capture the complex characteristics of shared resource contention, we trained a NN model at design time that can predict the slowdown of co-mapped applications to clusters at runtime. The high accuracy of our trained model enabled *TcRM* to meet the performance constraints of all applications in our evaluation experiments, while significantly reducing the chip temperature compared to the state of the art by 30% on average, confirming the significance of jointly considering contention and temperature problems, as motivated in Chapter 1.

Insights Static application-to-cluster and thread-to-core mapping at the beginning of the execution can miss opportunities that arise during the application's runtime, as it changes phases and as the contention landscape evolves. Migrating the application to another cluster, or the threads to different cores, could have led to further reductions in temperature. This migration knob and its potential benefits are therefore explored in the next chapter, through our proposed *SmartCM* technique.

6. Performance Optimization using ML-Based Contention-Aware Task Migration and DVFS

This chapter presents *SmartCM* [2], the first technique that mitigates cache contention under thermal constraints in clustered manycores by means of task migration and cluster-level DVFS. The complexity of shared cache contention was solved in this contribution with a lightweight neural network that was trained at design time, and successfully generalized to unseen contention scenarios at runtime.

As highlighted in Section 1.3, in addition to the cache contention problem, clustered multi-core processors are also subject to the challenge of *temperature interference*, wherein heat propagates between cores executing concurrently, potentially even across cluster boundaries. When the temperature rises above safe operational thresholds, the TCC—commonly implemented in commercial processors [52, 124]—is activated. This mechanism lowers the V/f levels of *all* cores, regardless of which one caused the thermal violation, in an effort to cool the chip. Such aggressive downscaling results in degraded performance for all executing applications. Nevertheless, as discussed in Section 2.4, existing approaches to performance optimization in the literature tend to target either cache contention or thermal interference in isolation. Such a *unidimensional* focus is inherently *sub-optimal*. To achieve comprehensive performance optimization, it is essential to *jointly manage both forms of interference*. The following motivational example illustrates this need by first demonstrating how addressing one interference source (e.g., cache contention) in isolation may inadvertently worsen the other (e.g., thermal effects), and second, by highlighting the intrinsic complexity of cache contention behavior.

6.1. Challenges and Novel Contributions

Building upon the observations from the motivational example in Section 1.2.2, we propose the *first application migration technique designed to mitigate cache contention in order to enhance system performance while simultaneously ensuring thermal safety.* Application migration between clusters can serve as a means to alleviate contention on the *source cluster*. However, it may also increase contention on the *destination cluster*, particularly if other applications are already active there. Thus, migration is beneficial *only if* the resulting contention on the destination is lower than that on the source, and the migration does not compromise thermal safety. Accordingly, any effective migration strategy must accurately predict its dual impact: on chip temperature and on performance, taking into account the cache contention at both the source and destination clusters.

While thermal estimation for a new application-to-core mapping can be readily achieved using established models such as the RC thermal model [100], evaluating performance impacts in the presence of cache contention is significantly more complex due to several key challenges:

1. As discussed in Challenge 1 (Section 1.3.1), cache contention exhibits intricate and asymmetric behavior, making it difficult to model analytically. The degree to which it affects performance depends on numerous factors including instruction execution patterns, frequency and timing of cache accesses, and inter-thread interactions. Moreover, common contention indicators often fail to provide direct quantification of their impact on application performance, as illustrated in Fig. 1.5.

2. Profiling concurrent execution scenarios at design time to estimate contention impact is impractical. The configuration space is exponential, encompassing variables such as the number of co-running applications, the number of threads per application, the operating V/f levels, and the timing of application arrivals. Variations in any of these parameters can substantially alter contention dynamics.

To address these challenges, we employ an NN-based prediction model in place of analytical approaches, which are infeasible under realistic assumptions. The proposed NN model estimates the performance impact of a potential application migration by considering the cache contention on both the source and destination clusters. It uses as input a set of relevant performance monitoring counters that effectively reflect contention levels. This model forms the foundation of our smart cache contention mitigation approach, *SmartCM*, which periodically evaluates the effect of candidate migrations on overall system performance. Only migrations that are predicted to enhance performance *without exceeding the critical temperature threshold* (T_{crit}) are executed. Furthermore, *SmartCM* leverages DVFS to dynamically exploit any available thermal headroom and proactively prevent thermal violations arising from phase changes in application behavior.

In summary, our novel contributions are:

- We introduce a task migration technique to, for the first time, jointly mitigate cache contention and enforce thermal safety, unveiling new performance optimization potentials.
- This technique is enabled by an accurate, yet lightweight, NN model to predict the impact of a migration on performance considering cache contention on source/destination clusters.

6.2. Problem Formulation

Based on the system model and mathematical notations defined in Section 3.1, the objective of *SmartCM* is *to maximize the overall system performance, quantified by the response time of applications, through cache contention mitigation*, while maintaining thermal safety. Mathematically, we can express this problem as periodically finding matrix G and f_c $\forall c$, in order to maximize $\sum_{k \in K} \text{IPS}_k$, s.t., $\max T \leq T_{crit}$.

$$\text{Maximize} \sum_{k \in K} \text{IPS}_k \text{s.t.:} \ \max T \leq T_{crit}$$

Our *SmartCM* solves this problem as illustrated in Fig. 6.1. Our NN model is trained at design time to predict the impact of potential migrations on performance (Section 6.3) and used by *SmartCM* at runtime (Section 6.4.1). The application-to-cluster migration that would maximize the overall system performance without violating T_{crit} is then performed. Additionally, *SmartCM* periodically selects the V/f levels of the clusters to exploit any available thermal margins on the cores to further improve performance and to avoid any potential thermal violation that could occur due to changes in execution phases of applications (Section 6.4.2).

6.3. NN for Contention-Aware Performance Prediction

To address the challenge of performance prediction under cache contention and thermal constraints, we propose *SmartCM*, as depicted in Fig. 6.1. Our NN model is trained offline at design phase to predict the performance implications of potential application migrations (Section 6.3), and it is utilized at runtime by *SmartCM* (Section 6.4.1). Additionally, *SmartCM* periodically adjusts the V/f levels of clusters to exploit available thermal margins, further enhancing performance and mitigating thermal violations due to dynamic changes in application behavior (Section 6.4.2). The following subsections describe our methodology for constructing the training and test datasets, and for selecting both the feature set and architecture of the NN model.

6.3.1. Training Data Generation

The training data generation process, illustrated in the upper section of Fig. 6.1, follows our methodology proposed in PHASEL [5], as detailed in Chapter 4. We conduct a wide range of simulated migration scenarios involving several multi-threaded applications and collect detailed execution traces. Each migration scenario involves an application of interest (AoI), migrated from a source cluster s, operating at frequency f_s and concurrently running other applications γ, to a destination cluster d with frequency f_d, where it co-executes with applications ω. As discussed in Section 6.1, the configuration space is exponential due to combinations of application sets, thread counts, arrival times, and more. Therefore, we constrain the simulations to scenarios involving one or two applications per cluster, each utilizing four threads and arriving simultaneously, as depicted in Fig. 6.1.

It is important to note that in our evaluation (Section 6.5), the trained model is applied to previously unseen scenarios with multiple applications per cluster and varying thread counts (1 to 4 threads), demonstrating its generalization

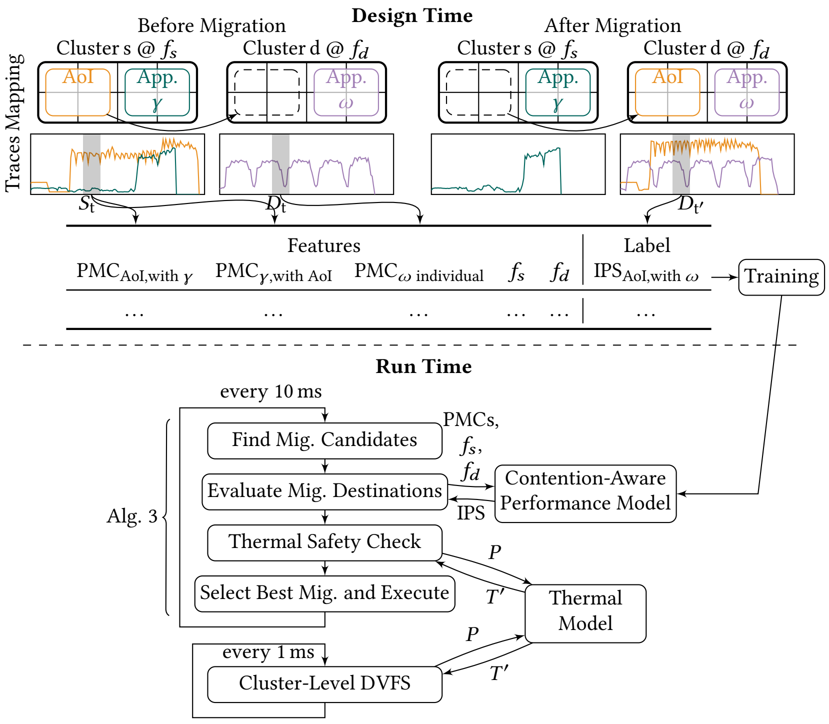


Figure 6.1.: Overview of *SmartCM*: During design time, training data is collected to train the NN model. At runtime, the NN is invoked periodically to predict the potential performance impact of candidate migrations, selecting the one that offers the greatest performance improvement while adhering to thermal constraints.

capability. In each scenario, our objective is to quantify the slowdown experienced by the AoI due to interference from γ and ω before and after migration, respectively. This is achieved by comparing the AoI's alone execution performance with its performance when co-running with γ (pre-migration) and with ω (post-migration).

The features used in the model are derived from the performance monitoring counters collected from cores executing AoI, γ, and ω on clusters s and d. Because each application may run at different V/f levels, the operating frequencies f_s and f_d are also included as input features. The model's target label is the IPS of AoI after migration.

To account for phase-dependent behavior in cache contention, each execution trace is segmented into fixed-length slices corresponding to the system's migration epoch (10 ms), as proposed in PHASEL [5]. From the trace of AoI and γ on cluster s before migration, we extract a slice $S_{\mathrm{t}} = (t_1, t_2)$, and collect:

- $\mathrm{PMC}_{\mathrm{AoI}}$: the PMCs of AoI during co-execution with γ.
- PMC_{γ}: the PMCs of γ during the same interval.
- $(\mathrm{ins}_1^{\mathrm{AoI}}, \mathrm{ins}_2^{\mathrm{AoI}})$: the instruction range executed by AoI between t_1 and t_2.

To obtain the label, we locate a corresponding slice $D_{\mathrm{t}'} = (t_1', t_2')$ in the post-migration trace, where AoI co-executes with ω on cluster d at frequency f_d, executing the same instruction range $(\mathrm{ins}_1^{\mathrm{AoI}}, \mathrm{ins}_2^{\mathrm{AoI}})$. From this, we compute $\mathrm{IPS}_{\mathrm{AoI}'}$, the label indicating the performance of AoI after migration. Additionally, we extract a third slice, D_{t}, from the standalone execution trace of ω, and collect its PMCs PMC_{ω}.

Together, the three slices—S_{t}, D_{t}, and $D_{\mathrm{t}'}$—form a complete migration scenario, capturing the behavioral characteristics of AoI, γ, and ω before migration, and the performance outcome of AoI after migration.

Each valid slice results in one training sample. This process is repeated across a large number of scenarios to construct a comprehensive dataset. The final training dataset comprises approximately 2 million samples, each consisting of the feature tuple $(\mathrm{PMC}_{\mathrm{AoI}}, \mathrm{PMC}_{\gamma}, \mathrm{PMC}_{\omega}, f_s, f_d)$ and the corresponding label $\mathrm{IPS}_{\mathrm{AoI}'}$.

6.3.2. Feature Selection and Model Topology

The constructed training dataset includes the operating frequencies of the source and destination clusters (f_s and f_d), along with a set of performance monitoring counters (PMCs) recorded periodically. These counters comprise the IPS and several cache- and memory-related statistics for AoI, as well as for the co-executing applications γ and ω. To facilitate model evaluation, the dataset is partitioned randomly into 75% for training and 25% for testing.

We initially focus on developing a model that delivers the highest possible prediction accuracy, and subsequently reduce its complexity to minimize runtime overhead.

Feature selection is performed using a combination of *Lasso Regression*, to identify the most predictive input variables for the target label $\text{IPS}_{\text{AoI}'}$, and Pearson Product-Moment Correlation analysis, to assess redundancy and mutual dependencies among features.

The most accurate model configuration employs six fully connected hidden layers, each with 256 neurons. This deep topology achieves exceptional predictive performance, yielding a mean-absolute-percentage error (MAPE) score of 1.5%. However, this configuration incurs a significant inference delay of 35 µs on the target platform, rendering it impractical for online deployment due to excessive computational overhead.

To achieve a better balance between prediction accuracy and runtime efficiency, we systematically reduce both the number of input features and the network complexity. The optimal trade-off is achieved with a neural network that utilizes a compact feature set comprising six elements: f_s, f_d, IPS, LLC accesses, LLC misses, and DRAM accesses. The corresponding model architecture consists of four hidden dense layers with 64, 128, 128, and 64 neurons, respectively, all using ReLU activation. The output layer contains a single neuron with linear activation.

This refined configuration attains an MAPE score of 2.3%, while reducing the inference time to just 13 µs on the target platform—making it suitable for integration in real-time migration decisions.

6.4. Smart Cache Contention Mitigation

The runtime operation of *SmartCM* is illustrated in the right section of Fig. 6.1. During system execution, *SmartCM* is activated periodically at fixed *migration epochs*—each lasting 10 ms[1]. At every such epoch, the trained NN model is queried to evaluate the performance impact of all feasible application migrations. *SmartCM* then initiates the migration that is predicted to yield

[1] This aligns with the Linux control interval, during which task migration operations are typically allowed [58].

the highest performance improvement, provided that it does not result in a thermal violation (i.e., exceeding T_{crit}). In parallel, *SmartCM* also executes at finer-grained *DVFS epochs* of 1 ms[2]. At each DVFS epoch, the technique adjusts the V/f levels of individual clusters to exploit existing thermal headroom. This allows for further performance gains while maintaining thermal safety, particularly in response to dynamic changes in application execution phases that could otherwise trigger thermal limit violations.

6.4.1. ML-Based Application Migration

The migration policy implemented by *SmartCM* is described in Algorithm 3, which proceeds through the following steps.

1) Identification of Migration Candidates *SmartCM* first identifies applications that are eligible for migration. An application of interest (AoI) becomes a migration candidate if other applications, denoted γ, are concurrently running on the same source cluster s. Given that the system supports cluster-level DVFS, both AoI and γ operate at the same frequency level f_s.

2) Destination Cluster Selection For each migration candidate, the system examines the available clusters to identify those with sufficient free cores to accommodate the AoI. If such a destination cluster d exists, a potential migration x is added to the set of migration options X. However, this migration will only be executed if it both improves overall system performance and satisfies thermal safety requirements.

3) Evaluation of Performance Impact A migration affects the performance of applications on the source and destination clusters only. To determine its benefit, we compare the total IPS after migration, denoted IPS'_t, against the baseline performance IPS_t before migration. Although only AoI is migrated, the migration influences the performance of γ and ω (applications on source and destination clusters, respectively), due to changes in cache contention.

[2] This is consistent with the frequency update interval used in commercial processors such as Intel's Turbo Boost [125].

Algorithm 3 *SmartCM* Migration Policy

```
Input: # of apps $K$, improvement factor $I$, current power $P$
Output: application to migrate AoI, destination cluster $d$
  $X \leftarrow \{\}$                                                   ▷ empty list of migration operations
  for each AoI $\in \{1, \ldots, K\}$ do                                ▷ iterate over running apps
      $s \leftarrow \{c : g_{c,\text{AoI}} = 1\}$                       ▷ source cluster
      $\gamma \leftarrow \{k : g_{s,k} = 1\} \setminus \{\text{AoI}\}$  ▷ co-running apps with AoI
      if $|\gamma| < 1$ then continue                                   ▷ AoI is running individually
      for each $d \in \{1, \ldots, C\} \setminus \{s\}$ do              ▷ iterate over other clusters
          $J \leftarrow \{i : q_{i,d} = 1 \wedge \forall k\ v_{i,k} = 0\}$   ▷ free cores on cluster $d$
          if $|J| < h_{\text{AoI}}$ then continue                       ▷ not enough free cores on $d$
          $\omega \leftarrow \{k : g_{d,k} = 1\}$                       ▷ apps running on cluster $d$
          $\text{IPS}_t \leftarrow \text{IPS}_{\text{AoI}} + \text{IPS}_\gamma + \text{IPS}_\omega$   ▷ current performance
          $\text{IPS}'_t \leftarrow \textsc{Predict}(f_s, f_d, \text{AoI}, \gamma, \omega)$   ▷ new performance
          $\Delta\text{Perf} \leftarrow \text{IPS}'_t / \text{IPS}_t - 1$
          if $\Delta\text{Perf} > I \wedge \textsc{Safe}(f_s, f_d, \text{AoI}, s, d)$ then
              $X \leftarrow X \cup \{(\text{AoI}, s, d, \Delta\text{Perf})\}$   ▷ save migration
      $x_{\text{best}} \leftarrow \arg\max X(\Delta\text{Perf})$        ▷ migration with best performance
  procedure $\textsc{Predict}(f_s, f_d, \text{AoI}, \gamma, \omega)$    ▷ performance prediction
      $\text{IPS}'_{\text{AoI}} \leftarrow PM(f_s, f_d, \text{PMC}_{\text{AoI}}, \text{PMC}_\gamma, \text{PMC}_\omega)$
      $\text{IPS}'_\gamma \leftarrow PM(f_s, f_s, \text{PMC}_\gamma, \text{PMC}_{\text{AoI}}, 0)$
      $\text{IPS}'_\omega \leftarrow PM(f_d, f_d, \text{PMC}_\omega, 0, \text{PMC}_{\text{AoI}})$
      return $\text{IPS}'_{\text{AoI}} + \text{IPS}'_\gamma + \text{IPS}'_\omega$
  procedure $\textsc{Safe}(f_s, f_d, \text{AoI}, s, d)$                 ▷ thermal safety check
      $P \leftarrow P^o + P^{\text{AoI}}(s \rightarrow d, f_s \rightarrow f_d)$   ▷ power with AoI running on $d$
      $\hat{T} \leftarrow \max TM(P)$                                   ▷ peak steady-state temperature on chip
      return $\hat{T} < T_{\text{crit}}$
```

When multiple applications co-run with AoI on the source cluster, IPS_γ is calculated as the average IPS across these applications, and PMC_γ reflects the aggregate performance counters from all cores assigned to them, excluding those used by AoI. The same logic applies for ω on cluster d.

The total pre-migration performance is defined as:

$$\text{IPS}_t = \text{IPS}_{\text{AoI}} + \text{IPS}_\gamma + \text{IPS}_\omega$$

To estimate the performance impact of migration x, the trained NN is invoked as follows:

1. $\mathrm{IPS}'_{\mathrm{AoI}}$ is predicted using $\mathrm{PC}_{\mathrm{AoI}}$, PC_{γ} (pre-migration), and PC_{ω} (post-migration).
2. To estimate IPS'_{γ}, the model is queried with PC_{γ}, $\mathrm{PC}_{\mathrm{AoI}}$, and zeros in place of PC_{ω}, since γ and AoI will no longer co-run.
3. Similarly, IPS'_{ω} is predicted using PC_{ω}, zeros for PC_{γ}, and $\mathrm{PC}_{\mathrm{AoI}}$, as AoI will be newly introduced into ω's environment.

The total predicted post-migration performance is then:

$$\mathrm{IPS}'_t = \mathrm{IPS}'_{\mathrm{AoI}} + \mathrm{IPS}'_{\gamma} + \mathrm{IPS}'_{\omega}$$

The relative performance gain is computed as:

$$\Delta\mathrm{Perf} = \frac{\mathrm{IPS}'_t}{\mathrm{IPS}_t} - 1$$

If $\Delta\mathrm{Perf} > I$, where I is an empirically determined threshold to account for model inaccuracies, migration x is flagged as beneficial in terms of performance.

4) Thermal Safety Verification Once a performance gain is predicted, *SmartCM* assesses whether migration x satisfies thermal constraints. Using the RC thermal model [100], the new steady-state temperature T' of the affected cores is predicted. Because AoI will transition from operating at f_s on cluster s to f_d on cluster d, its power consumption P^{AoI} must be adjusted accordingly for accurate thermal prediction. If the predicted temperature T' remains below the critical threshold T_{crit}, migration x is considered thermally safe. The associated performance improvement $\Delta\mathrm{Perf}_x$ is stored, and the process continues to evaluate other candidate migrations.

At the conclusion of this process, the migration x_{best} yielding the highest $\Delta\mathrm{Perf}_{\mathrm{best}}$ among all thermally safe candidates is executed.

6.4.2. Cluster-Level DVFS

To ensure clusters operate at the highest feasible V/f levels while avoiding thermal violations, *SmartCM* dynamically manages DVFS settings in response to runtime workload changes. At every DVFS epoch (1 ms), the system predicts the steady-state temperatures based on current power consumption using the RC-thermal model [100]. If a thermal violation is anticipated, *SmartCM* initiates an iterative search to identify which clusters should be throttled down.

In each iteration, the cluster c with the hottest core is added to the throttling list. The thermal model is then updated to reflect the reduced power consumption assuming c is operated at the next lower V/f level. This process is repeated until the maximum predicted temperature across all cores falls below T_{crit}. Once this condition is met, all selected clusters are downscaled to their new V/f levels.

Conversely, if no thermal violation is predicted during a given epoch and thermal margins exist, *SmartCM* considers increasing the V/f levels. However, upscaling is conservatively limited to a single step per epoch. This strategy reduces the error in power scaling estimates and improves the accuracy of subsequent thermal predictions.

6.5. Experimental Evaluation

All simulations are conducted using the HotSniper [48] simulation framework, described in Section 3.2.1. The simulated architecture is the 64-core clustered multicore in experimental setup (A) in Section 3.1.1. To evaluate *SmartCM*, we utilize 15 diverse multi-threaded applications drawn from the PARSEC [49] and SPLASH-2 [50] benchmark suites (detailed in Appendix A). These applications are executed with *simmedium* and *large* input sizes, respectively. The selected applications are: *blackscholes*, *bodytrack*, *streamcluster*, *swaptions*, *x264*, *cholesky*, *fft*, *fmm*, *lu.cont*, *lu.ncont*, *radiosity*, *radix*, *raytrace*, *water.sqn*, and *water.sp*. Each application supports varying levels of parallelism and is executed using 1 to 4 parallel threads. To assess system performance under realistic and varied workloads, we construct three distinct mixed workloads. Each workload includes 40 randomly selected applications, with half being memory-intensive. Application arrival times follow a Poisson distribution across five different arrival rates to reflect varying system utilization levels. It is important to emphasize that in these evaluation scenarios, up to 40 applications with heterogeneous thread configurations arrive dynamically, leading to scenarios with as many as 8 applications executing concurrently on a single cluster. These workload configurations are *unseen* by the NN model during training, thereby allowing us to rigorously assess its generalization capabilities in realistic, high-utilization conditions.

6.5.1. Comparison Technique

The most relevant state-of-the-art method to compare with *SmartCM* is the technique proposed in [89], referred to hereafter as *SoA*. This approach also aims to alleviate cache contention via application migration, but operates under a power constraint. At each migration epoch, *SoA* evaluates the accumulated shared bus accesses to guide the redistribution of applications across clusters, with the goal of balancing contention effects chip-wide. Furthermore, *SoA* incorporates a NN model to estimate the highest allowable V/f level for each cluster that remains within the chip's power envelope.

