PUCRS

ESCOLA POLITÉCNICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO
DOUTORADO EM CIÊNCIA DA COMPUTAÇÃO

IAÇANÃ IANISKI WEBER

ENHANCING LIFETIME RELIABILITY OF MANYCORE SYSTEMS THROUGH REINFORCEMENT LEARNING-BASED TASK MANAGEMENT

Porto Alegre

2024

PÓS-GRADUAÇÃO - *STRICTO SENSU*

Pontifícia Universidade Católica
do Rio Grande do Sul

PONTIFICAL CATHOLIC UNIVERSITY OF RIO GRANDE DO SUL
SCHOOL OF TECHNOLOGY
COMPUTER SCIENCE GRADUATE PROGRAM

ENHANCING LIFETIME RELIABILITY OF MANYCORE SYSTEMS THROUGH REINFORCEMENT LEARNING-BASED TASK MANAGEMENT

IAÇANÃ IANISKI WEBER

Doctoral Thesis submitted to the Pontifical Catholic University of Rio Grande do Sul in partial fulfillment of the requirements for the degree of Ph. D. in Computer Science.

Advisor: Prof. Dr. Fernando Gehm Moraes

Porto Alegre
2024

Ficha Catalográfica

Elaborada pelo Sistema de Geração Automática de Ficha Catalográfica da PUCRS
com os dados fornecidos pelo(a) autor(a).
Bibliotecária responsável: Clarissa Jesinska Selbach CRB-10/2051

IAÇANÃ IANISKI WEBER

ENHANCING LIFETIME RELIABILITY OF MANYCORE SYSTEMS THROUGH REINFORCEMENT LEARNING-BASED TASK MANAGEMENT

This Doctoral Thesis has been submitted in partial fulfillment of the requirements for the degree of Ph. D. in Computer Science, of the Computer Science Graduate Program, School of Technology of the Pontifical Catholic University of Rio Grande do Sul

Sanctioned on 5th March, 2024.

COMMITTEE MEMBERS:

Prof. Dr. Márcio Eduardo Kreutz (PPgSC/UFRN)

Profª. Drª. Fernanda Gusmão de Lima Kastensmidt (PGMICRO/UFRGS)

Prof. Dr. Fabiano Passuelo Hessel (PPGCC/PUCRS)

Prof. Dr. Fernando Gehm Moraes (PPGCC/PUCRS - Advisor)

AGRADECIMENTOS

-

OTIMIZANDO A CONFIABILIDADE E VIDA ÚTIL DE SISTEMAS MANYCORE POR MEIO DE GERENCIAMENTO DE TAREFAS BASEADO EM APRENDIZADO POR REFORÇO

RESUMO

Esta pesquisa aborda o desafio de melhorar a confiabilidade ao longo do tempo em sistemas manycore, uma questão crítica em microeletrônica. O estado da arte atual em Técnicas de Gerenciamento Térmico Dinâmico (DTM, do inglês, *Dynamic Thermal Management*) e Gerenciamento Dinâmico de Confiabilidade (DRM, do inglês, *Dynamic Reliability Management*) apresenta as seguintes lacunas: (*i*) subutilização do sistema em abordagens estáticas ou adoção de heurísticas complexas; (*ii*) trabalhos que focam somente em temperatura (DTM) ou confiabilidade (DRM); (*iii*) propostas que consideram poucos efeitos de envelhecimento. O objetivo principal desta Tese é abordar a questão da degradação precoce em sistemas manycore resultante de efeitos de desgaste acelerados por temperatura, abrangendo o desenvolvimento e a execução de estratégias para gerenciar tarefas de forma que mitiguem esses efeitos. A afirmação central da Tese é que o gerenciamento de tarefas baseado em aprendizado por reforço (RL, do inglês, *Reinforcement Learning*) pode melhorar a confiabilidade de sistemas manycore ao longo do tempo. A pesquisa adota uma abordagem inovadora utilizando um algoritmo de RL para gerenciamento de tarefas. Este método envolve a construção de modelos para prever a degradação do sistema e modificar dinamicamente as alocações de tarefas para minimizar o desgaste a longo prazo. A pesquisa utiliza simulações para verificar a eficácia dos modelos e algoritmos desenvolvidos. A contribuição significativa desta Tese é a criação da "Heurística de Aprendizado Ciente da Taxa de Falhas no Tempo para Alocação de Aplicações" (FLEA, do inglês, *Failure In Time-aware Learning Heuristic for Application Allocation*), que gerencia temperatura e confiabilidade concomitantemente. Os resultados mostram que a proposta FLEA reduz a taxa de degradação do sistema em comparação com abordagens convencionais de gerenciamento de tarefas. Os resultados apresentam melhora na confiabilidade e no tempo de vida útil do sistema. A FLEA representa um avanço no gerenciamento de sistemas, combinando técnicas de aprendizado por reforço com estratégias de gerenciamento de tarefas para aumentar proativamente o tempo de vida útil. Esta Tese oferece direções de pesquisa no tema do projeto e gerenciamento de manycores. Ela indica o caminho para o desenvolvimento de modelos de aprendizado por reforço mais sofisticados para gerenciamento de sistemas.