However, since *SmartCM* is designed to operate under a thermal constraint rather than a power constraint, we adapt *SoA* accordingly to ensure a fair comparison. Specifically, we replace the power constraint in *SoA* with the Thermal Safe Power (TSP) constraint introduced in [126]. The TSP defines the maximum allowable power per active core that ensures compliance with the critical temperature threshold T_{crit}, based on the number of concurrently active cores. This modification aligns the optimization objectives of both techniques, i.e., maximizing performance while maintaining thermal safety, thus enabling a direct and fair comparison. To maintain consistency, we train the *SoA* model using the same set of simulation traces employed for training our own NN model.

6.5.2. Experimental Results

Figure 6.2 (top) presents the normalized average response time across different workloads and arrival rates for both *SmartCM* and *SoA*. The results clearly indicate that *SmartCM* consistently outperforms *SoA* by achieving shorter

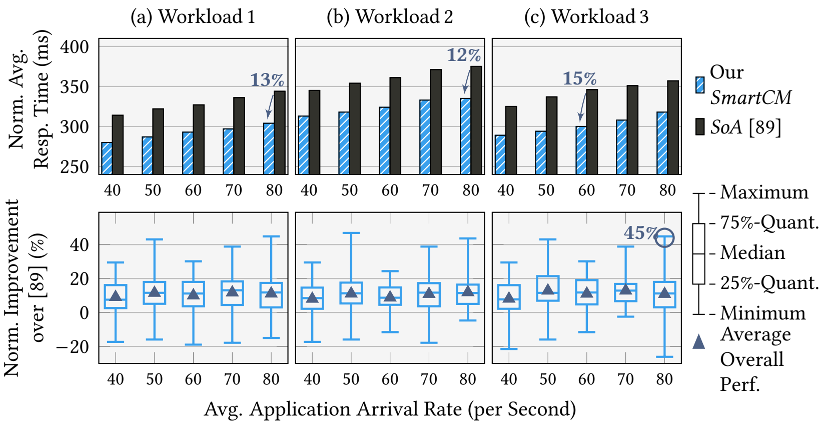


Figure 6.2.: *SmartCM* delivers notable performance enhancements across three distinct mixed workloads, achieving improvements of up to 45% for certain applications compared to *SoA* [89]. On average, *SmartCM* enhances overall system performance by 11%, with gains reaching as high as 15% relative to *SoA* [89].

average response times, thereby demonstrating higher overall system performance, yielding an average improvement of 11%, and up to 15% in certain cases. This performance advantage is primarily attributed to *SmartCM*'s more accurate performance prediction model, which leverages multiple relevant cache-related indicators. In contrast, *SoA* relies solely on DRAM access counts to assess cache contention, which limits its effectiveness.

To offer a more granular comparison, we analyze the execution time of each individual application under *SmartCM* versus its execution under *SoA*. The relative improvements or degradations in execution time are illustrated in Fig. 6.2 (bottom). These results reveal that application performance varies across workloads and arrival rates depending on the migration decisions taken by each technique. While the majority of applications experience performance gains (up to 45%), a small subset exhibits performance degradation under *SmartCM* relative to *SoA*.

Two key factors explain this behavior:

1. As previously discussed, migrating an application is expected to alleviate contention on the source cluster—benefiting co-running applica-

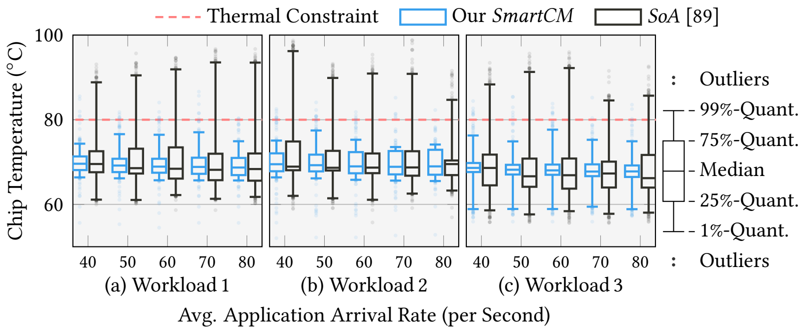


Figure 6.3.: *SmartCM* ensures thermally-safe system operation while enhancing the performance of all three mixed workloads. In contrast, *SoA* [89] frequently violates the thermal constraint.

tions there, while potentially introducing additional contention on the destination cluster, thus impacting its resident applications.

2. The DVFS strategy employed by *SoA* often selects higher frequencies for some clusters compared to *SmartCM*, which can yield localized performance improvements. However, as shown in Fig. 6.3, these gains come at the cost of thermal constraint violations, with peak temperatures reaching up to 98 °C, despite the use of TSP-constrained training data.

This phenomenon is attributed to the architecture of *SoA*'s decision pipeline, where the NN directly predicts the cluster frequency levels. Any inaccuracies in these predictions propagate directly to V/f settings, potentially causing the system to exceed thermal or power constraints.

By contrast, *SmartCM* employs its NN exclusively for performance estimation in the context of migration decisions. Only migrations predicted to offer performance improvements above a confidence threshold I are considered, mitigating the effect of potential model mispredictions. Even in cases where a performance misprediction leads to a suboptimal migration, thermal safety is enforced through an independent verification step. Moreover, *SmartCM* incorporates a cautious DVFS policy that estimates steady-state temperatures prior to frequency adjustments and incrementally increases V/f levels by a single step per epoch. This controlled frequency scaling improves the accuracy of thermal and power predictions, thus reducing the risk of violating thermal constraints. Finally, we observe that *SmartCM* performs only 1.5

migrations per application on average, meaning each application is migrated fewer than two times over its execution lifetime. This reflects the stability and efficiency of *SmartCM* 's runtime management strategy.

6.5.3. Runtime Overhead

To assess the runtime overhead of our proposed technique, *SmartCM* is implemented as an application and executed on a dedicated core of the same target platform, operating at 4 GHz. We utilize high-resolution time counters to measure the execution time of individual functional stages in the migration and DVFS policies.

The execution time of these policies varies depending on the system state at each control epoch. Specifically, during each migration epoch, the number of candidate migrations is influenced by the current application mix—e.g., the number of active threads, application execution duration, and system occupancy. Under high-utilization conditions, fewer applications succeed in identifying valid destination clusters, as most clusters are either fully occupied or lack sufficient free cores to host the migration candidate. Conversely, at lower utilization levels, applications are more likely to find suitable destinations, increasing the number of evaluated migrations. Similarly, during each DVFS epoch, the thermal distribution across the chip and the available thermal margins affect how many clusters require frequency adjustments to maintain operation at the highest thermally safe V/f levels.

We report the average runtime overhead based on the execution of the workloads evaluated in Section 6.5. On average, the execution of the migration policy incurs an overhead of 170 µs, accounting for only 1.7% of the 10 ms migration epoch. Of this, just 39 µs are attributed to model inference, made possible by the lightweight neural network architecture described in Section 6.3.2. The DVFS policy introduces an average overhead of 9.5 µs, which constitutes merely 0.95% of the 1 ms DVFS control interval.

In conclusion, both the migration and DVFS components of *SmartCM* exhibit negligible runtime overhead, making the technique practical for deployment in real-world multi-core systems.

6.6. Summary

This chapter introduced the first technique that jointly addresses cache contention and thermal constraints in clustered multi-core processors using application migration in conjunction with DVFS, to optimize performance under thermal constraints.

Tackled Challenges The inherent complexity of shared cache contention in Challenge 1 (Section 1.3.1) was effectively managed using a NN model. The migration knob allowed *SmartCM* to continuously adjust application-to-cluster mapping by migrating applications to clusters where the lowest contention for the LLC occurs, thereby maximizing the overall system performance. Once again, the lightweight nature of our NN model enabled *SmartCM* to achieve its objective with negligible overhead, thereby successfully tackling Challenge 5 (Section 1.3.5).

Achievements and Takeaways With our training methodology (PHASEL [5]), the NN model was able to successfully generalize to unseen applications and contention scenarios at runtime. Compared to the state of the art, our method yielded substantial performance improvements, achieving gains of up to 45%, and an average improvement of 11%, while consistently maintaining chip-level thermal safety. This confirms the significance of jointly considering contention and temperature problems, as motivated in Chapter 1.

Insights Similar to related works, our technique did not consider swap-migrations of applications. Instead, *SmartCM* migrates an application only when there is a cluster with sufficient free cores to accommodate its threads. Exploring additional possibilities through swapping is a logical continuation of this work. Enabling this option is straightforward and requires only a minor change in our technique. However, the overhead of testing all possible options might be prohibitive at runtime. Recent advancements in generative AI could open the door for a new solution, whereby all these options and their impacts could be explored with a single prompt to the model, returning a new application-to-cluster mapping that shuffles applications across the chip, without the need to exhaustively evaluate every possibility individually.

7. ML-Based Contention-Aware Performance Optimization for Clustered Multicores with HBM

This chapter presents *MTCM* [3], the first resource management technique that considers cache contention in maximizing system performance, while maintaining thermal safety across *both* the processor and the 3D HBM stack. Enabled by our accurate, yet lightweight, neural network models, our proposed task migration and dynamic voltage and frequency scaling policies can accurately predict the impact of runtime decisions on performance and temperature of both subsystems. Our extensive evaluation experiments reveal a significant performance improvement over existing state of the art by up to 1x, while maintaining thermal safety of both the multicore and the HBM.

7.1. Background

The increasing demand for high-performance computing has brought memory bandwidth as a key performance bottleneck, necessitating *structural* enhancements throughout the memory hierarchy. Among the most impactful innovations in this space is the introduction of HBM. Characterized by its 3D-stacked architecture and wide interface with multiple memory channels, HBM delivers substantially higher bandwidth, reduced latency, and improved energy efficiency, making it particularly suitable for memory-intensive applications. Recent progress in 2.5D and 3D integration technologies has enabled the deployment of HBM alongside commercial clustered multi-core processors [127]. A conceptual illustration of this architecture is provided in Fig. 7.1. The integration of HBM with clustered multicores allows multiple clusters to simultaneously exploit the memory channels of HBM through their respective memory controllers, significantly reducing memory access bottlenecks

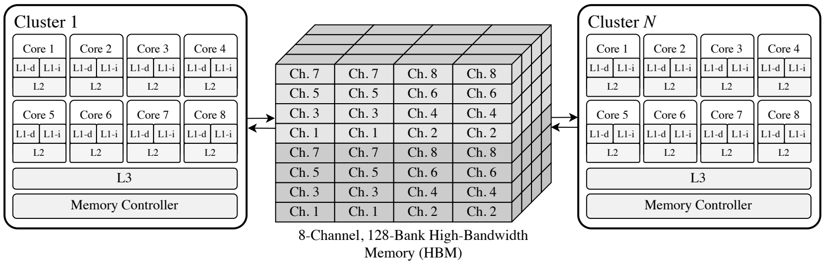


Figure 7.1.: The HBME2[102] as the main memory of the simulated AMD Zen 3-like homogeneous clustered processor.

and enhancing overall system throughput. However, this integration is not without challenges, arising from the intrinsic limitations of both subsystems, the HBM and the clustered multi-core architecture.

On the one hand, the high thermal density of HBM makes it susceptible to overheating, especially under intensive parallel access to its thermally-coupled memory layers [94]. Such thermal conditions can activate the DTM unit, which forces overheated memory channels into low-power or idle states [97] until thermal stability is restored. This leads to prolonged memory stalls and significant degradation in system performance. Various mitigation strategies have been proposed in the literature, ranging from architectural solutions involving embedded cooling mechanisms [94], to higher-level, system-oriented thermal management policies [98][97].

On the other hand, clustered multi-core architectures face their own interference challenges. Simultaneous execution of multiple applications within the same cluster can result in two primary issues: (i) contention for limited shared cache capacity, which can delay application progress, and (ii) thermal interference between adjacent cores, which may raise chip temperature and trigger the TCC [52]. When activated, the TCC uniformly reduces the V/f levels of all cores, consequently reducing performance across the chip.

A wide range of system-level resource management techniques have been proposed to address these issues—either independently or in combination—through strategies such as application mapping, migration, and DVFS [84, 54, 8, 21].

Nevertheless, prior work has primarily addressed the limitations of clustered multicores and HBM in isolation. As the following motivational example illustrates, the integration of these two subsystems introduces complex new trade-offs involving cache and thermal interference across both domains. Neglecting the cross-subsystem interactions risks underutilizing performance opportunities and failing to enforce system-level thermal safety.

7.2. Motivational Example

We simulate an AMD Zen 3-like 64-core processor, integrated with an 8-channel HBM system, modeled after the HBM2E model [102], as detailed in Section 3.1.1. Each cluster is mapped to a dedicated memory channel using a one-to-one cluster-to-channel configuration (i.e., cluster *n* accesses channel *n*), similar to the mappings used in the literature [98, 128], as illustrated in Fig. 7.2(a). We consider an 80 °C temperature limit as the thermal constraint for *both* the multicore and the HBM subsystems. To isolate the impact of resource management decisions, we disable the default thermal protection mechanisms, namely, the TCC for the multicore and the DTM mechanism for the HBM, which would otherwise throttle V/f levels or transition memory channels to low-power states upon thermal violations.

In our setup, clusters 2, 4, 5, 7, and 8 are fully utilized by compute-intensive background workloads running at the minimum V/f of 1.0 GHz, and they generate no memory traffic. Clusters 1, 3, and 6 are used to run three benchmark applications from the SPLASH-2 suite [50]: *cholesky*, *water.sp*, and *radix*, respectively. For controlled evaluation, these clusters are fixed to a medium V/f level of 2.8 GHz to ensure that no thermal violations originate from the multicore side. A fourth application, *lu.cont*, requiring four cores, must be assigned to one of the active clusters. This creates three possible application-to-cluster mappings, where *lu.cont* can be co-located with one of the three previously placed applications, as illustrated in Fig. 7.2(b). We evaluate the performance and thermal implications of each scenario.

Scenario 1 *lu.cont* is co-executed with *cholesky* on cluster 1. This configuration leads to substantial contention, resulting in an average performance degradation of 18.25% across both applications compared to their respective standalone execution times. More critically, it causes thermal violations in

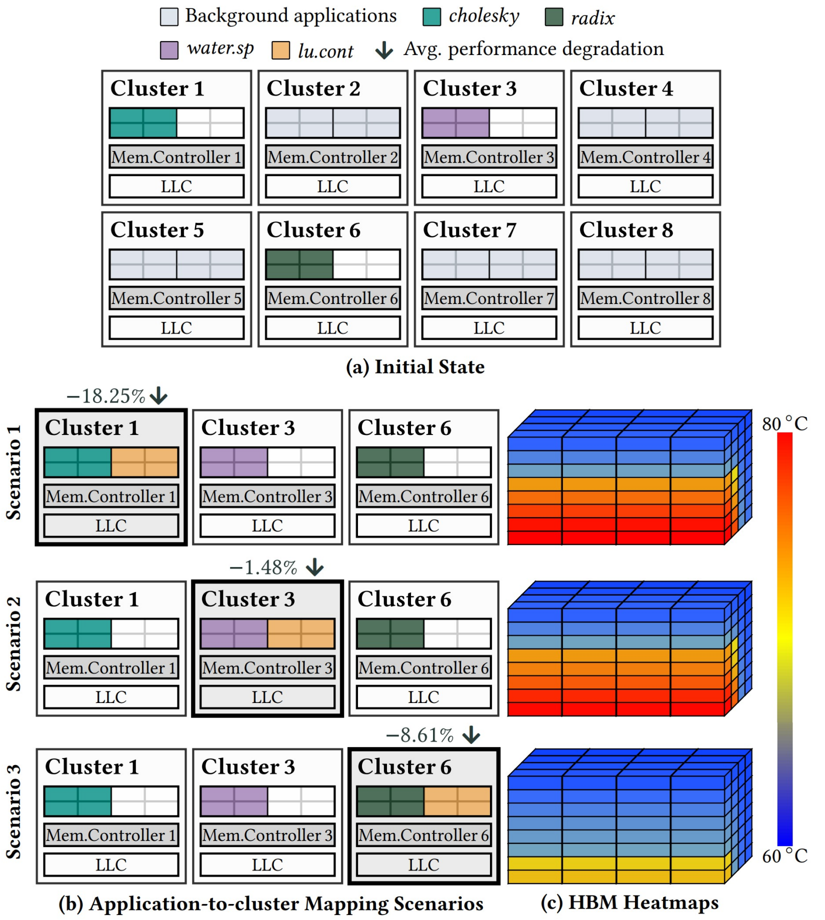


Figure 7.2.: While the application-to-cluster mapping in Scenario 2 yields greater performance improvements than the other scenarios—attributable to reduced cache contention on the cluster—it results in a temperature violation on the HBM. Neglecting such thermal implications at runtime would activate the DTM, thereby diminishing the expected performance benefits.

multiple HBM banks, with channel 1 reaching up to 85 °C. This thermal rise is due to the high memory access intensity from both applications, which not only heats up channel 1 but also propagates thermal stress to adjacent channels (2, 3, and 4), as shown in Fig. 7.2(c).

Scenario 2 *lu.cont* is mapped to cluster 3 alongside *water.sp*. This results in the highest overall performance, with only a 1.48% slowdown observed, suggesting negligible cache contention between the two applications. While *water.sp* exhibits minimal memory activity, *lu.cont* generates sustained accesses to channel 3. In conjunction with the heavy access pattern from *cholesky* on channel 1—located vertically adjacent in the memory stack—thermal coupling results in multiple banks exceeding the 80 °C threshold, thereby rendering this configuration thermally unsafe.

Scenario 3 *lu.cont* is assigned to cluster 6, sharing it with *radix* and accessing channel 6—thermally distant from channel 1. In this setup, the temperature across all memory banks remains below the thermal limit, and contention is reduced due to spatial separation of high-access channels. However, this mapping results in a moderate performance degradation of 8.61% for the co-running applications.

In summary, while Scenario 2 provides the best raw performance, it violates thermal constraints and would trigger DTM in a real system, potentially nullifying the performance benefit due to channel throttling. Scenario 3, though suboptimal in performance, maintains thermal safety and therefore emerges as the most balanced and sustainable mapping under thermal constraints. As applications transition through different execution phases at runtime, their cache and memory access patterns—and thus their thermal impact—can vary significantly. This highlights the necessity for a dynamic, system-level resource management approach that continuously adapts application-to-cluster mappings via task migration and regulates V/f levels through DVFS, to maximize system performance without violating thermal safety.

7.3. Challenges and Novel Contributions

As illustrated in the previous motivational example, maximizing system performance in the presence of thermal and cache-related constraints requires a comprehensive evaluation of all possible application-to-cluster mappings. This includes estimating the cache contention-induced slowdowns and the corresponding thermal effects on both the multicore and the HBM. Such analysis must also be feasible at runtime to inform migration decisions. Therefore, there exists a critical need to accurately predict both the *post-migration* system performance and the thermal state of the underlying hardware. This requirement poses several challenges, which we address in this work.

Performance Prediction As discussed in Challenge 1 (Section 1.3.1), accurately estimating the performance impact of application migration in the presence of cache contention is inherently complex. Contention behavior is shaped by both architectural factors (e.g., cache associativity, capacity, memory bandwidth) and application-specific characteristics (e.g., cache accesses, cache miss rates). Due to this multifaceted dependency, contention patterns cannot be captured through analytical models or exhaustive design-time profiling, as previously demonstrated in [1, 2].

To address this challenge, we propose a lightweight neural network (NN) model trained on a limited set of design-time data. This model learns to infer performance degradation due to cache contention by leveraging multiple relevant features such as cache accesses, cache misses, and memory traffic. Once trained, the model generalizes well to previously unseen contention scenarios during runtime.

Temperature Prediction While predicting the multicore temperature for a new mapping is relatively straightforward using RC-based thermal models like HotSpot [100], predicting the thermal behavior of the HBM is significantly more challenging. This is due to the architectural configuration, where each cluster's memory controller maps to a distinct HBM channel. Thus, when an application is migrated, its memory requests are rerouted to a different physical channel, modifying the power consumption pattern across HBM banks. The result is a *new* per-bank power distribution across the memory stack, which serves as input for thermal prediction. However, since clusters may operate at different V/f levels, an application's memory access rate, and therefore its power consumption, varies depending on the target cluster's frequency. Therefore, accurately modeling the post-migration power profile requires predicting the application's memory access behavior at the new V/f level.

To solve this, we use a second lightweight NN model trained to scale the number of memory accesses according to the operating V/f of the destination cluster. This enables precise generation of the updated power distribution required for accurate HBM temperature prediction.

Runtime Overhead of Temperature Prediction As discussed in Challenge 4 (Section 1.3.4) and 5 (Section 1.3.5), to ensure timely adaptation to runtime dynamics, a system-level resource management framework must operate within short decision epochs. For example, Linux employs a 1 ms epoch for DVFS and a 10 ms epoch for task migration [58]. Traditional thermal simulation tools such as HotSpot [100], 3D-ICE [129], or Green's Function-based methods like 3DSim [130], are computationally expensive, with simulation times ranging from tens to thousands of milliseconds [131], making them unsuitable for online use. In contrast, ML-based approaches offer the potential for microsecond-level inference while maintaining sufficient accuracy. However, no open-source ML-based thermal simulators currently exist [131].

To fill this gap, we develop two compact and accurate NN-based thermal models—one for the multicore subsystem and one for the HBM—capable of delivering reliable temperature predictions with inference latency on the order of microseconds.

Proposed Contribution Leveraging the predictive capabilities of our performance and thermal NN models, we propose a novel resource management framework, *MTCM*. This technique integrates task migration and DVFS to optimize system performance under both cache and thermal constraints. *MTCM* periodically evaluates migration opportunities to improve the balance of cache contention across clusters. Before applying a migration, it predicts the post-migration performance and the thermal effects on both the multicore and the HBM, proceeding only if the migration is expected to yield a net performance gain without violating thermal constraints on either subsystem. In addition, *MTCM* employs a proactive DVFS policy that adjusts cluster frequency levels based on real-time thermal headroom. It boosts cluster V/f when thermal margins permit and throttles them to prevent overheating due to sudden power surges, adapting to the evolving behavior of applications throughout execution.

Collectively, these mechanisms enable *MTCM* to exploit the full performance potential of HBM-integrated clustered multicore systems while maintaining strict thermal safety. In summary, our novel contributions are:

- We train two *lightweight* NN models to accurately predict the impact of task migration on system performance considering cache contention in clusters, and the impact of changing V/f levels on the HBM access behavior of applications.
- We train two *lightweight* NN-based thermal models, engineered to accurately predict the temperature of the clustered multicore and the HBM within the stringent time constraints at runtime.
- Enabled by our fast and accurate NN models, we present *MTCM*, the first resource management technique that jointly mitigates cache contention while enforcing thermal safety on systems with an integrated clustered multicore and HBM, by means of task migration and DVFS.

7.4. Problem Formulation

Based on the system model and mathematical notations defined in Section 3.1, the goal of *MTCM* is to optimize response time and manage cache contention while ensuring thermal safety for cores and HBM. To model the memory subsystem, where each cluster c is associated with a dedicated memory controller MC_c, which accesses a unique HBM channel ch_c. Cores in cluster c issue memory requests exclusively through MC_c. The HBM is composed of L vertically stacked memory layers, each containing B banks. These banks are distributed across S physical memory channels. Each channel $\mathrm{ch} \in \{1, \ldots, S\}$ manages a disjoint subset of banks $B_{\mathrm{ch}} \subseteq \{1, \ldots, B\}$, and the temperature of channel ch is defined as:

$$T_{\mathrm{ch}} = \max_{b \in B_{\mathrm{ch}}} t_b,$$

where t_b is the temperature of bank b in the assigned layer.

Thermal behavior in the memory subsystem is modeled using TM_{hbm}, which estimates the memory-bank temperature matrix $\mathbf{T}_{\text{banks}} = [t_{b,\ell}] \in \mathbb{R}^{B\times L}$, based on the per-bank power dissipation map $\mathbf{P}_{\text{banks}}$. Thermal safety in the memory is similarly governed by the threshold T_{crit}, as specified by JEDEC standards [101], i.e., $\max_{b,\ell} t_{b,\ell} < T_{\text{crit}}$.

Mathematically, we can express this problem as periodically finding matrix G and f_c $\forall c$, in order to maximize $\sum_{k\in K} \text{IPS}_k$, subject to the following thermal constraints:

$$\text{Maximize} \sum_{k\in K} \text{IPS}_k$$

$$\text{Subject to:} \quad T_{\text{core}} \leq T_{\text{crit}}^{\text{core}}, \quad T_{\text{HBM}} \leq T_{\text{crit}}^{\text{HBM}}$$

7.5. NN-Based Models

As demonstrated in Sections 7.2 and 7.3, accurately estimating the post-migration impact on system performance and the thermal behavior of both the multicore and the HBM is a non-trivial challenge. To address this, we employ neural network (NN) models that are trained offline at design time and deployed online during runtime to predict these critical post-migration metrics. This section describes our methodology for constructing and training two core models: NN_p, which predicts performance in terms of Instructions Per Second (IPS), and NN_m, which estimates post-migration memory access behavior. We then introduce two additional models, NN_{cores} and NN_{hbm}, which are responsible for predicting the thermal responses of the multicore and HBM, respectively.

7.5.1. Performance and Memory Access Prediction

Figure 7.3 provides an overview of our design-time training methodology for the NN_p and NN_m models, structured as follows.

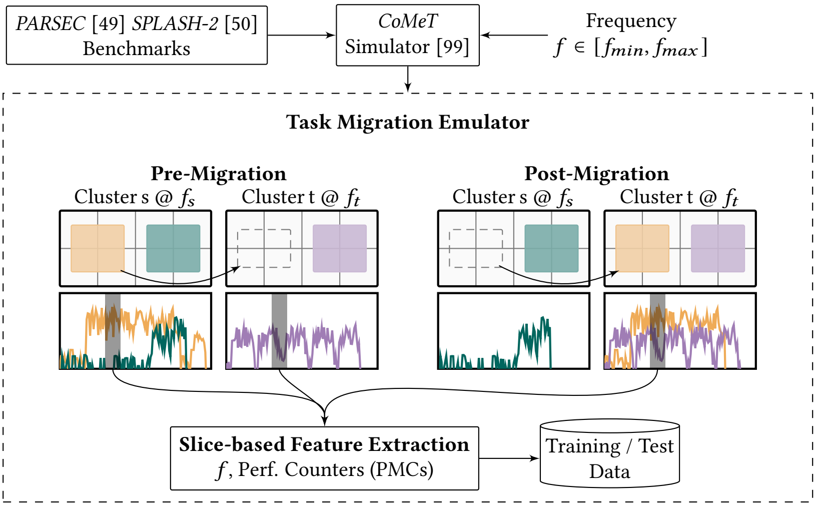


Figure 7.3.: Our slice-based training data generation methodology [5] performs a fine-grained analysis of application execution traces to capture the characteristics of their execution phases that are pertinent to predicting *post-migration* performance and memory access behavior.

1) Isolating Cache Contention Effects

We begin by executing an extensive set of simulations involving multi-threaded benchmark applications across all supported V/f levels. During each simulation, per-thread PMCs are collected every 1 ms, covering a comprehensive range of metrics such as IPS, IPC, L1/L2/L3 cache accesses and misses, as well as memory read/write counts. These *contention-free* traces (i.e., applications executed in isolation) establish a performance upper bound at each V/f level. Subsequently, we simulate all possible two-application combinations on the same cluster, each using 4 threads, to ensure full cluster utilization (8 cores), consistent with the one-thread-per-core model [51]. This setup allows direct attribution of any performance degradation to cache contention.

2) Emulating Task Migration Scenarios

We use the traces obtained from isolated and shared executions to emulate realistic migration scenarios. For each application App$_a$, we identify instances where it co-executes with an *other* second application App$_O$ on a source cluster s at frequency f_s, and simulate its migration to a target cluster t to co-execute with a new application App$'_O$ at frequency f_t. This process yields pairs of pre- and post-migration scenarios with corresponding 1 ms PMCs, IPS values, and V/f levels. Scenarios involving migrations to empty clusters (where App$_a$ executes alone post-migration) are also included to capture peak achievable performance at f_t.

3) Extracting Representative Samples

To reflect the actual runtime behavior, where migrations occur periodically (every 10 ms), we slice the execution traces from pre- and post-migration scenarios into 10 ms windows with 1 ms resolution. For each 10 ms pre-migration window (in which App$_a$ runs with App$_O$), corresponding slices from the isolated runs of App$_a$ and App$'_O$ are also extracted. These slices are aggregated at the thread level to construct application-level features. This enables the model to generalize to scenarios involving different numbers of active threads or co-runners, thus covering both upper and lower bounds of core occupation. At runtime, aggregated thread-level PMCs will be used to build the representations of App$_O$ and App$'_O$. This data engineering process results in a dataset of approximately two million entries. Each entry allows the models to learn the mapping: *Given the pre-migration PMCs of App$_a$ and App$_O$ at f_s, and the isolated characteristics of App$'_O$ at f_t, predict the post-migration IPS and memory accesses of App$_a$ at f_t.*

4) Feature Selection and Model Architecture Optimization

The final stage aims to minimize model size while maintaining prediction accuracy. Using the generated dataset, we apply Lasso Regression and Pearson Correlation analysis to rank features by importance and eliminate low-contribution features (e.g., L1-I and L1-D cache metrics). The data is randomly partitioned into 75% training and 25% test sets. We then use the KerasTuner

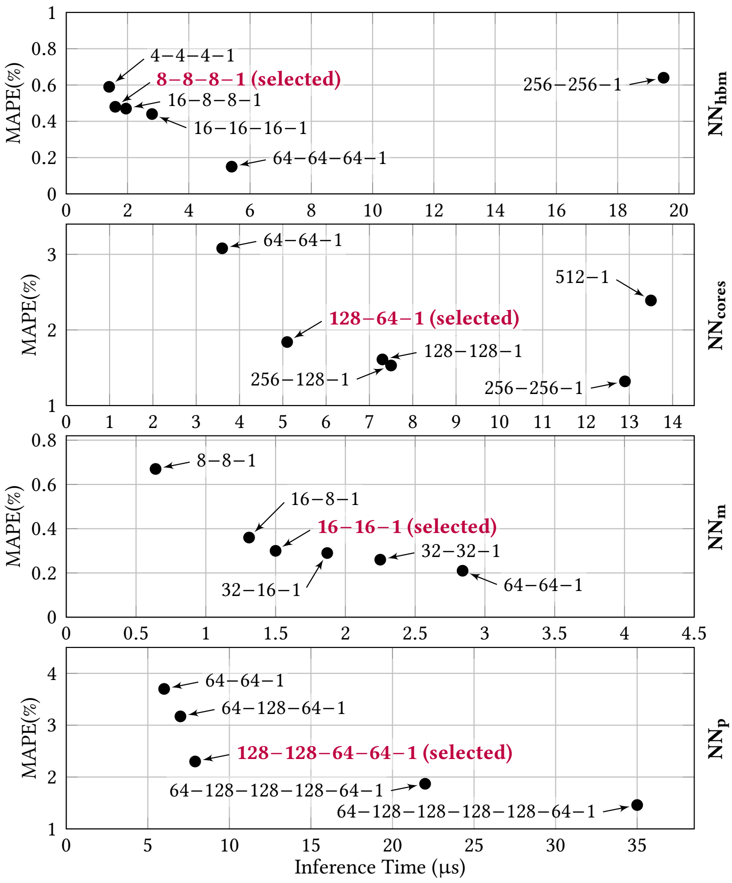


Figure 7.4.: An initial exploration using KerasTuner evaluates a range of neural network topologies with varying numbers of layers and neurons per layer. After discarding non-converging models, a Pareto front of viable candidates is identified. The final model topology is selected to balance prediction accuracy and inference latency on the target platform, thus allowing our runtime policies to optimize system performance within thermal constraints, while incurring minimal runtime overhead.

library to perform a structured hyperparameter search across multiple configurations, including numbers of layers, neurons per layer, batch size, regularization terms, and dropout factors. Figure 7.4 presents the explored model topologies, each evaluated by its MAPE and inference time on the target platform. Among the Pareto-optimal configurations, we select the models that meet strict run-time inference constraints (discussed in Section 7.7.4). Topologies with superior MAPE scores are also shown to illustrate the trade-offs between accuracy and runtime overhead. The final selected architecture for NN_p consists of four dense hidden layers with 128, 128, 64, and 64 neurons, followed by a single-neuron output layer. For NN_m, we use two hidden layers of 16 neurons each, also followed by a single-neuron output layer. The inference latency of NN_p and NN_m on our target platform is 7.9 µs and 2 µs, respectively. The achieved MAPE scores are 2.3% for NN_p and 0.3% for NN_m. The small but non-zero prediction error of these models is accounted for in the decision logic of *MTCM*, as discussed in Section 7.6.

7.5.2. Temperature Prediction for the Multicore and HBM

To enforce thermal safety at runtime, system-level RM decisions must be constrained to those that do not cause thermal violations on either the multicore or the HBM. Achieving this requires accurate and low-latency thermal prediction models for both subsystems, capable of delivering reliable estimates within the strict time budgets of runtime control loops. Such models must be trained on data representative of real scenarios, where core utilization and memory access patterns vary dynamically across different frequencies and workloads. However, constructing such a dataset using conventional simulation techniques becomes impractical for such a system. For instance, in our target architecture, comprising 64 cores and an HBM module with 8 channels and 128 banks (as defined in [102]), the number of required simulation permutations grows exponentially, making exhaustive design-time data generation infeasible. To overcome this scalability bottleneck, we adopt a hybrid methodology combining targeted simulations with synthetic data generation, as described below.

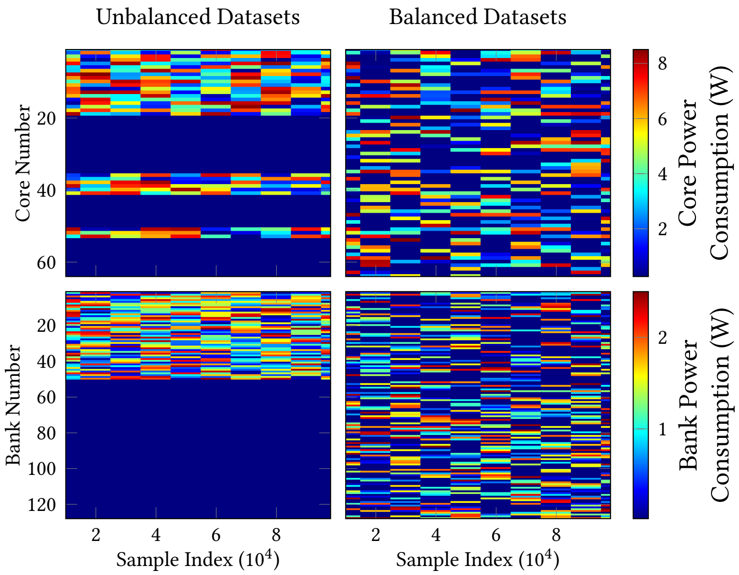


Figure 7.5.: Examples of skewed power value distributions in the training dataset, where activity is concentrated in a limited number of cores or memory banks, potentially restricting the generalizability of our thermal models to unseen power consumption patterns at runtime. In contrast, our synthetic data generation approach ensures a well-balanced distribution across all cores and memory banks, both in terms of occurrence frequency and power value ranges.

1) Generating the Base Datasets

Using the simulation framework introduced in Section 3.2.2, we first conduct a suite of controlled simulations in which single multi-threaded applications are mapped to one cluster at a time. These simulations are executed across all supported V/f levels to ensure coverage of a wide operating range. During each simulation, we collect fine-grained (1 ms resolution) power traces and corresponding steady-state temperatures from both the multicore and the HBM, using the HotSpot thermal simulator [100]. This yields two initial datasets: one for training the multicore thermal model NN_{cores} and the other for the HBM model NN_{hbm}.

2) Synthetic Data Augmentation

Although the base datasets provide accurate thermal profiles, they fail to capture the contention and heterogeneity of runtime workloads, where multiple applications simultaneously execute with varying thread counts, memory intensities, and frequency settings. To enhance the diversity and representativeness of the training data, we generate synthetic power traces for both the multicore and HBM subsystems. This is done by combining and randomizing power values from the single-application simulations to construct synthetic multi-application execution scenarios. These synthetic traces reflect a range of operational conditions, including varying levels of cluster occupation, frequency scaling, and memory utilization. We limit the number of synthetic traces to 2 million per subsystem to manage data size while ensuring sufficient coverage. However, this randomized generation introduces data imbalance. Specifically, as shown in the left part of Fig. 7.5, certain cores and banks may be underrepresented in terms of access frequency or power levels, which may bias the models toward learning unrealistic operational patterns.

To address this, we employ a stratified sampling strategy using power consumption bins, ensuring balanced representation across all cores and memory banks with respect to both frequency of appearance and the range of power levels observed. As illustrated in Fig. 7.5, this procedure mitigates bias and supports robust generalization. Notably, due to the extensive mixing and shuffling involved, the final dataset contains no application-level identifiers, eliminating potential bias toward specific workloads or application mixes. The corresponding steady-state temperature values are then recomputed using the HotSpot simulator [100], and used as the ground truth labels for model training. Each dataset is subsequently partitioned into 75% for training and 25% for testing.

3) Single-Label Model Optimization

To meet the runtime requirements of our system, we adopt a single-label regression approach, wherein each model is trained to predict the maximum temperature observed among all cores (for $\mathrm{NN}_{\mathrm{cores}}$) or memory banks (for $\mathrm{NN}_{\mathrm{hbm}}$). This approach aligns with the thermal safety objective, ensuring that no core or bank exceeds the critical thermal threshold, while significantly reducing the computational overhead of inference, i.e., compared to predicting

Model	Topology	MAPE (%)	Inference Time (μs)	Memory Footprint (KB)
$\mathrm{NN_p}$	4 hidden layers 128, 128, 64, 64	2.3	7.9	134
$\mathrm{NN_m}$	2 hidden layers 16, 16	0.3	1.5	5
$\mathrm{NN_{cores}}$	2 hidden layers 128, 64	1.8	5.1	67
$\mathrm{NN_{hbm}}$	3 hidden layers 8, 8, 8	0.4	1.6	5

Table 7.1.: Our neural network models are both lightweight and highly accurate, allowing the migration and DVFS policies to efficiently fulfill the objective of maximizing performance while adhering to the thermal constraints of both the clustered multi-core architecture and the HBM.

all temperature values of each core and bank. The datasets are thus refined to include only the maximum temperature for each sample, and model training is conducted accordingly. Following a structured neural architecture search (as detailed in Fig. 7.4), we identify pareto-optimal configurations that balance predictive accuracy and runtime latency.

The selected $\mathrm{NN_{cores}}$ model consists of two dense hidden layers with 128 and 64 neurons, while $\mathrm{NN_{hbm}}$ is configured with three hidden layers of 8 neurons each. As detailed in Section 7.5.2, this architectural optimization results in models that are compact, efficient, and capable of delivering high-fidelity thermal predictions within microseconds—satisfying the strict timing constraints of *MTCM* 's runtime control policies, as will be demonstrated in Section 7.7.

7.6. ML-Based Resource Management for Clustered Multicores with HBM

Our proposed technique, *MTCM*, performs two core resource management policies at runtime: application migration and cluster-level DVFS, as shown in Fig. 7.6. These policies are periodically executed to maximize performance

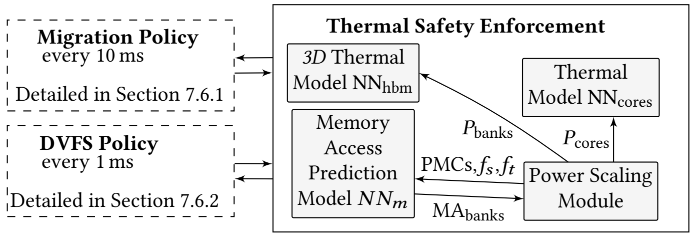

Migration Policy
every 10 ms
Detailed in Section 7.6.1
DVFS Policy
every 1 ms
Detailed in Section 7.6.2
Thermal Safety Enforcement
3D Thermal Model NN$_{hbm}$
Thermal Model NN$_{cores}$
Memory Access Prediction Model NN_m
P_{banks}
P_{cores}
PMCs,f_s,f_t
Power Scaling Module
MA$_{banks}$

Figure 7.6.: Our runtime thermal safety enforcement strategy ensures that the proposed policies implement thermally-safe resource management decisions, thereby enhancing the system's ability to fully exploit its performance potential.

while enforcing thermal safety across both the multicore and the HBM subsystems.

7.6.1. ML-Based Application Migration

In alignment with the Linux *migration epoch* [58], *MTCM* triggers its migration policy every 10 ms and performs the following steps.

1) Identifying Migration Candidates

Applications that are executing alone on a cluster are assumed to be achieving near-optimal performance at the current V/f level and are excluded from migration. Conversely, applications App$_a$ that are co-located with other applications App$_O$ on the same cluster are considered as migration candidates, as they may be suffering from cache contention. For each such candidate, potential destination clusters with sufficient core availability are identified. If migrated, App$_a$ would execute in parallel with zero or more applications App$'_O$ on the destination cluster at V/f level f_t. To support performance and thermal prediction, the pre-migration PMCs over the preceding 10 ms window, including the current IPS values of all involved applications and the frequencies f_s and f_t, are recorded.

2) Evaluating Performance Potential

For each valid migration candidate, *MTCM* evaluates the expected performance gain using the NN_p model, which predicts the post-migration IPS of App_a. A migration is considered beneficial if the predicted improvement exceeds the model's MAPE, ensuring that only statistically significant performance gains are considered. Such candidate migrations are temporarily stored for further thermal validation in the next stage.

3) Ensuring Thermal Safety

Even when performance benefits are expected, a migration may be counterproductive if it induces thermal violations on the multicore or the HBM, as highlighted in Section 7.2. To assess thermal feasibility, *MTCM* constructs a post-migration power distribution map P_{cores} based on the application-to-cluster mapping and target frequency f_t. Using the current temperatures and P_{cores}, the NN_{cores} model estimates the steady-state core temperatures. For the HBM, a more complex procedure is required since prior to migration, App_a executes on cluster s, with memory accesses routed via channel ch_s at frequency f_s, while post-migration, memory requests are rerouted to channel ch_t at frequency f_t.

To construct the updated memory access map MA_{banks}:

1. NN_{m} predicts the number of memory accesses that App_a would issue at f_t.
2. The pre-migration accesses are uniformly subtracted from banks of channel ch_s.
3. The predicted accesses are uniformly distributed over banks in channel ch_t.

Given MA_{banks} and energy-per-access values from CACTI-3DD [103], the corresponding post-migration power map P_{banks} is constructed. The NN_{hbm} model then predicts the new per-bank steady-state temperatures. A migration is considered thermally safe if no core or memory bank exceeds the defined temperature threshold T_{thresh}. Among all thermally-safe options, *MTCM* applies the migration that yields the highest predicted system performance improvement.

7.6.2. Cluster-Level DVFS

At each 1 ms *DVFS epoch*, *MTCM* evaluates and adjusts the frequency levels of each cluster to preserve thermal safety while maximizing performance. First, current power consumption maps P_{cores} and P_{banks} are used as input to the $\mathrm{NN}_{\mathrm{cores}}$ and $\mathrm{NN}_{\mathrm{hbm}}$ models to predict steady-state temperatures. If thermal violations are predicted, *MTCM* identifies the clusters with the highest temperatures. Each cluster is assigned a *thermal score* derived from the maximum temperature among its cores and the highest temperature in its corresponding HBM channel ch_k. Clusters are throttled sequentially in decreasing order of thermal score. After each throttling step (i.e., lowering the V/f level), temperatures are re-evaluated until the thermal safety of both subsystems is restored. The selected clusters are then downscaled accordingly.

If no violations are predicted and thermal headroom is available, clusters are conservatively boosted to the next higher V/f level. Importantly, predicting the HBM power map P_{banks} for potential frequency increases requires invoking NN_m to estimate the new memory access rate for each application. This tight integration of machine learning predictions ensures that *MTCM* remains both thermally safe and performance-aware during runtime operation, as will be demonstrated in Section 7.7.

7.7. Experimental Evaluation

We evaluate the proposed *MTCM* framework using the CoMeT simulator [99], as described in Section 3.2.2. Our simulated platform consists of the clustered 64-core clustered processor (experimental setup (A) in Section 3.1.1), integrated with an 8-channel HBM2E [102] memory system (experimental setup (B) in Section 3.1.1). A thermal threshold of 80 °C is imposed on both the multicore and the HBM subsystems. When this limit is exceeded on any core, the DTM mechanism is triggered, reducing the V/f level of all cores to the minimum. Likewise, if a thermal violation is detected on any HBM bank, the corresponding memory channel enters a low-power state until thermal safety is restored.

We select a diverse set of benchmark applications from the *PARSEC* [49] and *SPLASH-2* [50] suites, as listed in Appendix A. These include: *blackscholes*, *bodytrack*, *canneal*, *streamcluster*, *fluidanimate*, *swaptions*, *x264*, *barnes*, *cholesky*, *fft*, *fmm*, *lu.cont*, *lu.ncont*, *radix*, *raytrace*, *water.nsq*, and *water.sp*. Each of these applications can be executed at 2, 3, 4 or 5 parallel threads, leading to a total of 68 unique applications. From this pool, we construct three distinct workloads, each consisting of 40 randomly-selected applications. Application arrival times within each workload are drawn from a Poisson distribution, using four different arrival rates: 120, 140, 160, and 180 applications per second. Each workload is evaluated under all four arrival rates. This setup produces diverse system utilization scenarios, varying in both CPU and memory activity, including differences in cluster occupancy, cache contention, and memory bank access intensity. Importantly, these runtime conditions expose our NN-based models to runtime configurations that were not observed during training. Specifically, the evaluation includes new combinations of application co-location per cluster, thread-level parallelism, core utilization patterns, cache access behaviors, and HBM channel usage. This comprehensive evaluation thus enables a robust assessment of the generalization capabilities and runtime effectiveness of *MTCM* under realistic and varied workloads.

7.7.1. Comparison Techniques

We evaluate the performance of our proposed *MTCM* against two comparison techniques described below.

***NeuroMap* [98]** This technique represents the most relevant state-of-the-art resource management approach for systems integrating clustered multicores with HBM. *NeuroMap* employs task migration and cluster-level DVFS to enhance system performance under thermal constraints, but exclusively focuses on the HBM subsystem, disregarding the thermal state of the multicore processor. The method first assigns memory channels to predefined core groups, a strategy compatible with our target architecture, where clusters inherently share a common memory controller and associated HBM channel. At runtime, *NeuroMap* periodically adjusts application-to-cluster mappings using rule-based heuristics driven by application behavior. For instance, memory-intensive applications are preferentially migrated to clusters linked to cooler channels (i.e., those located physically closer to the heat sink), whereas compute-intensive applications with low memory demands may be allocated to clusters connected to hotter channels near the bottom of the

HBM stack. When an ideal placement is not found, *NeuroMap* uses DVFS to either boost or throttle frequency levels. However, all decisions are made without considering the thermal state of the processor cores.

***NeuroDTPM* [98, 132]** None of the existing techniques in the literature has jointly addressed the thermal constraints of both the multicore and the HBM subsystems while aiming for performance maximization. To enable a fair comparison, we construct an augmented version of *NeuroMap* by integrating it with *DTPM* [132], a well-established thermal-aware DVFS policy. *DTPM* aggressively scales frequency levels to maximize performance under a processor temperature constraint. The resulting hybrid technique, *NeuroDTPM*, now considers thermal constraints in both subsystems—the multicore and the HBM, thus aligning with the optimization goals of *MTCM*.

For all techniques, we adopt a uniform thermal control mechanism:

- Temperature violations on the HBM trigger the affected memory channels to enter a low-power state until safe operating conditions are restored.
- Violations on the multicore result in all cores being throttled to the minimum supported V/f level, i.e., 1 GHz.

7.7.2. Evaluation Results

We present a comparative evaluation of our proposed *MTCM* against the two state-of-the-art techniques: *NeuroMap* and *NeuroDTPM*. The comparison is conducted in terms of system performance, thermal behavior, and thermal efficiency. Figure 7.7 summarizes the performance and temperature results across all workloads and arrival rates.

Comparison with *NeuroMap* Our proposed *MTCM* consistently outperforms *NeuroMap*, achieving an average performance improvement of 1x% across all experimental configurations. This superior performance can be attributed to two main factors.

First, *NeuroMap* employs a coarse-grained DVFS policy, which assigns a low, medium, or high V/f level to each cluster based on the current memory-intensity phase of the executing applications. This rule-based heuristic fails to exploit finer opportunities for performance maximization that our *MTCM*'s fine-grained, ML-guided DVFS policy effectively captures. This distinction is clear in the box plots of Fig. 7.7, where our technique more effectively utilizes available thermal headroom across both the multicore and the HBM.

Second, although *NeuroMap* successfully maintains a thermally-safe HBM operation for over 99.5% of the execution time, it lacks awareness of the thermal state of the multicore. As a result, frequent violations of the multicore thermal threshold occur, triggering the TCC, which in turn throttles all clusters to the minimum frequency, degrading overall performance. This behavior validates the design rationale presented in Section 7.2, emphasizing the necessity of jointly managing the thermal states of both subsystems.

Comparison with *NeuroDTPM* To address the thermal blind spots of *NeuroMap*, the augmented baseline *NeuroDTPM* incorporates the thermal-aware *DTPM* [132] DVFS policy. As expected, this leads to a significant improvement in thermal safety and overall system performance. The system maintains thermally-safe operation in both the multicore and HBM subsystems for 99.1% of the execution time. This is reflected in Fig. 7.7, where *NeuroDTPM* shows more effective utilization of multicore thermal headroom, with only minor thermal violations (less than 1% of the time), typically in the upper outliers of the 99th percentile, resulting from sudden transient power surges. These improvements reduce the frequency of TCC activation, thereby preserving performance gains achieved via task migration. Nevertheless, *MTCM* still achieves superior performance, delivering an average performance improvement of 25.4% over *NeuroDTPM*.

The key distinction lies in the migration policy: while *NeuroMap* and its augmented variant focus on thermal considerations alone, our *MTCM* explicitly incorporates cache contention effects. By selecting application-to-cluster mappings that minimize contention among co-executing applications, *MTCM* reduces slowdown and further enhances system throughput. This reinforces our initial claim that ignoring cache contention in heterogeneous memory hierarchies leads to suboptimal resource management decisions.

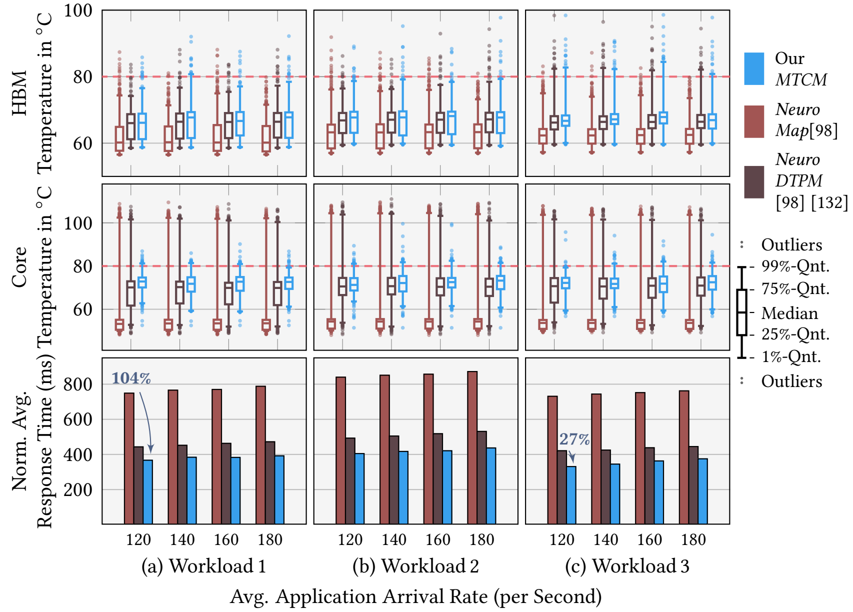


Figure 7.7.: Our *MTCM* achieves significant performance improvements—up to 1× across multiple workloads—when compared to state-of-the-art techniques, highlighting the critical role of jointly addressing cache and thermal interference in optimizing the performance of modern systems featuring clustered multicores and HBM. Moreover, *MTCM* consistently operates both the multicore and the HBM closer to the system's thermal limits than comparison approaches, thereby more effectively leveraging the system's performance capabilities.

Thermal Efficiency Analysis To further substantiate our findings, we report the thermal efficiency achieved by each technique in Fig. 7.8, quantified as the average number of millions of instructions executed per second per degree Celsius. The trend remains consistent with previous observations: *MTCM* demonstrates the highest thermal efficiency across all workloads and arrival rates. In contrast, *NeuroMap* exhibits the lowest thermal efficiency due to prolonged execution times, lower IPS, and frequent thermal violations on the multicore. *NeuroDTPM* achieves significantly higher thermal efficiency than *NeuroMap*, benefiting from its integration of a fine-grained DVFS scheme and improved thermal safety.

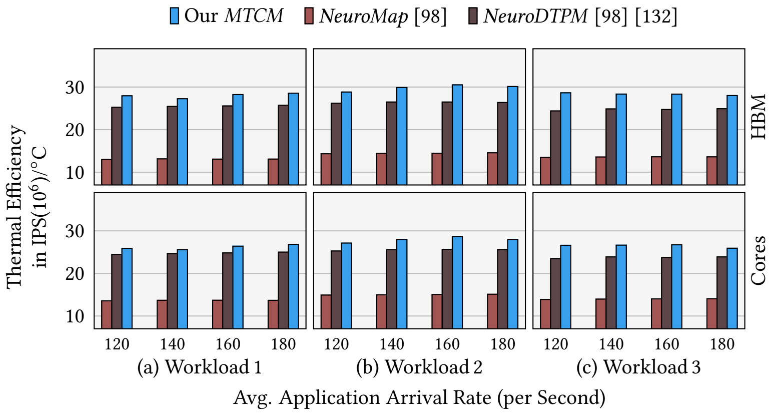


Figure 7.8.: Compared to the two state-of-the-art techniques, our *MTCM* consistently achieves higher instruction throughput per unit of temperature, thereby delivering enhanced thermal efficiency across all workloads and arrival rates.

Overall, these results confirm that jointly accounting for cache contention and thermal constraints across both the multicore and HBM subsystems enables *MTCM* to better unlock performance potential of the system while preserving thermal integrity.

7.7.3. Generalization Analysis

As demonstrated in the previous section, *MTCM* consistently outperforms state-of-the-art baselines (*NeuroMap* and *NeuroDTPM*), achieving substantial performance gains while maintaining thermal safety for both the multicore and the HBM, even under previously unseen workloads. To further examine the generalization capability of our machine learning models, we conduct a series of *Leave-Group-Out* (GroupKFold) cross-validation experiments. In each of the K iterations, the dataset is shuffled and partitioned such that each fold includes a distinct group of applications in the validation set that were excluded from the training set in that iteration. This setup ensures that validation always occurs on unseen application groups. It is important to note that the thermal models $\mathrm{NN}_{\mathrm{cores}}$ and $\mathrm{NN}_{\mathrm{hbm}}$ are not considered in this analysis, as their training datasets are synthetically generated and lack application-specific per-core or per-bank mappings. For the contention-aware

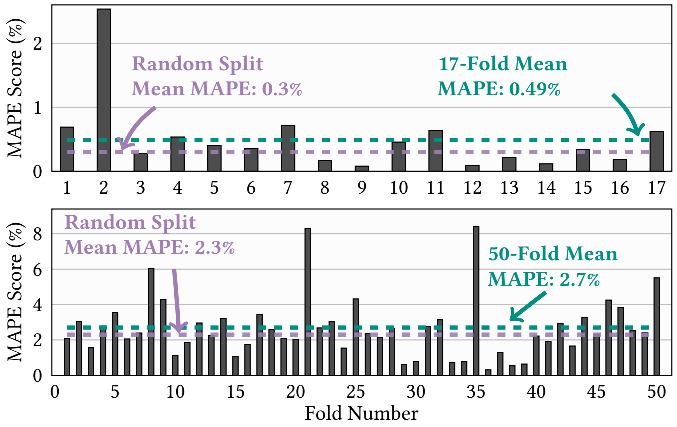


Figure 7.9.: Using a dataset comprising 17 distinct applications, a leave-group-out cross-validation—employing a 17-Fold setup for NN_m and a 50-Fold setup for NN_p, with random application groups excluded in each iteration—demonstrates that our models exhibit minimal performance degradation on unseen applications across the folds. The mean MAPE is only slightly higher than that obtained through a random training/test split. This highlights the application-independence of our models and their strong generalization capability to unseen traces, as further evidenced by their performance on larger workloads and novel scenarios.

models NN_m and NN_p, we use the *LeaveOneGroupOut* approach from the *scikit-learn* library. The model NN_m is trained with a dataset limited to single-application traces, allowing a 17-fold evaluation corresponding to the 17 unique applications. For NN_p, we construct 50 folds, each containing up to three applications—reflecting real-world migration scenarios where App_a is migrated from a source to a destination cluster, possibly involving other co-running applications. This setup allows us to validate model generalization in scenarios where up to 24% of applications are unseen during training.

Figure 7.9 illustrates the error distributions obtained across folds. For NN_m, we observe stable MAPE values across most folds, with the exception of fold $K = 2$, which excludes *canneal*, a long-running application that contributes disproportionately to the dataset. Despite this, the mean MAPE remains low at 0.49%, only slightly higher than the 0.3% achieved with the full dataset, as reported in Section 7.5.

For NN_p, the error trends are similar. For instance, fold $K = 36$, comprising less complex scenarios such as migrations to empty clusters, shows a notably lower MAPE. In contrast, fold $K = 21$, which excludes *canneal*, exhibits higher prediction error due to increased behavioral complexity. Overall, the model maintains a mean MAPE of 2.7%, which is only 0.4% above the result from the randomly partitioned dataset. These findings confirm the robustness and generalizability of our models under realistic deployment conditions.

7.7.4. Overhead Analysis

Policy	Model	Latency (µs)	Model Description
Migration	NN_p	64.5	Performance prediction
	NN_m	7.5	Memory access prediction
	NN_{cores}	25.5	Thermal model for CPU cores
	NN_{hbm}	8.0	Thermal model for HBM stack
DVFS	NN_m	4.4	Memory access prediction
	NN_{cores}	15.3	Thermal model for CPU cores
	NN_{hbm}	4.8	Thermal model for HBM stack

Table 7.2.: Inference latency and detailed description of each NN model employed in our migration and DVFS policies. The models' lightweight nature, high accuracy, and minimal memory footprint enable both policies to execute within their designated epoch lengths, incurring negligible runtime overhead.

To assess the runtime overhead of *MTCM*, we perform additional experiments in which the complete resource management policy is deployed as a single-threaded application pinned to one core on the target system, operating at the maximum supported V/f level. The same randomly-generated workloads and arrival rates used in the main evaluation are reused here. Section 7.7.4 reports the average execution time per epoch for each policy component across all experiments. On average, the DVFS policy of *MTCM* requires 24.6 µs to complete, which constitutes only 2.46% of the 1 ms DVFS control window on a single core. Similarly, the migration policy requires 105.5 µs on average to evaluate and apply a migration decision—corresponding to just 1.05% of the 10 ms migration epoch. These measurements confirm that the benefits of *MTCM* are realized with negligible runtime overhead, making it a lightweight yet effective solution for deployment on high-performance 64-core systems.

7.8. Summary

This chapter presented *MTCM*, the first resource management technique in the literature that considers cache contention in maximizing system performance, while maintaining thermal safety on modern systems with clustered multicores and HBM.

Tackled Challenges While the five challenges in Section 1.3 were successfully tackled, Challenge 5 in Section 1.3.5 was particularly more pronounced in this work. In fact, the state-of-the-art thermal simulators/predictors are too slow, i.e., **millisecond-level** predictions, to be considered for temperature prediction at runtime in systems with a 128-bank 3D-stacked HBM and a 64-core processor. This made ensuring thermal safety proactively across the HBM and the processor, before any RM action, particularly challenging within the 10 ms epoch length. The challenge was successfully addressed using our proposed NN-based thermal models, which delivered highly accurate **microsecond-level** temperature predictions for both subsystems.

Achievements and Takeaways This work has showcased how orchestrating four ML-based models enabled *MTCM*'s migration and DVFS policies to achieve its goal. *MTCM* achieves substantial performance improvements compared to the state-of-the-art, confirming the significance of jointly considering contention and temperature problems, which we motivated in Chapter 1.

Insights In this work, the HBM is an off-chip memory subsystem with no heat coupling with the processing cores. Recently, 2.5D integration has been rising in popularity, whereby the processor and the HBM are packaged together on the same interposer. Depending on the *beachfront* distance between the edge of the HBM and that of the processor—typically around 2 mm—heat transfers occur between the two subsystems. This architectural change would necessitate modifications to the training methodology of our proposed thermal models, as predicting the steady-state temperature of either subsystem would require considering the temperature of the other as an input. It would also be logical to have a joint model (instead of two) that predicts the maximum temperature across both subsystems, which would be sufficient to enforce thermal safety in our proposed RM technique.

8. Energy Efficiency Forecasting for Clustered Heterogeneous Processors

This chapter introduces EffiCast, the first methodology for forecasting energy efficiency in clustered heterogeneous processors with contention awareness, employing sequence-based ML models. Through comprehensive experimental analysis into the sensitivities of energy efficiency across various core types, V/f level configurations, application execution phases, and resource contention conditions, EffiCast identifies the principal contributors to energy efficiency variability in modern heterogeneous platforms. By systematically leveraging generated data and leveraging advanced sequence models based on LSTM and Transformer architectures, EffiCast demonstrates superior predictive accuracy, surpassing existing state-of-the-art approaches. Implemented on a real-world heterogeneous processor enhanced with Intel's oneDNN library, EffiCast achieves inference latencies as low as 1.82 ms per sequence, enabling seamless integration in proactive RM techniques.

The need for energy-efficient computing has become increasingly critical in modern systems [133], including smart industries, autonomous vehicles, and similar domains, due to the necessity of operating under strict energy constraints while sustaining adequate performance. At the same time, workloads are becoming more diverse and complex, varying significantly in terms of computational demands and memory usage. This variability adds complexity to the objective of energy-efficient operation, as different applications require resource management strategies that are adapted to their specific profiles. Heterogeneous processors, such as the heterogeneous Intel *Alder Lake* Architecture [29] introduced in Section 3.1.1, offer a compelling architectural response to this issue by integrating high-**P**erformance and energy-**E**fficient cores. To leverage this heterogeneity, system-level RM should dynamically assign applications to the most appropriate core type based on their individual characteristics. Nevertheless, as illustrated in the following motivational example, selecting the core type that yields the best energy efficiency for a given application is a non-trivial task, particularly when combined with V/f scaling.

8.1. Motivational Example

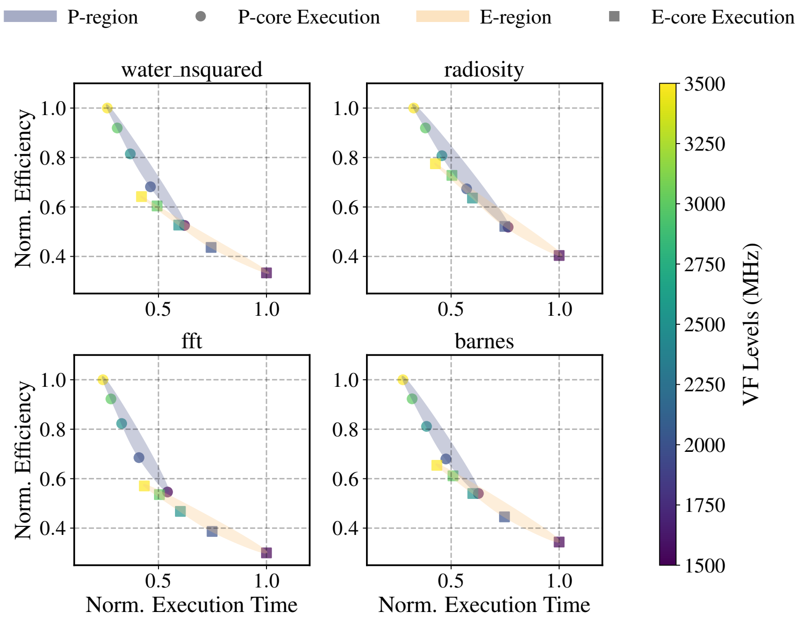


Figure 8.1.: Executing applications on a heterogeneous processor highlights distinct sensitivities in performance and energy efficiency to variations in core types and V/f levels. Identical color coding denotes the same V/f level across both core types, while different colors represent different V/f levels. *fft* displays pronounced sensitivity to both core type and V/f level, achieving superior energy efficiency on the P-cores, whereas *radiosity* follows substantially different trends.

Figure 8.1 presents the execution time and system-wide energy efficiency of two applications from the PARSEC [49] and SPLASH-2 [50] benchmark suites, evaluated across varying V/f levels and core types on the heterogeneous Intel® Core™ i9 12th generation Alder Lake processor (details in Section 3.1.1). The reported values are normalized relative to the slowest execution time and the highest energy efficiency observed for each respective application. Energy efficiency is measured in terms of instructions per Joule (IPJ), defined as the total number of instructions executed per Joule of system-wide energy consumption; higher IPJ values indicate more energy-efficient executions.

Applications, *fft*, *water_nsquared* and *barnes* display a clear sensitivity to both the core type and the V/f configuration, achieving its *lowest execution time* and *highest energy efficiency* when executed on a P-core. In contrast, the behavior of the *radiosity* application is notably different: several configurations executed on E-cores yield performance and energy efficiency comparable to those on P-cores. Notably, under mid-range V/f settings, the E-core execution of *radiosity* surpasses the P-core execution in terms of energy efficiency. This observation highlights the limitations of heuristic-based task mapping and DVFS strategies, which often fail to adapt to the *nuanced* energy-performance trade-offs presented by diverse workloads.

Consequently, more sophisticated RM methods are required. This need is increasingly acknowledged in recent work. ML-based and predictive approaches are gaining traction as essential tools for proactively estimating the *effects* of selecting specific core types and/or V/f levels. For instance, earlier efforts have explored the use of code instrumentation [119, 120] for application characterization. This enabled the development of an imitation learning-based runtime optimization technique in [118], which accounted for the influence of core heterogeneity on energy consumption. However, such techniques depend on prior knowledge of the application and access to its source code—assumptions that are typically *unrealistic* in modern computing environments. Other studies have adopted supervised learning models to predict how application performance and power consumption respond to V/f changes [134, 8], though these were evaluated exclusively on homogeneous processors.

Recognizing this limitation, [135] introduced LSTM-based forecasting for power consumption. However, their approach was validated only on simulated homogeneous processors, limiting its applicability to real-world heterogeneous systems. In contrast, our proposed EffiCast methodology introduces a novel forecasting approach to predict the energy efficiency sensitivity of applications to both core types and V/f levels on real heterogeneous processors.

This approach not only accounts for complex phase-level and contention dynamics but also generalizes to unseen runtime conditions. As will be further elaborated in Sections 8.2 and 8.3, the challenge of predicting energy efficiency sensitivity with respect to *both* core type and V/f level remains an open challenge in current literature.

8.2. Challenges and Novel Contributions

Predicting energy efficiency in heterogeneous processors is a highly-complex problem, due to several intertwined challenges. **First**, as illustrated in Fig. 8.1, both energy and performance sensitivities exhibit variability depending on the application characteristics and the specifics of the underlying hardware. Even when applications are well characterized and profiled, determining the optimal combination of core type and V/f level can remain non-trivial, as exemplified by the case of *radiosity*. At runtime, this challenge is further exacerbated when applications are previously unseen, rendering prior profiling infeasible. In the absence of such prior knowledge, accurately forecasting energy efficiency necessitates the use of predictive models that are sufficiently robust to generalize across diverse, unseen workloads and runtime conditions.

Secondly, application behavior is not uniform throughout execution; instead, applications progress through distinct phases, each characterized by different resource usage patterns and varying sensitivity to architectural heterogeneity. As will be shown in the analysis section (Section 8.4), a single application may exhibit significantly different energy efficiency behaviors across its execution phases, necessitating predictive models that can effectively account for this intra-application variability.

Third, and more critically, is the rapid rate at which applications may shift between these phases. When attempting to predict the consequences of runtime decisions—such as changing the V/f setting or migrating an application across core types—the system state may evolve before the action is applied, thereby invalidating the prediction. Traditional machine learning methods that depend on static or *short-horizon* input representations are inadequate for modeling such dynamics. As will be demonstrated in Section 8.6, accurately forecasting future system states is essential to enabling proactive and effective decision-making during runtime.

Finally, heterogeneous processors incorporate complex hierarchies of *shared* resources, including cache subsystems and memory bandwidth, which can introduce contention both within and across clusters. As discussed in Section 1.3, such contention is inherently dynamic, being influenced not only by the applications co-located on the same cluster, but also by the phase transitions of applications executing on other clusters. Capturing the resulting contention effects and their implications for energy efficiency adds another layer of difficulty to the prediction problem.

To address these intertwined challenges, we begin by conducting a comprehensive analysis to identify the key factors that affect the predictability of application energy efficiency in heterogeneous processors. Building upon the insights obtained, we introduce EffiCast, a new methodology for contention-aware energy efficiency forecasting based on sequence modeling techniques.

In summary, our novel contributions are:

- We perform a comprehensive analysis of energy efficiency sensitivities across core types, V/f levels, application phases, and resource contention scenarios, uncovering key factors that impact energy efficiency prediction.
- We propose EffiCast, the first methodology for training energy efficiency forecasting models, incorporating structured data generation and sequence-based techniques to predict future system states under dynamic and unseen scenarios.
- We build, train, and evaluate LSTM- and Transformer-based forecasting models on a real heterogeneous processor platform, demonstrating superior accuracy over traditional state-of-the-art predictive models.

8.3. Related Work

A significant body of research has investigated application sensitivity to processor and system characteristics. Early works such as [136] proposed neural network models to optimize application-to-core mappings for minimizing energy consumption, leveraging design-time application profiles. Similarly, [137] used linear regression modelsto guide runtime application mapping decisions based on these profiles. While these techniques have considered the contention effects on the processor, they have relied on coarse-grained, whole-execution profiles, missing the finer optimization opportunities available through phase-level analysis, as demonstrated previously in Section 8.4.

The importance of phase-level sensitivity was first highlighted in [138, 139], which demonstrated the benefits of adaptive application-to-core mapping adjustments at runtime. Although validated on simulated heterogeneous processors, these studies established the foundation for subsequent approaches. For example, [118] applied imitation learning on a real heterogeneous big.LITTLE platform to optimize performance per watt, using LLVM-based benchmarking to generate training data. Similarly, [119, 140] leveraged LLVM instrumentation to train classifiers that identify phase-level Pareto-optimal configurations at runtime. Compiler-based techniques were also explored in [120], where Java bytecode analysis via the Soot framework triggered runtime configuration changes. In [121], Linux kernel-level tracepoint instrumentation was employed to predict phase-level energy-efficient configurations. However, all these methods depend on source code access and recompilation, making them unsuitable for scenarios where source code is unavailable or recompilation is infeasible.

To address these limitations, performance counters have emerged as a non-intrusive alternative for runtime data collection without requiring source code access. In [8], phase-level performance counter data was used to train machine learning models for predicting application sensitivities to V/f level changes, enabling smart and proactive DVFS at runtime. Similar methods in [2, 22] predicted phase-level performance sensitivities for task migration. However, these works have been validated only in simulated environments, focusing solely on sensitivity to V/f levels or single core types. Furthermore, these methods assume that sensitivity remains static in the immediate subsequent epoch, a limitation demonstrated in Section 8.4.

Recognizing this limitation, [135] introduced LSTM-based forecasting for power consumption. However, their approach was validated only on simulated homogeneous processors, limiting its applicability to real-world heterogeneous systems. In contrast, our proposed EffiCast methodology introduces a novel forecasting approach to predict the energy efficiency sensitivity of applications to both core types and V/f levels on real heterogeneous processors.

This approach not only accounts for complex phase-level and contention dynamics but also generalizes to unseen runtime conditions.

8.4. Analyzing Energy Efficiency Sensitivity to Heterogeneity

To build upon the observations and challenges outlined in Section 8.1, this section presents a detailed analysis of energy efficiency sensitivities with respect to core types, V/f levels, execution phases of applications, and resource contention conditions. The experiments are conducted using the PARSEC [49] and SPLASH-2 [50] benchmark suites executed on a Linux-based heterogeneous Intel® Core™ i9 12th generation Alder Lake processor (experimental setup (C) in Section 3.1.1), where PMCs are sampled every 100 ms via *perf* [77], and system-wide energy consumption is recorded at the same interval using the RAPL interface.

8.4.1. Volatility of Phase-Level Sensitivities

We begin by examining the phase-level sensitivity of the *radiosity* application to core types and V/f settings, when executed in parallel with multiple background applications on the chip, in relation to the earlier findings presented in Fig. 8.1, where the application was executed on a fixed core type across its entire runtime. Figure 8.2 shows the cumulative count of retired instructions and the corresponding energy consumption sampled at periodic intervals, for *radiosity* running at 3.5 GHz on both a P-core and an E-core.

To facilitate phase-level analysis, the execution is segmented into slices, following our PHASEL [5] approach, each comprising 1.6 billion retired instructions. Due to variations in the IPS, the duration of each phase differs, resulting in variable execution times. Since both configurations process the same total number of instructions, slices are aligned between P-core and E-core runs, with color-coding used to simplify visual comparison. Annotated labels highlight the energy efficiency gain of the E-core execution relative to the P-core execution within the same phase.

While Fig. 8.1 indicated that P-cores were generally more energy-efficient when executing the entire application, Fig. 8.2 reveals that certain individual

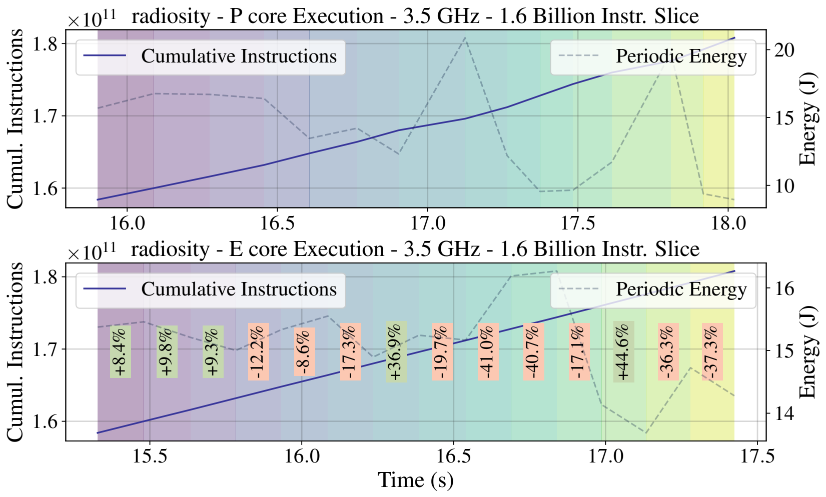


Figure 8.2.: A zoomed-in view of phase-level energy consumption over time—using phases of 1.6 billion retired instructions—for *radiosity* reveals improvements in energy efficiency at the phase level on the E-core under contention scenarios, as indicated by the colored labels. This observation contrasts with the initial finding in isolation execution in Fig. 8.1, where the overall execution of applications appeared consistently more efficient on the P-cores.

phases achieve **up to 45% better energy efficiency** on the E-core under contention scenarios. At runtime, before reassigning an application to a different core or adjusting the V/f level, a resource management mechanism must be capable of *evaluating* whether the change will enhance energy efficiency for the application's current phase. To support such *proactive* decision-making, it is necessary to accurately predict the *phase-level energy efficiency sensitivities* of applications.

8.4.2. Temporal Variability in Energy Efficiency

The temporal dynamics of energy consumption and application performance introduce complex patterns and fluctuations in energy efficiency, which in turn pose significant challenges to its prediction at runtime. Figure 8.3 presents the absolute energy efficiency, measured in IPJ, alongside the percentage variations in energy efficiency over one and two successive epochs for

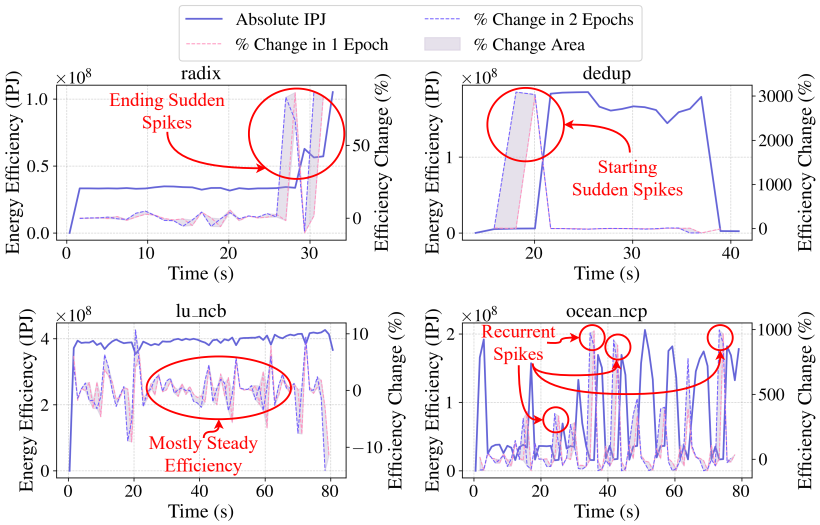


Figure 8.3.: Absolute energy efficiency (IPJ) and percentage variations over one and two epochs for four representative applications. Shaded regions indicate fluctuations in energy efficiency, capturing dynamic behaviors such as initial spikes in *dedup*, recurring spikes in *ocean_ncp*, and sudden phase shifts in *radix*. These varied patterns emphasize the need for sequence-based models that can effectively capture temporal dependencies and dynamic phase transitions.

four representative applications: *radix*, *dedup*, *lu_ncb*, and *ocean_ncp*. These data were derived by processing periodic runtime measurements, where energy efficiency is defined as the ratio of instructions retired to the total system energy consumed.

Figure 8.3 highlights three key aspects of the short-term temporal variability in energy efficiency. The dashed lines depict the percentage change in energy efficiency over the next one and two epochs, offering additional insight into the magnitude and frequency of fluctuations. The shaded region represents the range between minimum and maximum percentage changes, thereby quantifying the variability envelope.

The figure demonstrates that application-specific patterns emerge clearly. *lu_ncb* maintains a relatively consistent efficiency profile, with minimal deviations, keeping percentage changes below 10% throughout its execution. In

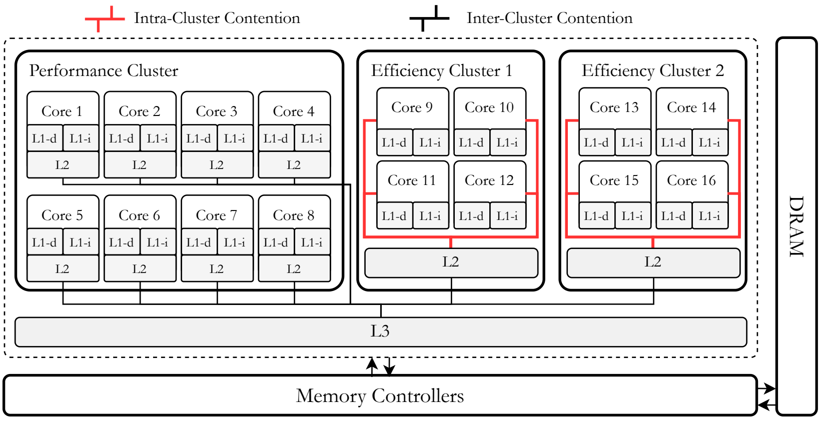


Figure 8.4.: Illustration of the heterogeneous Intel® Core™ i9 12th generation Alder Lake processor, emphasizing the complexity of contemporary memory hierarchies. The shared L2 cache within each E-cluster and the shared L3 cache across all clusters introduce both intra-cluster and inter-cluster contention, respectively.

contrast, both *radix* and *dedup* show mostly stable absolute IPJ values, but exhibit abrupt spikes, early in execution for *radix*, and late in execution for *dedup*. In the case of *dedup*, the observed spike in energy efficiency exceeds 3000%. The behavior of *ocean_ncp* is particularly distinct, showing recurring peaks in energy efficiency indicative of periodic execution patterns with alternating characteristics, and displaying relative IPJ changes surpassing 1000%.

8.4.3. Intra-Cluster and Cross-Cluster Contention

The challenge of accurately predicting energy efficiency in heterogeneous multi-core processors is further exacerbated by dynamic contention occurring at both intra-cluster and cross-cluster levels. Figure 8.4 depicts the shared resource architecture of the target processor, highlighting potential sources of contention. Within clusters, resource sharing—such as contention for the L2 cache in E-core clusters—introduces *intra-cluster* interference. Simultaneously, *cross-cluster* contention occurs at the shared L3 cache, which is

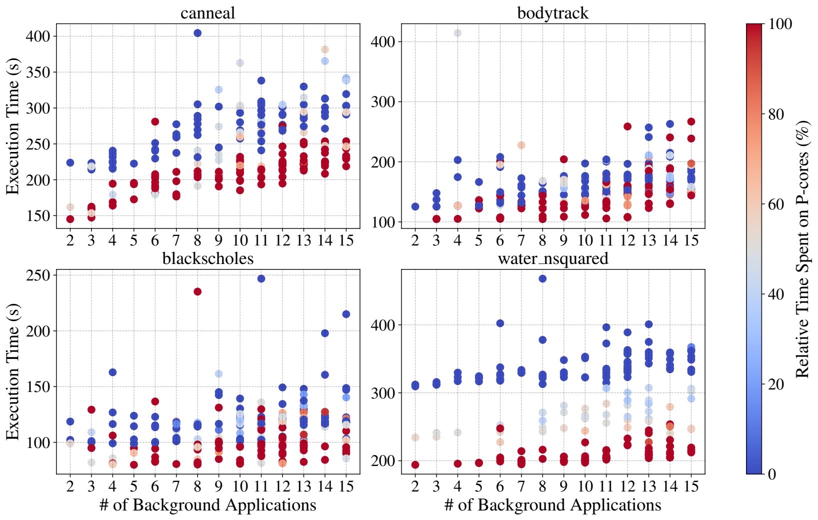


Figure 8.5.: Execution time of four applications under varying numbers of background applications. Random core mapping and periodic migrations expose the effects of intra- and cross-cluster contention, with some applications (*canneal*, *water_nsquared*) exhibiting consistent yet varying sensitivity across core types, while others (*bodytrack*, *blackscholes*) show pronounced variability. These results underscore the intricate nature of contention in heterogeneous processors.

accessible by all cores across clusters. As discussed in Section 1.3, this hierarchical and dynamic contention model introduces significant complexity in predicting energy sensitivity, as the behavior of an application is shaped not only by its own resource demands but also by the activity of co-located workloads.

To investigate the implications of such contention, we conducted a set of experiments involving up to 16 concurrently executing applications. Figure 8.5 presents the execution times of four representative applications, *canneal*, *bodytrack*, *blackscholes*, and *water_nsquared*, under varying background workload intensities. In each experiment, the applications were initially assigned randomly to cores, and the application of interest was periodically migrated across different core types and clusters. Data points are color-coded according to the proportion of execution time spent on P-cores, which are typically associated with higher performance.

Several important observations can be drawn from the results. First, execution time does not exhibit a linear relationship with the number of co-running applications, as illustrated by the behavior of *canneal* and *water_nsquared*. This suggests that performance degradation is more closely linked to the nature of the interfering workloads than to their quantity. Second, for applications such as *water_nsquared* and *canneal*, execution on P-cores consistently yields better performance across different contention scenarios. Conversely, *bodytrack* and *blackscholes* display pronounced variability between P-core and E-core executions, with performance outcomes strongly affected by both resource contention and core migration patterns.

In conclusion, this analysis highlights that accurate modeling of energy efficiency sensitivity in contemporary heterogeneous processors necessitates advanced *forecasting* models. Such models must be capable of capturing the *temporal dynamics* of *phase-level sensitivities*, while accounting for the effects of both *intra-cluster* and *cross-cluster* contention.

8.5. Problem Formulation

The objective of the forecasting task is to estimate the energy efficiency of a clustered, heterogeneous multi-core processor system operating under dynamic workload conditions. Formally, let E denote the number of observed epochs, and let $\mathcal{X}_e = \{\mathbf{X}_e^k\}$ represent the set of input features at epoch e, where each $\mathbf{X}_e^k \in \mathbb{R}^d$ corresponds to the features of the k-th application, as well as system-level metrics. The energy efficiency of the system at epoch e is denoted by U_e, expressed in IPJ units.

Given the input feature sequence $\{\mathcal{X}_e, \mathcal{X}_{e+1}, \ldots, \mathcal{X}_{e+E}\}$, the goal is to forecast the system's energy efficiency at epoch $e + E + 1$, denoted U_{e+E+1}, *under a prospective configuration change* $\mathbf{G}_{e+E+1}$, where:

- $\mathbf{G}_{e+E+1}$ specifies the proposed application-to-core mapping and the selected V/f level.
- Each $\mathcal{X}_e$ encapsulates application-level features for the application of interest, aggregated metrics from individual clusters, and global system metrics.

The forecasting objective is to minimize the prediction loss $\mathcal{L}$, defined as the mean squared error between the actual energy efficiency value U_{e+E+1} and the predicted value $\hat{U}_{e+E+1}$, averaged over all $\mathcal{N}$ training samples:

$$\mathcal{L} = \frac{1}{\mathcal{N}} \sum_{j=1}^{\mathcal{N}} \left(U_{e+E+1}^{(j)} - \hat{U}_{e+E+1}^{(j)} \right)^2 .$$

8.6. EffiCast: Novel Forecasting Methodology

The formulation and training process of EffiCast directly address the challenges outlined in Section 8.2, while being guided by the experimental observations presented in Section 8.4. The following subsections describe the machine learning formulation, training data generation strategy, feature engineering process, and model architectures employed in our approach.

8.6.1. Training Data Generation

To construct a dataset for training the forecasting models, we perform a series of experiments using applications from the PARSEC and SPLASH-2 benchmark suites. Each experiment randomly selects a subset of applications and runs them on a Linux-based heterogeneous Intel® Core™ i9 12th generation Alder Lake processor. PMCs are collected at 100 ms intervals using *perf* [77], while system-wide energy consumption is simultaneously monitored via the RAPL interface. Energy efficiency is measured in terms of IPJ, representing the total number of instructions retired per unit of energy consumed.

The experiments span a range of V/f settings, from 1.5 GHz to 3.5 GHz. Each workload comprises between 2 and 16 applications, concurrently utilizing the 16 available cores on the platform. The cores are partitioned into three clusters: a performance (P) cluster, and two energy-efficient clusters (E1 and E2). The E clusters share clock domains, whereas the P cluster allows for per-core DVFS adjustments. At every T epochs, one application is selected as the application of interest. This application is migrated to a different core or cluster, and its V/f level is modified. The system's energy efficiency, measured at the end of the T epochs, serves as the *label* for training. The input *features* consist of performance metrics from the preceding T epochs, along with the

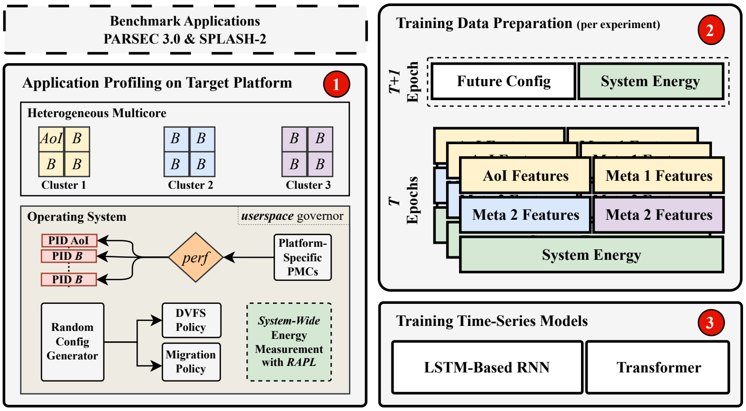


Figure 8.6.: An overview of EffiCast, our novel methodology for contention-aware, sequence-based forecasting of energy efficiency in heterogeneous processors.

newly applied configuration—specifically, the updated core mapping and V/f level.

8.6.2. Feature Selection and Engineering

Accurate forecasting of energy efficiency in heterogeneous processors necessitates a structured feature set that captures temporal, spatial, and application-level dynamics. The feature representation characterizes the behavior of the application of interest (AoI), as well as its interaction with co-running applications, shared hardware resources, and the overall system environment. These features are organized hierarchically across three levels: individual application metrics, cluster-level aggregated metrics, and system-wide energy measurements.

At the application level, fine-grained PMCs are extracted for the AoI, including metrics such as the number of instructions retired, CPU cycles, cache accesses and misses, and branch instructions and mispredictions. Cache-related metrics are interpreted differently depending on the core type: for E clusters, they predominantly reflect L2 cache activity, whereas for P cores, they pertain primarily to L3 cache interactions. This distinction enables the modeling of core-specific contention behaviors. For example, a high L2 miss rate in an E cluster may result in delays due to subsequent L3 cache misses, whereas P cores—when encountering L3 misses—access main memory directly, resulting in different performance penalties. Such features capture both execution characteristics and contention effects across the memory hierarchy.

At the cluster level, aggregated metrics from co-running applications are incorporated. Metrics from applications co-located with the AoI on the same cluster constitute the "Meta Application 1" feature group, capturing *intra-cluster contention*, such as L2 cache contention in E clusters or L3 contention in the P cluster. Metrics aggregated from the other two clusters form the "Meta Applications 2 and 3" groups, capturing *cross-cluster contention* effects, such as competition for shared L3 cache and memory bandwidth resources.

At the system level, system-wide energy consumption is also included as a feature, providing a global perspective on energy dynamics under varying workload conditions. Additionally, the core type and V/f level corresponding to the hypothetical next-epoch configuration are encoded, enabling the model to generate predictions under dynamic runtime scenarios, including task migration and frequency scaling.

Feature selection is guided by both domain knowledge and empirical data analysis. Metrics that exhibit strong correlation with energy efficiency are retained, while redundant or weakly informative features are excluded. All numerical features are standardized to have zero mean and unit variance, while categorical variables are encoded as integers. For sequence-based modeling, the features are organized into *fixed-length sequences*, each representing a series of consecutive epochs to preserve temporal dependencies.

The final epoch in each sequence reflects a modified core type and V/f level for the AoI, aligning with the forecasting objective. This hierarchical and temporally structured feature representation enables the predictive models to capture the AoI's execution behavior, its interaction with co-runners, and the resulting impact on system energy efficiency. It also facilitates the learning of temporal dynamics, contention patterns, and runtime variability, thereby supporting accurate energy efficiency predictions under diverse and previously unseen operating conditions.

8.6.3. ML Models

Conventional ML approaches, such as decision trees and shallow neural networks lack the expressiveness required to model the complex relationships of our task. In contrast, sequence-based models—specifically LSTMs and Transformers—are well-suited to this task, owing to their ability to process temporal input and model dynamic dependencies. Transformers utilize attention mechanisms to learn long-range interactions, whereas LSTMs maintain temporal continuity through their recurrent structure.

The Transformer model adopts an encoder-based architecture to process sequences of system metrics and predict future energy efficiency values. The input sequence $\{\mathbf{X}_t, \mathbf{X}_{t+1}, \ldots, \mathbf{X}_{t+T}\}$, enriched with positional encodings, is fed through multiple *Multi-Head Attention* layers that extract temporal dependencies and incorporate information about hypothetical reconfigurations, such as core type and V/f level. The output of the attention layers is passed through a *Feed-Forward Network* (FFN), which applies non-linear transformations:

$$\mathrm{FFN}(\mathbf{u}) = \mathrm{ReLU}(\mathbf{u}\mathbf{W}_1 + \mathbf{b}_1)\mathbf{W}_2 + \mathbf{b}_2,$$

where $\mathbf{W}_1, \mathbf{W}_2, \mathbf{b}_1, \mathbf{b}_2$ are learnable parameters. The encoder's final output is passed to a regression head, which predicts the energy efficiency E_{t+T+1}. Hyperparameters such as the number of encoder layers, attention heads, and hidden dimensions are optimized using grid search.

The LSTM-based model processes sequences by maintaining a hidden state across epochs. Each element of the input sequence is passed through an LSTM cell, which updates the hidden state according to:

$$\mathbf{h}_t = \sigma(\mathbf{W}_h\mathbf{X}_t + \mathbf{U}_h\mathbf{h}_{t-1} + \mathbf{b}_h),$$

where $\mathbf{h}_t$ denotes the hidden state at epoch t, and $\mathbf{W}_h, \mathbf{U}_h, \mathbf{b}_h$ are learnable parameters. The final hidden state is passed through a regression head to produce the prediction E_{t+T+1}. LSTM model hyperparameters—including the number of layers, hidden units, and dropout rates—are optimized using random search. The model is trained using the Adam optimizer, coupled with a learning rate scheduler to ensure efficient convergence.

Both models are trained to minimize the mean squared error (MSE) loss function, with early stopping employed to prevent overfitting. As will be demonstrated in Section 8.7, these sequence-based architectures effectively capture both temporal and spatial patterns, and achieve significantly higher forecasting accuracy compared to traditional baselines.

8.7. Experimental Evaluation

All experiments and evaluations are conducted on a Linux-based platform with heterogeneous Intel® Core™ i9 12th generation Alder Lake processor, as described in experimental setup (C) (Section 3.1.1).

8.7.1. Model Architectures

The Transformer model utilizes an encoder composed of multi-head attention layers with 8 attention heads, each having a key dimension of 64, to effectively capture temporal dependencies across epochs. The encoder incorporates a feed-forward network comprising two dense layers: the first with 128 units and ReLU activation, followed by a dropout layer with a rate of 0.1; the second adjusts the dimensionality to match that of the residual connections. Dropout is applied after both the attention and feed-forward layers to mitigate overfitting, and layer normalization is used throughout to enhance training stability. The final encoder output corresponding to the last time step is passed to a regression head to predict energy efficiency for the next epoch. The model is optimized using the *Adam* optimizer with a learning rate of 1×10^{-3} and trained for up to 100 epochs, using early stopping with a patience of 5 epochs based on validation loss.

The LSTM model adopts a stacked architecture consisting of two recurrent layers. The first LSTM layer includes 64 units and returns a sequence of hidden states, which is then processed by a second LSTM layer with 32 units that outputs the final hidden state. Dropout with a rate of 0.2 is applied after each recurrent layer to improve generalization. The final hidden state is passed to a dense regression layer that outputs the predicted energy efficiency for the next epoch. This model also uses the *Adam* optimizer with a learning rate of 1×10^{-3} and follows the same early stopping protocol as the Transformer.

Both models are trained using MSE loss on a dataset comprising 1.81 million samples, partitioned into training (70%), validation (15%), and test (15%) sets.

Through empirical analysis, we determined a sequence length of 5 epochs to provide the best trade-off between model complexity and performance. The prediction target is the IPJ for the next epoch, expressed in millions of instructions per Joule.

8.7.2. Model Performance Comparison

As discussed in Section 8.4.2, existing ML-based resource management approaches typically aim to estimate the effect of hypothetical configurations on individual system metrics—such as power, performance, or energy consumption. However, these approaches do not explicitly predict energy efficiency, nor do they forecast future system states in real heterogeneous processor environments.

To benchmark our proposed models, we evaluate two commonly used baselines in resource management: Neural Networks (NN) and Boosted Decision Trees (XGB). To ensure a fair comparison, both baseline models are trained under two configurations.

In the first setting, NN and XGB are trained to predict energy efficiency for the *current* epoch using the same feature set as used in our sequence-based models. In the second setting, they are trained to forecast energy efficiency for the *next* epoch, simulating a minimal sequence length of two by using current epoch features to predict the subsequent one.

Figure 8.7 presents the residual distributions (i.e., the difference between actual and predicted values) for each model, along with metrics including MSE, RMSE, and the means of both target and residual distributions. In the *current* epoch prediction task, the NN and XGB baselines perform well, achieving RMSE values of 1.51 and 1.354, respectively. However, in the more demanding *future* epoch prediction task—especially under conditions with high temporal variability, as outlined in Section 8.4.2—their performance degrades significantly. This demonstrates their limited ability to capture temporal dependencies and adapt to dynamically evolving system states.

By contrast, the proposed sequence-based models,Transformer and LSTM, perform robustly in the *future* prediction setting. They achieve RMSE values of 1.87 and 1.14, respectively, with mean residuals close to zero, indicating strong predictive accuracy and minimal bias. These results underscore

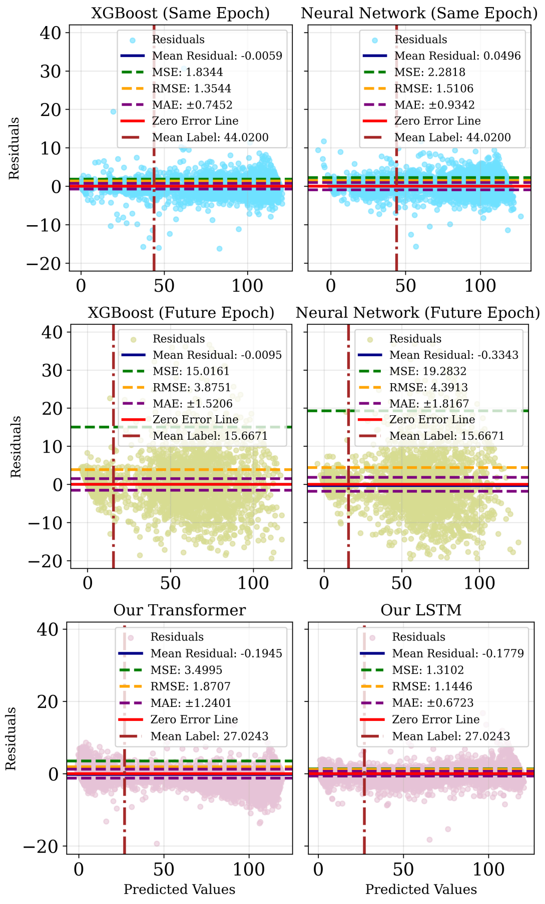


Figure 8.7.: Our Transformer and LSTM models exhibit excellet accuracy in forecasting future energy efficiency under hypothetical system configurations, achieving RMSE scores as low as 1.14. In contrast, simpler state-of-the-art models—though effective at predicting energy efficiency under the assumption of no change in the immediate next epoch—struggle to generalize for predictions spanning future epochs. This highlights the essential role of sequence data and advanced machine learning models in accurately forecasting energy efficiency in dynamic, evolving scenarios.

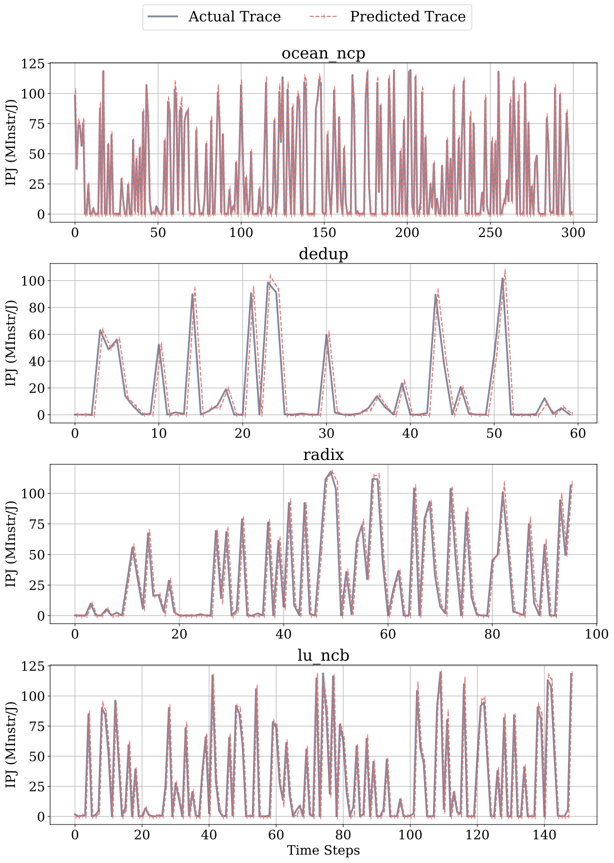


Figure 8.8.: Actual vs. **forecasted** energy efficiency traces for four applications executing concurrently with 10 background applications in an experiment entirely **unseen** during training. Every 5 epochs, the target application is migrated, and the V/f level of its assigned core is adjusted. Our Transformer model effectively captures phase-level sensitivities to these changes, as well as chip-wide contention effects, achieving high prediction accuracy at runtime and facilitating robust proactive resource management.

the effectiveness of our models in capturing both temporal dynamics and configuration sensitivities in heterogeneous systems.

8.7.3. Evaluation of Model Generalization

To assess the generalization capabilities of the Transformer model under previously unseen runtime conditions, we conducted evaluations on experiments that were explicitly excluded from the training dataset. The scenario illustrated in Fig. 8.8 involves 14 concurrently executing applications. Among these, we focus on the same four applications analyzed in Fig. 8.3—*radix*, *dedup*, *ocean_ncp*, and *lu_ncb*—each exhibiting distinct temporal sensitivity profiles. Figure 8.8 displays both the predicted and actual energy efficiency values, evaluated at intervals of five epochs. The results demonstrate that the Transformer model maintains high prediction accuracy even in contention-rich, previously unseen scenarios. Notably, the model accurately tracks fluctuations in *ocean_ncp*, despite its known propensity for periodic spikes in energy efficiency.

A particularly-important observation is that, even in rare cases where the model does not match the exact absolute value of the label, it consistently predicts the *correct trend*—that is, it never forecasts an increase when the actual value decreases, or vice versa. This behavior was observed across all applications in the evaluation set. Such consistency highlights the robustness of our model architecture in capturing complex temporal and spatial dependencies inherent to heterogeneous multi-core systems. This analysis demonstrates that models exhibiting this level of generalization are well-positioned to support intelligent, proactive resource management strategies that optimize energy efficiency in real time.

8.7.4. Training and Inference Overhead

Although the Transformer and LSTM models incur longer training times relative to conventional approaches such as feedforward neural networks and XGB, this overhead is considered acceptable, as training is performed offline during the design phase. Once trained, the models are deployed on the target heterogeneous processor and used solely for runtime inference.

Inference is executed using Intel's oneDNN (Deep Neural Network Library), which provides hardware-accelerated routines optimized for neural network computations. On the target platform, the average inference time for evaluating a single input sequence is 94 ms. To improve efficiency, we employ batch inference, enabling parallel processing of multiple sequences. This optimization reduces the average per-sequence inference time to just 1.82 ms. Given that the epoch length is 100 ms, this represents less than 2% of the runtime interval, indicating that the model imposes negligible overhead. As such, the proposed forecasting framework is well-suited for integration into real-time energy efficiency optimization workflows on clustered heterogeneous processors.

8.8. Summary

This chapter introduced EffiCast, the first methodology in the literature for forecasting energy efficiency in clustered heterogeneous multi-core processors using Transformers and LSTMs.

Tackled Challenges Three challenges from Section 1.3 were of particular relevance in this work. EffiCast was deployed on the Intel Alder Lake machine (Section 3.1.1), which features a very complex memory hierarchy that creates both intra- and inter-cluster contention between concurrently running applications, as discussed in Challenge 2 (Section 1.3.2). This challenge was successfully addressed thanks to our robust machine learning problem formulation, the training methodology, and the feature selection. Furthermore, the 1.82 ms inference time and the very high forecasting accuracy of our models validate the integration of energy efficiency forecasting into proactive resource management frameworks, enabling smarter RM technique for energy-constrained systems, relating to Challenges 4 and 5 (Sections 1.3.4 and 1.3.5).

Achievements and Takeaways Unlike simpler state-of-the-art techniques, our models achieved exceptional accuracy in capturing complex temporal and resource contention dynamics. Our models were deployed on **real hardware** with Intel's oneDNN acceleration and demonstrated minimal runtime overhead, with batch inference reducing per-sequence latency to a very negligible 1.82 ms.

Insights Unlike the three previous contributions (*TcRM*, *SmartCM*, and *MTCM*), EffiCast is **not** a RM technique. The proposed contribution lies in the Transformer- and LSTM-based forecasting models and their associated training methodology, with the analysis section (Section 8.4) demonstrating substantial potential gains in energy efficiency if the latter can be accurately forecasted. Still, this work did not experimentally demonstrate such gains through a RM technique. A logical continuation of this work would therefore be to build a RM technique that leverages the capabilities of the proposed models to maximize energy efficiency.

9. Conclusion

This dissertation has presented a comprehensive body of work that addressed the challenges of shared cache contention in modern homogeneous and heterogeneous multi-core processors.

9.1. Consolidated Contributions

***TcRM*: Mapping and DVFS with Cache Contention Awareness** The first contribution, *TcRM* in Chapter 5, investigated for the first time in the literature the trade-off between cache contention and thermal effects in clustered multicore processors. By training a neural network model at design time to predict application slowdowns caused by cache contention, *TcRM* enabled effective application-to-cluster and thread-to-core mapping, combined with per-cluster voltage and V/f level selection. This resulted in a significant reduction in chip temperature while satisfying application performance constraints.

***SmartCM*: Dynamic Migration under Thermal Constraints** Building upon *TcRM*, *SmartCM* introduced dynamic application migration as a control knob in Chapter 6, allowing continuous adaptation to changing workload phases and contention conditions. This migration capability enabled significant performance gains while maintaining thermal safety. Our lightweight neural network model allowed runtime decisions with minimal overhead, thus addressing the stringent timing requirements of thermal- and contention-aware dynamic resource management.

***MTCM*: Exploring HBM Integration** The third contribution, *MTCM* in Chapter 7, extended resource management to 3D-stacked memory systems, specifically high-bandwidth memory, in clustered multicore architectures. It introduced the first methodology in the literature that jointly manages cache contention and thermal safety across both the processor and HBM subsystems. A suite of four machine learning models was used to coordinate application-to-cluster migration and frequency scaling, achieving substantial performance improvements over state-of-the-art solutions while ensuring thermal safety with microsecond-level temperature predictions.

PHASEL: Enabling Phase-Level Learning of Contention Dynamics The challenges of capturing complex contention patterns using ML-based models were successfully addressed in the three previous works, thanks to the robustness of our proposed training methodology, PHASEL in Chapter 4. It addressed the challenge of generating training data for training lightweight ML models, suitable for system-level RM at runtime, by extracting the phase-level behavior of applications through non-intrusive phase-level slicing of their execution traces.

EffiCast: Energy Efficiency Forecasting with Transformers and LSTMs on Real Hardware Unlike the previous three contributions, EffiCast is not a runtime management technique but rather a forecasting framework. It introduced in Chapter 8 the first use of Transformer and LSTM models for predicting energy efficiency in heterogeneous multicore systems. Deployed on real hardware with Intel's oneDNN acceleration and evaluated using **our proposed ARDiS framework** [6] (Section 3.3), the proposed models demonstrated excellent accuracy and minimal overhead (1.82 ms per sequence), validating their practicality. EffiCast enables future integration into proactive management systems that optimize task placement and DVFS based on predicted energy behavior, representing a significant step toward smarter and more energy-efficient computing.

9.2. Future Work

While the contributions presented in this dissertation have demonstrated significant progress in mitigating cache contention-related challenges in modern multicore and heterogeneous systems, several promising research directions naturally emerge as extensions of this work.

Explainable and Interpretable ML-Based Resource Management with XAI All the contributions presented in this dissertation, as well as the majority of recent state-of-the-art techniques in resource management, have relied on some form of AI or machine learning model(s). While these models have proven to be key enablers in managing cache contention, thermal behavior, and energy efficiency, they generally operate as *opaque* black-box systems. Despite their predictive accuracy, they offer little to no transparency about the internal mechanisms that influence their decisions, e.g., why a particular application is mapped to a specific core cluster, how frequency levels are selected in response to thermal constraints, etc. This lack of interpretability makes it hard to debug, validate, or to improve these models, particularly when they fail under unseen or corner-case scenarios. Moreover, in cases where runtime decisions affect system safety, e.g., hard real-time systems, performance isolation, or energy-critical operations, understanding the *rationale* behind model outputs becomes crucial.

A promising direction to tackle this limitation is the integration of explainable AI (XAI) methods into ML-based resource management. Tools such as SHAP (SHapley Additive exPlanations), integrated gradients, or model-specific introspection techniques, e.g., attention map visualization in Transformers, could be use to clarify which input features or indicators are most influential in the prediction of a label. This would help identify erroneous model behavior, non-obvious and counter-intuitive correlations, or even refine feature engineering processes. Embedding XAI into these frameworks would thus increase *trustworthiness* and *robustness*, and could also be the basis for certifiable ML-driven management strategies in future operating systems.

The Future of Integration and Packaging A particularly compelling direction lies in adapting our proposed techniques to emerging integration paradigms such as chiplet-based, 2.5D, and 3D-stacked architectures. These advanced packaging technologies enable heterogeneous integration of processing cores, 3D memories like HBM, and accelerators either on a shared silicon interposer (2.5D) or through vertical stacking (3D). Such architectures alter the thermal landscape of the system. The proximity between compute and memory subsystems introduces significant thermal coupling, whereby heat generated in one component directly influences the thermal profile of the other. This introduces new challenges in thermal modeling and management, as assumptions of *thermal independence* between subsystems, valid our considered architectures, no longer hold.

To ensure thermal safety and maintain performance efficiency in these environments, resource management techniques must evolve accordingly. Our existing thermal-aware techniques would need to incorporate models that reflect these thermal interactions. This implies the development of joint thermal predictors that account for heat propagation across the stack or interposer, possibly using multi-input learning architectures that take the state of both compute and memory elements as input features.

Training these models would also present additional complexity. Thermal behavior in 2.5D/3D systems is highly sensitive to layout, floorplanning, and even workload-specific activity patterns. Gathering these architectural parameters can be challenging, as manufacturers rarely reveal all the details required to model their chips, e.g., energy per access, distance between dies, etc. Standardization and collaborations between manufacturers, design automation and operating system community would be necessary to enable further advancements in this direction.

Advances in Generative AI for Resource Management Recent advances in generative AI offer exciting possibilities for reimagining resource management. Rather than relying on strategies that evaluate individual RM decisions, generative models could be employed to directly synthesize optimized application-to-cluster mappings via a single inference step or prompt. This shift could reduce decision-making latency significantly, particularly in multicore systems with complex and dynamic contention landscapes. We have initiated the exploration of fine tuning large language models (LLMs) in our recent work [7] for a Verilog code classification task. However, further research is required to transition these techniques into the system-level resource management realm. Moreover, this direction could be extended to consider *Agentic AI* frameworks, where a collection of specialized agents collaboratively manage different optimization aspects, e.g., performance, temperature, and energy efficiency, within the operating system. Realizing these capabilities in practice would also necessitate the availability of efficient inference acceleration hardware, similar to the Intel OneDNN backend that enabled fast, Transformer-based inference in our EffiCast deployment.

9.3. Closing Takeaways

This dissertation has demonstrated that machine learning-based approaches, when carefully integrated with system-level RM, can successfully tackle the cache contention problem, and lead to significant improvement in terms of performance, temperature, and energy efficiency in modern homogeneous and heterogeneous multicores. By combining predictive modeling and proactive RM, we have shown the relevance, feasibility, *and* impact of cache contention-aware resource management, thereby paving the way for future developments in explainable, scalable, and generative AI-driven system-level optimization.

Bibliography

[1] Mohammed Bakr Sikal, Heba Khdr, Martin Rapp, and Jörg Henkel. “Thermal- and Cache-Aware Resource Management based on ML-Driven Cache Contention Prediction”. In: *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 2022, pp. 1384–1388. DOI: 10.23919/DATE54114.2022.9774776.

[2] Mohammed Bakr Sikal, Heba Khdr, Martin Rapp, and Jörg Henkel. “Machine Learning-based Thermally-Safe Cache Contention Mitigation in Clustered Manycores”. In: *2023 60th ACM/IEEE Design Automation Conference (DAC)*. 2023, pp. 1–6. DOI: 10.1109/DAC56929.2023.10247708.

[3] Mohammed Bakr Sikal, Heba Khdr, Lokesh Siddhu, and Jörg Henkel. “ML-Based Thermal and Cache Contention Alleviation on Clustered Manycores With 3-D HBM”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)* 43.11 (2024), pp. 3614–3625. DOI: 10.1109/TCAD.2024.3438998.

[4] Mohammed Bakr Sikal, Jeferson González-Gómez, Heba Khdr, and Jörg Henkel. “Contention-Aware Forecasting of Energy Efficiency through Sequence-Based Models in Modern Heterogeneous Processors”. In: *2025 62nd ACM/IEEE Design Automation Conference (DAC)*. 2025.

[5] Mohammed Bakr Sikal, Jeferson González-Gómez, Osama Abboud, Xun Xiao, Heba Khdr, and Jörg Henkel. “PHASEL: Learning Phase-Level Application Sensitivities for Energy-Efficient Resource Management in Heterogeneous 6G Systems”. In: Under Review. 2025.

[6] Mohammed Bakr Sikal, Jeferson González-Gómez, Heba Khdr, and Jörg Henkel. “ARDiS: A Portable and Unified Resource Management Framework in Real Hardware Systems”. In: *ACM Transactions on Design Automation of Electronic Systems (TODAES)* (2025). Under Review.

[7] Mohammed Bakr Sikal, Hassan Nassar, Heba Khdr, and Jörg Henkel. "A Dataset for LLM-Based Detection of Power-Wasters in Routed FPGA Netlists". In: *2025 IEEE International Conference on LLM-Aided Design (ICLAD)*. 2025, pp. 235–241. DOI: `10.1109/ICLAD65226.2025.00023`.

[8] Martin Rapp, Mohammed Bakr Sikal, Heba Khdr, and Jörg Henkel. "SmartBoost: Lightweight ML-Driven Boosting for Thermally-Constrained Many-Core Processors". In: *2021 58th ACM/IEEE Design Automation Conference (DAC)*. 2021, pp. 265–270. DOI: `10.1109/DAC18074.2021.9586287`.

[9] Jeferson González-Gómez, Mohammed Bakr Sikal, Heba Khdr, Lars Bauer, and Jörg Henkel. "Balancing Security and Efficiency: System-Informed Mitigation of Power-Based Covert Channels". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 43.11 (2024), pp. 3395–3406. DOI: `10.1109/TCAD.2024.3438999`.

[10] Jeferson González-Gómez, Mohammed Bakr Sikal, Heba Khdr, Lars Bauer, and Jörg Henkel. "Smart Detection of Obfuscated Thermal Covert Channel Attacks in Many-core Processors". In: *2023 60th ACM/IEEE Design Automation Conference (DAC)*. 2023, pp. 1–6. DOI: `10.1109/DAC56929.2023.10247844`.

[11] Heba Khdr, Mustafa Enes Batur, Kanran Zhou, Mohammed Bakr Sikal, and Jörg Henkel. "Multi-Agent Reinforcement Learning for Thermally-Restricted Performance Optimization on Manycores". In: *2024 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 2024, pp. 1–6. DOI: `10.23919/DATE58400.2024.10546574`.

[12] Konstantinos Balaskas, Heba Khdr, Mohammed Bakr Sikal, Fabian Kreß, Kostas Siozios, Jürgen Becker, and Jörg Henkel. "Heterogeneous Accelerator Design for Multi-DNN Workloads via Heuristic Optimization". In: *IEEE Embedded Systems Letters* 16.4 (2024), pp. 317–320. DOI: `10.1109/LES.2024.3443628`.

[13] Heba Khdr, Mohammed Bakr Sikal, Benedikt Dietrich, and Jörg Henkel. "Towards the Optimization of Hardware Efficiency through Machine Learning". In: *2025 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. 2025.

[14] Jörg Henkel, Lars Bauer, Nikil Dutt, Puneet Gupta, Sani Nassif, Muhammad Shafique, Mehdi Tahoori, and Norbert Wehn. "Reliable on-chip systems in the nano-era: lessons learnt and future trends". In: *Proceedings of the 50th Annual Design Automation Conference*. DAC '13. Austin, Texas: Association for Computing Machinery, 2013. ISBN: 9781450320719. DOI: `10.1145/2463209.2488857`. URL: `https://doi.org/10.1145/2463209.2488857`.

[15] Thomas Ebi, David Kramer, Wolfgang Karl, and Jörg Henkel. "Economic Learning for Thermal-Aware Power Budgeting in Many-Core Architectures". In: *Conf. Hardw./Softw. Codesign and System Synthesis (CODES+ISSS)*. 2011.

[16] Heba Khdr, Santiago Pagani, Muhammad Shafique, and Jörg Henkel. "Thermal constrained resource management for mixed ILP-TLP workloads in dark silicon chips". In: *Design Automation Conference (DAC)*. 2015, pp. 1–6. DOI: `10.1145/2744769.2744916`.

[17] Heba Khdr, Hussam Amrouch, and Jörg Henkel. "Aging-Constrained Performance Optimization for Multi Cores". In: *Design Automation Conference (DAC)*. 2018, pp. 1–6. DOI: `10.1109/DAC.2018.8465829`.

[18] Heba Khdr, Hussam Amrouch, and Jörg Henkel. "Dynamic Guardband Selection: Thermal-Aware Optimization for Unreliable Multi-Core Systems". In: *IEEE Transactions on Computers* 68.1 (2019), pp. 53–66. DOI: `10.1109/TC.2018.2848276`.

[19] Heba Khdr, Hussam Amrouch, and Jörg Henkel. "Aging-Aware Boosting". In: *IEEE Transactions on Computers* 67.9 (2018), pp. 1217–1230. DOI: `10.1109/TC.2018.2816014`.

[20] Heba Khdr, Santiago Pagani, Éricles Sousa, Vahid Lari, Anuj Pathania, Frank Hannig, Muhammad Shafique, Jürgen Teich, and Jörg Henkel. "Power Density-Aware Resource Management for Heterogeneous Tiled Multicores". In: *IEEE Transactions on Computers* 66.3 (2017), pp. 488–501. DOI: `10.1109/TC.2016.2595560`.

[21] Jörg Henkel, Heba Khdr, and Martin Rapp. "Smart Thermal Management for Heterogeneous Multicores". In: *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 2019, pp. 132–137. DOI: `10.23919/DATE.2019.8715001`.

[22] Martin Rapp, Anuj Pathania, Tulika Mitra, and Jörg Henkel. “Neural Network-Based Performance Prediction for Task Migration on S-NUCA Many-Cores”. In: *IEEE Transactions on Computers* 70.10 (2021), pp. 1691–1704. DOI: 10.1109/TC.2020.3023022.

[23] Jeferson Gonzalez-Gomez, Lars Bauer, and Jörg Henkel. “Cache-based side-channel attack mitigation for many-core distributed systems via dynamic task migration”. In: *IEEE Transactions on Information Forensics and Security* 18 (2023), pp. 2440–2450.

[24] Jorg Henkel, Lokesh Siddhu, Lars Bauer, Jurgen Teich, Stefan Wildermann, Mehdi Tahoori, Mahta Mayahinia, Jeronimo Castrillon, Asif Ali Khan, Hamid Farzaneh, Joao Paulo C. De Lima, Jian-Jia Chen, Christian Hakert, Kuan-Hsun Chen, Chia-Lin Yang, and Hsiang-Yun Cheng. “Special Session - Non-Volatile Memories: Challenges and Opportunities for Embedded System Architectures with Focus on Machine Learning Applications”. In: *Proceedings of the International Conference on Compilers, Architecture, and Synthesis for Embedded Systems*. CASES ’23 Companion. Hamburg, Germany: Association for Computing Machinery, 2024, pp. 11–20. ISBN: 9798400702907. DOI: 10.1145/3607889.3609088. URL: https://doi.org/10.1145/3607889.3609088.

[25] Muhammad Shafique, Waqas Ahmad, Rehan Hafiz, and Jörg Henkel. “A low latency generic accuracy configurable adder”. In: *Proceedings of the 52nd Annual Design Automation Conference*. DAC ’15. San Francisco, California: Association for Computing Machinery, 2015. ISBN: 9781450335201. DOI: 10.1145/2744769.2744778. URL: https://doi.org/10.1145/2744769.2744778.

[26] R. Ernst, J. Henkel, and T. Benner. “Hardware-software cosynthesis for microcontrollers”. In: *IEEE Design & Test of Computers* 10.4 (1993), pp. 64–75. DOI: 10.1109/54.245964.

[27] Advanced Micro Devices, Inc. *EPYC: The Next Generation of AMD Server Processors*. https://www.amd.com/en/products/processors/server/epyc.html. Accessed: 2025-04-13. 2017.

[28] ARM Ltd. *big.LITTLE Technology: The Future of Mobile*. https://armkeil.blob.core.windows.net/developer/Files/pdf/white-paper/big-little-technology-the-future-of-mobile.pdf. Accessed: 2025-04-20. 2013.

[29] *Hybrid architecture (code name Alder Lake)*. URL: https://www.intel.com/content/www/us/en/developer/articles/technical/hybrid-architecture.htm.

[30] Zixuan Zhang. "Analysis of the Advantages of the M1 CPU and Its Impact on the Future Development of Apple". In: *2021 2nd International Conference on Big Data & Artificial Intelligence & Software Engineering (ICBASE)*. 2021, pp. 732–735. DOI: 10.1109/ICBASE53849.2021.00143.

[31] John Shalf. "The future of computing beyond Moore's Law". In: *Philosophical Transactions of the Royal Society A* 378.2166 (2020), p. 20190061.

[32] R.R. Schaller. "Moore's law: past, present and future". In: *IEEE Spectrum* 34.6 (1997), pp. 52–59. DOI: 10.1109/6.591665.

[33] Mark Bohr. "A 30 Year Retrospective on Dennard's MOSFET Scaling Paper". In: *IEEE Solid-State Circuits Society Newsletter* 12.1 (2007), pp. 11–13. DOI: 10.1109/N-SSC.2007.4785534.

[34] Zhiwen Chen, Jiaju Zhang, Shizhao Wang, and Ching-Ping Wong. "Challenges and prospects for advanced packaging". In: *Fundamental Research* 4.6 (2024), pp. 1455–1458. ISSN: 2667-3258. DOI: https://doi.org/10.1016/j.fmre.2023.04.014. URL: https://www.sciencedirect.com/science/article/pii/S2667325823001334.

[35] Shekhar Borkar and Andrew A. Chien. "The Future of Microprocessors". In: *Communications of the ACM* 54.5 (2011), pp. 67–77.

[36] *AMD "Zen 3" Core Architecture*. 2020. URL: https://www.amd.com/en/technologies/zen-core-3.

[37] Counterpoint Research. *Mobile SoC Market Share: Q3 2024 Report*. https://www.counterpointresearch.com/insights/global-smartphone-soc-market-share/. Accessed: 2025-04-13. 2023.

[38] Samuel Naffziger, Noah Beck, Thomas Burd, Kevin Lepak, Gabriel H. Loh, Mahesh Subramony, and Sean White. "Pioneering Chiplet Technology and Design for the AMD EPYC™ and Ryzen™ Processor Families : Industrial Product". In: *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. 2021, pp. 57–70. DOI: 10.1109/ISCA52012.2021.00014.

[39] Advanced Micro Devices, Inc. *AMD EPYC 9004 Series Product Brief*. https://www.amd.com/en/processors/epyc-9004-series. Accessed: 2025-04-13. 2022.

[40] Tom's Hardware from Mercury Research. *AMD records its highest server market share in decades — Intel fights back in client PCs.* `https://www.tomshardware.com/pc-components/cpus/amd-records-its-highest-server-market-share-in-decades-but-intel-fights-back-in-client-pcs`. Accessed: 2025-05-02. 2024.

[41] Intel Corporation. *Intel Xeon Scalable Processors (Sapphire Rapids).* `https://www.intel.com/content/www/us/en/products/details/processors/xeon/scalable.html`. Accessed: 2025-04-13. 2023.

[42] Zhenyu Zhang, Henry Hoffmann, and David Hsu. "Performance Interference and Resource Management in Multi-core Systems: A Survey". In: *ACM Computing Surveys (CSUR)* 48.1 (2014), pp. 1–39.

[43] John L Hennessy and David A Patterson. *Computer architecture: a quantitative approach.* Elsevier, 2011.

[44] Parul Sohal, Michael Bechtel, Renato Mancuso, Heechul Yun, and Orran Krieger. "A closer look at intel resource director technology (rdt)". In: *Proceedings of the 30th International Conference on Real-Time Networks and Systems.* 2022, pp. 127–139.

[45] Swadhesh Kumar and PK Singh. "An overview of modern cache memory and performance analysis of replacement policies". In: *2016 IEEE International Conference on Engineering and Technology (ICETECH).* IEEE. 2016, pp. 210–214.

[46] Sparsh Mittal. "A Survey of Techniques for Cache Partitioning in Multicore Processors". In: *ACM Comput. Surv.* 50.2 (May 2017). ISSN: 0360-0300. DOI: `10.1145/3062394`. URL: `https://doi.org/10.1145/3062394`.

[47] Song Liu, Jie Ma, Zengyuan Zhang, Xinhe Wan, Bo Zhao, and Weiguo Wu. "Scalpel: High Performance Contention-Aware Task Co-Scheduling for Shared Cache Hierarchy". In: *IEEE Transactions on Computers* 74.2 (2025), pp. 678–690. DOI: `10.1109/TC.2024.3500381`.

[48] Anuj Pathania and Jörg Henkel. "HotSniper: Sniper-Based Toolchain for Many-Core Thermal Simulations in Open Systems". In: *IEEE Embedded Systems Letters (ESL)* (2018). DOI: `10.1109/LES.2018.2866594`.

[49] Christian Bienia, Sanjeev Kumar, Jaswinder Pal Singh, and Kai Li. "The PARSEC Benchmark Suite: Characterization and Architectural Implications". In: *Parallel Architectures and Compilation Techniques (PACT).* ACM. 2008. DOI: `10.1145/1454115.1454128`.

[50] Steven Cameron Woo, Moriyoshi Ohara, Evan Torrie, Jaswinder Pal Singh, and Anoop Gupta. “The SPLASH-2 Programs: Characterization and Methodological Considerations”. In: *Int. Symp. Computer Architecture (ISCA)* (1995).

[51] Silas Boyd-Wickizer, Haibo Chen, Rong Chen, Yandong Mao, M Frans Kaashoek, Robert Morris, et al. “Corey: An Operating System for Many Cores”. In: *Symp. Operating System Design and Implementation (OSDI)*. 2008.

[52] *Intel 64 and IA-32 Architectures Software Developer’s Manual*. Intel Corporation. 2016.

[53] Dror G Feitelson and Larry Rudolph. “Metrics and Benchmarking for Parallel Job Scheduling”. In: *Workshop on Job Scheduling Strategies for Parallel Processing*. Springer. 1998.

[54] Shivam Kundan and Iraklis Anagnostopoulos. “A Machine Learning Approach for Improving Power Efficiency on Clustered Multi-Processor System”. In: *International Symposium on Circuits and Systems (ISCAS)*. 2020. DOI: 10.1109/ISCAS45731.2020.9180474.

[55] Sergey Zhuravlev, Sergey Blagodurov, and Alexandra Fedorova. “Addressing shared resource contention in multicore processors via scheduling”. In: *Proceedings of the Fifteenth International Conference on Architectural Support for Programming Languages and Operating Systems*. ASPLOS XV. Pittsburgh, Pennsylvania, USA: Association for Computing Machinery, 2010, pp. 129–142. ISBN: 9781605588391. DOI: 10.1145/1736020.1736036. URL: https://doi.org/10.1145/1736020.1736036.

[56] Yuejian Xie and Gabriel Loh. “Dynamic classification of program memory behaviors in CMPs”. In: *The 2nd workshop on Chip multiprocessor memory systems and interconnects*. Citeseer. 2008.

[57] Lokesh Siddhu, Rajesh Kedia, and Preeti Ranjan Panda. “Leakage-aware dynamic thermal management of 3D memories”. In: *ACM TODAES* 26.2 (2020), pp. 1–31.

[58] Venkatesh Pallipadi and Alexey Starikovskiy. “The Ondemand Governor”. In: *The Linux Symposium* (2006).

[59] Jason Mars, Lingjia Tang, Robert Hundt, Kevin Skadron, and Mary Lou Soffa. “Bubble-Up: increasing utilization in modern warehouse scale computers via sensible co-locations”. In: *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture.* MICRO-44. Porto Alegre, Brazil: Association for Computing Machinery, 2011, pp. 248–259. ISBN: 9781450310536. DOI: 10.1145/2155620.2155650. URL: https://doi.org/10.1145/2155620.2155650.

[60] Dimitris Kaseridis, Muhammad Faisal Iqbal, and Lizy Kurian John. “Cache Friendliness-Aware Managementof Shared Last-Level Caches for HighPerformance Multi-Core Systems”. In: *IEEE Transactions on Computers* 63.4 (2014), pp. 874–887. DOI: 10.1109/TC.2013.18.

[61] Yeseong Kim, Ankit More, Emily Shriver, and Tajana Rosing. “Application performance prediction and optimization under cache allocation technology”. In: *Design, Automation & Test in Europe (DATE).* IEEE. 2019.

[62] Lavanya Subramanian, Vivek Seshadri, Arnab Ghosh, Samira Khan, and Onur Mutlu. “The Application Slowdown Model: Quantifying and Controlling the Impact of Inter-Application Interference at Shared Caches and Main Memory”. In: *International Symposium on Microarchitecture (MICRO).* 2015. DOI: 10.1145/2830772.2830803.

[63] Xi E Chen and Tor Aamodt. “Modeling Cache Contention and Throughput of Multiprogrammed Manycore Processors”. In: *IEEE Trans. on Computers (TC)* 61.7 (2011), pp. 913–927.

[64] Matheus A. Souza and Henrique C. Freitas. “Reinforcement Learning-Based Cache Replacement Policies for Multicore Processors”. In: *IEEE Access* 12 (2024), pp. 79177–79188. DOI: 10.1109/ACCESS.2024.3409228.

[65] Jian-He Liao, He-Ru Chen, and Ya-Shu Chen. “A Cache Contention-aware Run-time Scheduling for Power-constrained Asymmetric Multicore Processors”. In: *Research in Adaptive and Convergent Systems (RACS).* 2020.

[66] Parthasarathy Ranganathan, Sarita Adve, and Norman P. Jouppi. “Reconfigurable caches and their application to media processing”. In: *Proceedings of the 27th Annual International Symposium on Computer Architecture.* ISCA ’00. Vancouver, British Columbia, Canada: Association for Computing Machinery, 2000, pp. 214–224. ISBN: 1581132328.

DOI: 10.1145/339647.339685. URL: https://doi.org/10.1145/339647.339685.

[67] Jiang Lin, Qingda Lu, Xiaoning Ding, Zhao Zhang, Xiaodong Zhang, and P. Sadayappan. “Gaining insights into multicore cache partitioning: Bridging the gap between simulation and real systems”. In: *2008 IEEE 14th International Symposium on High Performance Computer Architecture*. 2008, pp. 367–378. DOI: 10.1109/HPCA.2008.4658653.

[68] Noriaki Suzuki, Hyoseung Kim, Dionisio De Niz, Bjorn Andersson, Lutz Wrage, Mark Klein, and Ragunathan Rajkumar. “Coordinated bank and cache coloring for temporal protection of memory accesses”. In: *2013 IEEE 16th International Conference on Computational Science and Engineering*. IEEE. 2013, pp. 685–692.

[69] Sri Harsha Gade and Sujay Deb. “A Novel Hybrid Cache Coherence with Global Snooping for Many-core Architectures”. In: 27.1 (Sept. 2021). ISSN: 1084-4309. DOI: 10.1145/3462775. URL: https://doi.org/10.1145/3462775.

[70] Kousik Kumar Dutta, Prathamesh Nitin Tanksale, and Shirshendu Das. “A Fairness Conscious Cache Replacement Policy for Last Level Cache”. In: *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 2021, pp. 695–700. DOI: 10.23919/DATE51398.2021.9474096.

[71] Aamer Jaleel, Kevin B. Theobald, Simon C. Steely, and Joel Emer. “High performance cache replacement using re-reference interval prediction (RRIP)”. In: *Proceedings of the 37th Annual International Symposium on Computer Architecture*. ISCA ’10. Saint-Malo, France: Association for Computing Machinery, 2010, pp. 60–71. ISBN: 9781450300537. DOI: 10.1145/1815961.1815971. URL: https://doi.org/10.1145/1815961.1815971.

[72] Masayuki Sato, Yongcheng Chen, Haruya Kikuchi, Kazuhiko Komatsu, and Hiroaki Kobayashi. “Perceptron-based Cache Bypassing for Way-Adaptable Caches”. In: *2019 IEEE Symposium in Low-Power and High-Speed Chips (COOL CHIPS)*. 2019, pp. 1–3. DOI: 10.1109/CoolChips.2019.8721331.

[73] Intel. *Intel® Resource Director Technology (Intel® RDT)*. https://www.intel.com/content/www/us/en/architecture-and-technology/resource-director-technology.html. Accessed: 2025-04-20. 2021.

[74] AMD. *AMD64 Technology Platform Quality of Service Extensions*. https://kib.kiev.ua/x86docs/AMD/MISC/56375_1.03_PUB.pdf. Accessed: 2025-04-20. 2022.

[75] ARM. *Arm Memory System Resource Partitioning and Monitoring (MPAM) System Component Specification*. https://developer.arm.com/documentation/ihi0099/latest/. Accessed: 2025-04-20. 2024.

[76] Intel. *Cache Allocation Technology in Intel® Xeon® Processor*. https://www.intel.com/content/www/us/en/developer/articles/technical/introduction-to-cache-allocation-technology.html. Accessed: 2025-04-20. 2021.

[77] *perf: Linux profiling with performance counters*. June 2009. URL: https://perf.wiki.kernel.org/index.php/Main_Page.

[78] Lavanya Subramanian, Donghyuk Lee, Vivek Seshadri, Harsha Rastogi, and Onur Mutlu. "BLISS: Balancing performance, fairness and complexity in memory access scheduling". In: *IEEE Trans. on Parallel and Distributed Systems* 27.10 (2016), pp. 3071–3087.

[79] Martin Rapp, Hussam Amrouch, Yibo Lin, Bei Yu, David Z. Pan, Marilyn Wolf, and Jörg Henkel. "MLCAD: A Survey of Research in Machine Learning for CAD Keynote Paper". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 41.10 (2022), pp. 3162–3181. DOI: 10.1109/TCAD.2021.3124762.

[80] Young Geun Kim, Minyong Kim, Joonho Kong, and Sung Woo Chung. "An Adaptive Thermal Management Framework for Heterogeneous Multi-Core Processors". In: *IEEE Trans. on Computers (TC)* 69.6 (2020), pp. 894–906.

[81] Daniel Hackenberg, Robert Schöne, Thomas Ilsche, Daniel Molka, Joseph Schuchart, and Robin Geyer. "An Energy Efficiency Feature Survey of the Intel Haswell Processor". In: *Int. Parallel and Distributed Processing Symp. Workshop (IPDPSW)*. IEEE. 2015, pp. 896–904.

[82] Somdip Dey, Enrique Zaragoza Guajardo, Karunakar Reddy Basireddy, Xiaohang Wang, Amit Kumar Singh, and Klaus McDonald-Maier. "EdgeCoolingMode: An Agent Based Thermal Management Mechanism for DVFS Enabled Heterogeneous MPSoCs". In: *International Conference on VLSI Design and International Conference on Embedded Systems (VLSID)*. IEEE. 2019, pp. 19–24.

[83] Xiaohang Wang, Amit Kumar Singh, Bing Li, Yang Yang, Hong Li, and Terrence Mak. “Bubble Budgeting: Throughput Optimization for Dynamic Workloads by Exploiting Dark Cores in Many Core Systems”. In: *IEEE Trans. on Computers (TC)* 67.2 (2017).

[84] Behnaz Pourmohseni, Stefan Wildermann, Fedor Smirnov, Paul E. Meyer, and Jürgen Teich. “Task Migration Policy for Thermal-Aware Dynamic Performance Optimization in Many-Core Systems”. In: *IEEE Access* 10 (2022), pp. 33787–33802.

[85] Somdip Dey, Amit Kumar Singh, Xiaohang Wang, and Klaus Dieter McDonald-Maier. “DeadPool: Performance Deadline Based Frequency Pooling and Thermal Management Agent in DVFS Enabled MPSoCs”. In: *2019 6th IEEE International Conference on Cyber Security and Cloud Computing (CSCloud)/ 2019 5th IEEE International Conference on Edge Computing and Scalable Cloud (EdgeCom)*. 2019, pp. 190–195.

[86] Di Liu, Shi-Gui Yang, Zhenli He, Mingxiong Zhao, and Weichen Liu. “CARTAD: Compiler-Assisted Reinforcement Learning for Thermal-Aware Task Scheduling and DVFS on Multicores”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 41.6 (2022), pp. 1813–1826.

[87] Martin Rapp, Heba Khdr, Nikita Krohmer, and Jörg Henkel. “NPU-Accelerated Imitation Learning for Thermal Optimization of QoS-Constrained Heterogeneous Multi-Cores”. In: *ACM Trans. Design Autom. Electr. Syst.* 29.1 (2024), 16:1–16:23.

[88] Hiroyuki Usui, Lavanya Subramanian, Kevin Kai-Wei Chang, and Onur Mutlu. “DASH: Deadline-Aware High-Performance Memory Scheduler for Heterogeneous Systems with Hardware Accelerators”. In: *ACM Trans. on Architecture and Code Optimization (TACO)* 12.4 (2016).

[89] Theodoros Marinakis, Shivam Kundan, and Iraklis Anagnostopoulos. “Meeting Power Constraints While Mitigating Contention on Clustered Multiprocessor System”. In: *IEEE Embedded Systems Letters (ESL)* 12.3 (2019).

[90] Nikita Mishra, John D. Lafferty, and Henry Hoffmann. “ESP: A Machine Learning Approach to Predicting Application Interference”. In: *International Conference on Autonomic Computing (ICAC)*. 2017, pp. 125–134.

[91] Anil Kanduri, Mohammad-Hashem Haghbayan, Amir M Rahmani, Muhammad Shafique, Axel Jantsch, and Pasi Liljeberg. “adBoost: Thermal Aware Performance Boosting Through Dark Silicon Patterning”. In: *IEEE Trans. Computers (TC)* (2018).

[92] Xiaohang Wang, Amit Kumar Singh, and Shengyan Wen. “Exploiting Dark Cores for Performance Optimization via Patterning for Many-core Chips in the Dark Silicon Era”. In: *Int. Symposium on Networks-on-Chip (NOCS)*. 2018. DOI: `10.1109/NOCS.2018.8512169`.

[93] Prashant J. Nair, David A. Roberts, and Moinuddin K. Qureshi. “Citadel: Efficiently Protecting Stacked Memory from Large Granularity Failures”. In: *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*. 2014, pp. 51–62. DOI: `10.1109/MICRO.2014.57`.

[94] Keeyoung Son, Seongguk Kim, Hyunwook Park, Taein Shin, Keunwoo Kim, et al. “Thermal and Signal Integrity Co-Design and Verification of Embedded Cooling Structure With Thermal Transmission Line for High Bandwidth Memory Module”. In: *IEEE Transactions on Components, Packaging and Manufacturing Technology* 12.9 (2022), pp. 1542–1556.

[95] Hyunwoong Kim, Seonghi Lee, Jongcheol Park, Yujun Shin, Seongho Woo, Jongwook Kim, Jaeyong Cho, and Seungyoung Ahn. “Signal Integrity Analysis of Through-Silicon Via (TSV) With a Silicon Dioxide Well to Reduce Leakage Current for High-Bandwidth Memory Interface”. In: *IEEE Transactions on Components, Packaging and Manufacturing Technology* 13.5 (2023), pp. 700–714.

[96] Wei-Hen Lo, Kai-zen Liang, and TingTing Hwang. “Thermal-aware dynamic page allocation policy by future access patterns for Hybrid Memory Cube (HMC)”. In: *Design, Automation & Test in Europe Conference (DATE)*. Mar. 2016, pp. 1084–1089.

[97] Yixian Shen, Leo Schreuders, Anuj Pathania, and Andy D. Pimentel. “Thermal Management for 3D-Stacked Systems via Unified Core-Memory Power Regulation”. In: 22.5s (Sept. 2023). ISSN: 1539-9087. DOI: `10.1145/3608040`. URL: `https://doi.org/10.1145/3608040`.

[98] Shailja Pandey and Preeti Ranjan Panda. “NeuroMap: Efficient Task Mapping of Deep Neural Networks for Dynamic Thermal Management in High-Bandwidth Memory”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 41.11 (2022), pp. 3602–3613. DOI: `10.1109/TCAD.2022.3197698`.

[99] Lokesh Siddhu, Rajesh Kedia, Shailja Pandey, Martin Rapp, Anuj Pathania, Jörg Henkel, and Preeti Ranjan Panda. "CoMeT: An Integrated Interval Thermal Simulation Toolchain for 2D, 2.5D, and 3D Processor-Memory Systems". In: 19.3 (Aug. 2022). ISSN: 1544-3566. DOI: 10.1145/3532185. URL: https://doi.org/10.1145/3532185.

[100] Wei Huang, Shougata Ghosh, Sivakumar Velusamy, Karthik Sankaranarayanan, Kevin Skadron, and Mircea R Stan. "HotSpot: A Compact Thermal Modeling Methodology for Early-Stage VLSI Design". In: *IEEE Trans. Very Large Scale Integration (VLSI) Systems* 14.5 (2006), pp. 501–513. DOI: 10.1109/TVLSI.2006.876103.

[101] *High Bandwidth Memory DRAM(HBM3)*. JEDEC STANDARD. 2022.

[102] Chi-Sung Oh, Ki Chul Chun, Young-Yong Byun, Yong-Ki Kim, So-Young Kim, et al. "22.1 A 1.1V 16GB 640GB/s HBM2E DRAM with a Data-Bus Window-Extension Technique and a Synergetic On-Die ECC Scheme". In: *2020 IEEE International Solid-State Circuits Conference - (ISSCC)*. 2020, pp. 330–332.

[103] Ke Chen, Sheng Li, Naveen Muralimanohar, Jung Ho Ahn, Jay B. Brockman, and Norman P. Jouppi. "CACTI-3DD: Architecture-level modeling for 3D die-stacked DRAM main memory". In: *2012 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 2012, pp. 33–38. DOI: 10.1109/DATE.2012.6176428.

[104] Trevor E Carlson, Wim Heirman, and Lieven Eeckhout. "Sniper: Exploring the Level of Abstraction for Scalable and Accurate Parallel Multi-Core Simulation". In: *High Performance Computing, Networking, Storage and Analysis (SC)*. ACM. 2011. DOI: 10.1145/2063384.2063454.

[105] Sheng Li, Jung Ho Ahn, Richard D Strong, Jay B Brockman, Dean M Tullsen, and Norman P Jouppi. "The McPAT Framework for Multicore and Manycore Architectures: Simultaneously Modeling Power, Area, and Timing". In: *ACM Trans. Arch. and Code Opt. (TACO)* (2013). DOI: 10.1145/2445572.2445577.

[106] Jörg Henkel, Heba Khdr, Santiago Pagani, and Muhammad Shafique. "New Trends in Dark Silicon". In: *Design Automation Conf. (DAC)*. IEEE. 2015.

[107] *Jetson TX2 Module*. URL: https://developer.nvidia.com/embedded/jetson-tx2.

[108] Manjari Gupta, Lava Bhargava, and S Indu. “Dynamic workload-aware DVFS for multicore systems using machine learning”. In: *Computing* 103 (2021), pp. 1747–1769.

[109] Claudiu Buduleci, Arpad Gellert, Adrian Florea, and Remus Brad. “Architectural and Technological Approaches for Efficient Energy Management in Multicore Processors”. In: *Computers* 13.4 (2024). ISSN: 2073-431X. DOI: 10.3390/computers13040084. URL: https://www.mdpi.com/2073-431X/13/4/84.

[110] Yahya H Yassin, Magnus Jahre, Per Gunnar Kjeldsberg, Snorre Aunet, and Francky Catthoor. “Fast and accurate edge computing energy modeling and DVFS implementation in GEM5 using system call emulation mode”. In: *Journal of Signal Processing Systems* 93.1 (2021), pp. 33–48.

[111] Nan Che, Weihua Chen, Puning Zhao, Fei Yu, Zhijun Li, Xing Gao, Yuandi Li, Xiaogang Cui, and Jie Cheng. “OS-Level PMC-Based Runtime Thermal Control for ARM Mobile CPUs”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 43.7 (2024), pp. 2023–2036. DOI: 10.1109/TCAD.2024.3360319.

[112] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K. Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R. Hower, Tushar Krishna, Somayeh Sardashti, Rathijit Sen, Korey Sewell, Muhammad Shoaib, Nilay Vaish, Mark D. Hill, and David A. Wood. “The gem5 simulator”. In: *SIGARCH Comput. Archit. News* 39.2 (Aug. 2011), pp. 1–7. ISSN: 0163-5964. DOI: 10.1145/2024716.2024718. URL: https://doi.org/10.1145/2024716.2024718.

[113] Rafael Ubal, Byunghyun Jang, Perhaad Mistry, Dana Schaa, and David Kaeli. “Multi2Sim: a simulation framework for CPU-GPU computing”. In: *Proceedings of the 21st International Conference on Parallel Architectures and Compilation Techniques*. PACT ’12. Minneapolis, Minnesota, USA: Association for Computing Machinery, 2012, pp. 335–344. ISBN: 9781450311823. DOI: 10.1145/2370816.2370865. URL: https://doi.org/10.1145/2370816.2370865.

[114] Maxime Chéramy, Pierre-Emmanuel Hladik, and Anne-Marie Déplanche. “Simso: A simulation tool to evaluate real-time multiprocessor scheduling algorithms”. In: *5th International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS)*. 2014, 6–p.

[115] Daniel Sanchez and Christos Kozyrakis. “ZSim: fast and accurate microarchitectural simulation of thousand-core systems”. In: *Proceedings of the 40th Annual International Symposium on Computer Architecture.* ISCA ’13. Tel-Aviv, Israel: Association for Computing Machinery, 2013, pp. 475–486. ISBN: 9781450320795. DOI: `10.1145/2485922.2485963`. URL: `https://doi.org/10.1145/2485922.2485963`.

[116] Davy Genbrugge, Stijn Eyerman, and Lieven Eeckhout. “Interval simulation: Raising the level of abstraction in architectural simulation”. In: *HPCA - 16 2010 The Sixteenth International Symposium on High-Performance Computer Architecture.* 2010, pp. 1–12. DOI: `10.1109/HPCA.2010.5416636`.

[117] Ujjwal Gupta, Sumit K. Mandal, Manqing Mao, Chaitali Chakrabarti, and Umit Y. Ogras. “A Deep Q-Learning Approach for Dynamic Management of Heterogeneous Processors”. In: *IEEE Computer Architecture Letters* 18.1 (2019), pp. 14–17. DOI: `10.1109/LCA.2019.2892151`.

[118] Sumit K. Mandal, Ganapati Bhat, Chetan Arvind Patil, Janardhan Rao Doppa, Partha Pratim Pande, and Umit Y. Ogras. “Dynamic Resource Management of Heterogeneous Mobile Platforms via Imitation Learning”. In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 27.12 (2019), pp. 2842–2854. DOI: `10.1109/TVLSI.2019.2926106`.

[119] Ujjwal Gupta, Chetan Arvind Patil, Ganapati Bhat, Prabhat Mishra, and Umit Y. Ogras. “DyPO: Dynamic Pareto-Optimal Configuration Selection for Heterogeneous MpSoCs”. In: *ACM Trans. Embed. Comput. Syst.* 16.5s (Sept. 2017). ISSN: 1539-9087. DOI: `10.1145/3126530`. URL: `https://doi.org/10.1145/3126530`.

[120] Junio Cezar Ribeiro Da Silva, Lorena Leão, Vinicius Petrucci, Abdoulaye Gamatié, and Fernando Magno Quintão Pereira. “Mapping Computations in Heterogeneous Multicore Systems with Statistical Regression on Program Inputs”. In: *ACM Trans. Embed. Comput. Syst.* 20.6 (Oct. 2021). ISSN: 1539-9087. DOI: `10.1145/3478288`. URL: `https://doi.org/10.1145/3478288`.

[121] Bryan Donyanavard, Tiago Mück, Santanu Sarma, and Nikil Dutt. “SPARTA: Runtime task allocation for energy efficient heterogeneous manycores”. In: *2016 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS).* 2016, pp. 1–10.

[122] Xin You, Hailong Yang, Zhibo Xuan, Zhongzhi Luan, and Depei Qian. “PowerSpector: Towards Energy Efficiency with Calling-Context-Aware Profiling”. In: *2022 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. 2022, pp. 1272–1282. DOI: 10.1109/IPDPS53621.2022.00126.

[123] Wim Heirman, Trevor E Carlson, Shuai Che, Kevin Skadron, and Lieven Eeckhout. “Using Cycle Stacks to Understand Scaling Bottlenecks in Multi-Threaded Workloads”. In: *Symp. Workload Characterization (IISWC)*. IEEE. 2011. DOI: 10.1109/IISWC.2011.6114195.

[124] *Intel® Joule™ Developer Kit*. Intel Corporation. 2016.

[125] *Intel Turbo Boost Technology in Intel Core Microarchitecture (Nehalem) Based Processors*. Intel Corporation. 2008.

[126] Santiago Pagani, Heba Khdr, Jian-Jia Chen, Muhammad Shafique, Minming Li, and Jörg Henkel. “Thermal Safe Power (TSP): Efficient Power Budgeting for Heterogeneous Manycore Systems in Dark Silicon”. In: *IEEE Trans. Computers (TC)* 66.1 (2017), pp. 147–162.

[127] *Intel® Xeon® CPU Max Series*. 2023. URL: https://www.intel.com/content/dam/www/central-libraries/us/en/documents/2023-01/xeon-cpu-max-series-product-brief.pdf (visited on 11/12/2023).

[128] Sai Prashanth Muralidhara, Lavanya Subramanian, Onur Mutlu, Mahmut Kandemir, and Thomas Moscibroda. “Reducing memory interference in multicore systems via application-aware memory channel partitioning”. In: *Proceedings of the 44th annual IEEE/ACM international symposium on microarchitecture*. 2011, pp. 374–385.

[129] Arvind Sridhar, Alessandro Vincenzi, Martino Ruggiero, Thomas Brunschwiler, and David Atienza. “3D-ICE: Fast compact transient thermal modeling for 3D ICs with inter-tier liquid cooling”. In: *2010 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 2010, pp. 463–470. DOI: 10.1109/ICCAD.2010.5653749.

[130] Hameedah Sultan and Smruti R. Sarangi. “A fast leakage aware thermal simulator for 3D chips”. In: *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*. 2017, pp. 1733–1738. DOI: 10.23919/DATE.2017.7927273.

[131] Hameedah Sultan, Anjali Chauhan, and Smruti R. Sarangi. "A Survey of Chip-level Thermal Simulators". In: *ACM Comput. Surv.* 52.2 (Apr. 2019). ISSN: 0360-0300. DOI: 10.1145/3309544. URL: https://doi.org/10.1145/3309544.

[132] Ganapati Bhat, Gaurav Singla, Ali K. Unver, and Umit Y. Ogras. "Algorithmic Optimization of Thermal and Power Management for Heterogeneous Mobile Platforms". In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 26.3 (2018), pp. 544–557. DOI: 10.1109/TVLSI.2017.2770163.

[133] Bartłomiej Kocot, Paweł Czarnul, and Jerzy Proficz. "Energy-Aware Scheduling for High-Performance Computing Systems: A Survey". In: *Energies* 16.2 (2023). ISSN: 1996-1073. DOI: 10.3390/en16020890. URL: https://www.mdpi.com/1996-1073/16/2/890.

[134] Jun S. Shim, Bogyeong Han, Yeseong Kim, and Jihong Kim. "DeepPM: Transformer-based Power and Performance Prediction for Energy-Aware Software". In: *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 2022, pp. 1491–1496. DOI: 10.23919/DATE54114.2022.9774589.

[135] Mark Sagi, Martin Rapp, Heba Khdr, Yizhe Zhang, Nael Fasfous, Nguyen Anh Vu Doan, Thomas Wild, Jörg Henkel, and Andreas Herkersdorf. "Long Short-Term Memory Neural Network-based Power Forecasting of Multi-Core Processors". In: *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 2021, pp. 1685–1690. DOI: 10.23919/DATE51398.2021.9474028.

[136] Ayobami Edun, Ruben Vazquez, Ann Gordon-Ross, and Greg Stitt. "Dynamic Scheduling on Heterogeneous Multicores". In: *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 2019, pp. 1685–1690. DOI: 10.23919/DATE.2019.8714804.

[137] Elham Shamsa, Anil Kanduri, Pasi Liljeberg, and Amir M. Rahmani. "Concurrent Application Bias Scheduling for Energy Efficiency of Heterogeneous Multi-Core Platforms". In: *IEEE Transactions on Computers* 71.4 (2022), pp. 743–755. DOI: 10.1109/TC.2021.3061558.

[138] Jian Chen and Lizy K. John. "Efficient program scheduling for heterogeneous multi-core processors". In: *Proceedings of the 46th Annual Design Automation Conference*. DAC '09. San Francisco, California: Association for Computing Machinery, 2009, pp. 927–930.

ISBN: 9781605584973. DOI: 10.1145/1629911.1630149. URL: https://doi.org/10.1145/1629911.1630149.

[139] Kenzo Van Craeynest, Aamer Jaleel, Lieven Eeckhout, Paolo Narvaez, and Joel Emer. “Scheduling heterogeneous multi-cores through performance impact estimation (PIE)”. In: *2012 39th Annual International Symposium on Computer Architecture (ISCA)*. 2012, pp. 213–224. DOI: 10.1109/ISCA.2012.6237019.

[140] Sumit K. Mandal, Ganapati Bhat, Janardhan Rao Doppa, Partha Pratim Pande, and Umit Y. Ogras. “An Energy-aware Online Learning Framework for Resource Management in Heterogeneous Platforms”. In: *ACM Trans. Des. Autom. Electron. Syst.* 25.3 (May 2020). ISSN: 1084-4309. DOI: 10.1145/3386359. URL: https://doi.org/10.1145/3386359.

A. Appendix: Benchmark Suites

The contributions of this thesis were validated using state-of-the-art benchmark applications from the PARSEC and SPLASH-2 benchmark suites. In the following subsections, we describe the characteristics of each of the benchmark suites and their corresponding applications.

A.1. PARSEC: The Princeton Application Repository for Shared-Memory Computers

The Princeton Application Repository for Shared-Memory Computers (PARSEC) is a comprehensive benchmark suite developed to evaluate the performance characteristics of chip-multiprocessors [49]. Unlike prior benchmark suites, which are often skewed toward high-performance computing workloads, PARSEC includes a diverse collection of multi-threaded applications drawn from a range of emerging and conventional domains. This makes it particularly well-suited for evaluating modern and future parallel architectures. The suite was designed to meet several objectives. Firstly, all applications in PARSEC are inherently *parallelized*, reflecting the execution environments of realistic shared-memory systems, which is critical given the focus on cache management in this thesis. Secondly, it includes workloads from a broad array of domains such as computer vision, financial analytics, media processing, machine learning, and scientific computing, ensuring relevance across diverse computational demands. Lastly, PARSEC supports architectural research through the provision of multiple input sets and comprehensive instrumentation capabilities, facilitating detailed simulation, performance evaluation, and experimentation.

PARSEC applications exhibit a wide range of runtime and architectural behaviors, which is essential for the effective evaluation of the proposed cache-aware RM techniques. These behaviors include various parallelization models, e.g., data-parallel, task-parallel, and pipeline approaches, working set sizes that span from kilobytes to several gigabytes, diverse synchronization methods involving locks, barriers, and condition variables, as well as variable levels of memory traffic, with some workloads producing considerable off-chip bandwidth demands particularly when shared cache resources are constrained.

blackscholes Analytically prices European options using the Black-Scholes PDE. It is compute-bound and has minimal synchronization, representing financial analytics workloads.

bodytrack A computer vision application that tracks human body movement using stochastic search techniques. It features significant data sharing and barrier synchronization.

canneal Implements simulated annealing for chip layout optimization. Notable for its aggressive, fine-grained lock-free parallelism and unbounded memory usage.

dedup A data compression kernel mimicking real-world backup systems. It uses a pipelined programming model and exhibits substantial inter-thread communication.

facesim Physically simulates facial expressions using Newton-Raphson iterative methods. This application is representative of physics-based animation systems.

ferret Implements content-based similarity search using image feature vectors. It employs pipeline parallelism and represents next-generation multimedia search engines.

fluidanimate Simulates incompressible fluid flow using Smoothed Particle Hydrodynamics (SPH). The program benefits from cache line streaming effects.

freqmine Performs frequent itemset mining with a FP-growth variant. It features large working sets and intense use of shared structures like prefix trees.

streamcluster Solves the online clustering problem for streaming data. Notable for moderate sharing and memory bandwidth requirements.

swaptions Uses Monte Carlo simulations under the Heath-Jarrow-Morton model to price interest rate derivatives. This application is both compute- and memory-bound.

vips A high-performance image processing system derived from a print-on-demand service. Performs fundamental transformations like convolution and affine transforms.

x264 A widely used H.264/AVC encoder. Features stage-wise thread specialization and significant inter-thread data dependency through reference frame sharing.

Each benchmark provides six input sets:

- `test`, `simdev` — small sets for functionality and simulator testing.
- `simsmall`, `simmedium`, `simlarge` — increasingly large sets for scalable simulation studies.
- `native` — used for performance evaluation on physical hardware.

These sets allow the user to tailor experiments depending on the available computational resources and the desired fidelity. In the context of this dissertation, the input size recommendation is followed. Mainly, the *simmedium* and *simlarge* input sizes were used on the simulation platforms, i.e., Hot-Sniper and CoMeT, while the *native* input size was used in the real hardware experiments on the Intel Alder Lake.

A.2. SPLASH-2: Stanford ParalleL Applications for SHared Memory

The SPLASH-2 (Stanford Parallel Applications for Shared memory) benchmark suite was developed to provide a standardized collection of parallel applications and computational kernels suitable for evaluating shared-address-space multiprocessor systems. The suite includes programs that cover a broad spectrum of scientific, engineering, and graphics workloads, and it is designed to support investigations into aspects such as cache coherence, memory hierarchy design, processor scalability, and parallel algorithm performance. SPLASH-2 comprises both full-scale applications and computational kernels. In total, it includes 12 programs: 8 applications and 4 kernels. These programs were chosen or redesigned to (i) be representative of important workloads, (ii) utilize improved and scalable algorithms, and (iii) be optimized for architectural realism in shared memory systems.

Barnes Barnes simulates the interaction of a system of bodies using the Barnes-Hut N-body method in three dimensions. The domain is modeled as an octree, with internal nodes representing space cells and leaves containing multiple particles. Computation is dominated by partial octree traversals for force calculations. The communication pattern is irregular and input-dependent. Data is not optimized for memory locality beyond per-leaf grouping.

Cholesky This is a blocked, sparse Cholesky factorization kernel, which decomposes a sparse matrix into a lower triangular matrix and its transpose. Unlike LU, it does not use global synchronization between steps and exhibits a relatively high communication-to-computation ratio due to sparsity.

FFT The Fast Fourier Transform (FFT) kernel implements a one-dimensional complex radix-n six-step algorithm optimized for reduced interprocessor communication. The major communication overhead stems from three distributed transpose phases requiring all-to-all communication, managed using blocked submatrix transfers to exploit spatial locality.

FMM The Fast Multipole Method (FMM) application simulates interactions among bodies in two dimensions. It performs upward and downward traversals of a hierarchical tree structure to model distant interactions via cell-cell computations and local interactions directly. The access pattern is highly unstructured.

LU LU decomposes a dense $n \times n$ matrix into a product of lower and upper triangular matrices using block decomposition. Matrix blocks are assigned to processors in a 2D scatter fashion to minimize communication and enhance data locality. The block size must strike a balance between locality and load balancing.

Ocean Ocean models large-scale ocean currents using a red-black Gauss-Seidel solver. It improves over the original SPLASH version by partitioning the domain into square-like subgrids and using contiguous memory allocation per processor to improve cache performance.

Radiosity This application performs hierarchical radiosity computations for realistic image synthesis. The scene is composed of polygons subdivided into patches, with interactions tracked and recursively refined for accuracy. The computation features highly irregular access patterns and data structures such as quadtrees and BSP trees.

Raytrace Raytrace renders a 3D scene using ray tracing with hierarchical uniform grids for spatial acceleration. Rays are traced through pixels and recursively followed through intersections with objects. The task allocation is dynamic and handled by distributed queues with work stealing. Memory access patterns are highly unpredictable.

Volrend Volrend implements volume rendering via ray casting through a voxel grid, supported by an octree for early ray termination and acceleration. Each ray samples along its path to produce color values. Accesses are input-dependent and irregular. The workload is parallelized using image-space partitioning and task queues.

Radix Radix is an integer sorting kernel that implements radix sort with global histogram accumulation and redistribution phases. All-to-all communication is necessary in the permutation step, which involves sender-determined writes. The performance is limited by parallel prefix steps that are not fully parallelizable.

Water-Nsquared This molecular dynamics simulation models water molecules using an $O(n^2)$ algorithm to compute pairwise forces. It uses a predictor-corrector integrator and updates shared acceleration arrays through improved synchronization. Each process maintains local acceleration vectors before updating global values.

Water-Spatial Similar to Water-Nsquared, this version uses a uniform spatial grid to limit interactions to nearby cells, resulting in $O(n)$ complexity. The algorithm benefits from reduced computation time and localized communication, though data migration across cells incurs additional overhead.

Each application in the SPLASH-2 suite is characterized by specific architectural behaviors. In terms of load balancing and concurrency, applications such as `Barnes`, `FMM`, and `Ocean` scale well, while others like `Radiosity` and `Cholesky` suffer from load imbalance at higher core counts. Most applications exhibit working sets between 64KB and 1MB, though larger working sets in `Ocean`, `Raytrace`, and `Radix` may exceed typical cache sizes depending on problem parameters. Bandwidth demands also vary, with `FFT` and `Radix` being bandwidth-intensive, whereas `LU` and `Water-Spatial` demonstrate moderate communication overheads. Spatial locality is application-dependent; structured codes like `LU` benefit from long cache lines, while irregular ones such as `Radiosity` and `Raytrace` are prone to false sharing and fragmentation.

Overall, the SPLASH-2 suite offers a diverse and realistic set of programs for shared memory multiprocessor evaluation, and careful selection of parameters (problem size, processor count, and memory configuration) is essential for producing meaningful architectural insights.