Palavras-Chave: Sistemas Manycore, Confiabilidade, Aprendizado por Reforço, Gerenciamento de Temperatura, Gerenciamento de Confiabilidade.

ENHANCING LIFETIME RELIABILITY OF MANYCORE SYSTEMS THROUGH REINFORCEMENT LEARNING-BASED TASK MANAGEMENT

ABSTRACT

This research tackles the challenge of improving the lifetime reliability of manycore systems, a critical issue in microelectronics. The current state-of-the-art in Dynamic Thermal Management (DTM) and Dynamic Reliability Management (DRM) techniques present the following gaps: (*i*) system underutilization in patterning approaches or adoption of complex heuristics; (*ii*) works focusing only on temperature (DTM) or reliability (DRM); (*iii*) proposals considering few aging effects. The primary goal of this Thesis is to address the issue of early degradation in manycore systems resulting from temperature-amplified wear-out effects, encompassing the development and execution of strategies to manage tasks in ways that mitigate these effects. The central claim of the Thesis is that task management based on reinforcement learning (RL) can enhance manycore systems lifetime reliability. The research adopts an innovative approach using an RL algorithm for task management. This method involves building models to predict system degradation and dynamically modifying task allocations to minimize long-term wear. The research employs simulations to verify the effectiveness of the developed models and algorithms. The significant contribution of this Thesis is the creation of the "Failure In Time-aware Learning Heuristic for Application Allocation" (FLEA), which manages temperature and reliability concomitantly. Results show that FLEA lowers the rate of system degradation compared to conventional task management approaches. The results data present an enhancement in system reliability and lifetime. FLEA represents an advancement in management, combining reinforcement learning techniques with task management strategies to proactively increase lifetime. This Thesis provides insights into the design and management of manycores. It paves the way for developing more sophisticated reinforcement learning models for systems management.

Keywords: Manycore Systems, Lifetime Reliability, Reinforcement Learning, Temperature Management, Reliability Management.

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS

API – Application Programming Interface

APP_{rep} – Application Repository

C – Capacitance

CBGA – Ceramic Ball Grid Array

CNI – Core Network Interface

CPI – Cycles per Instruction

CTG – Communicating Task Graphs

CTM – Compact Thermal Models

DMA – Direct Memory Access

DNN – Deep Neural Network

DPM – Dynamic Power Manager

DRM – Dynamic Reliability Management

DSSOCS – Domain-specific System-on-Chips

DTM – Dynamic Thermal Management

DTW – Dynamic Time Warping

DVFS – Dynamic Voltage and Frequency Scaling

DVS – Dynamic Voltage Scaling

EDA – Electronic Design Automation

EM – Electromigration

EV – Electron Volts

F – Frequency

FIT – Failure in Time

FET – Field Effect Transistor

FLEA – FIT-aware Learning Heuristic for Application Allocation

GHZ – Gigahertz

GPPE – General Purpose Processing Elements

GUI – Graphical User Interface

HC – Hot Carrier

IC – Integrated Circuit

ILP – Integer-Linear Programming

IP – Intellectual Property

IPCM – Intel's Performance Counter Monitor

ISS – Instruction Set Simulator

J – Joule

K – Kelvin

LBRM – Lifetime Budgeting Reliability Management

LF – Longevity Framework

LUT – Lookup Table

M – Meter

MAC – Multiply and Accumulate

MCSOC – Manycore System on Chip

MF – Memory Factor

MIG – Migration

MILP – Mixed-Integer Linear Programming

MMR – Memory Mapped Register

MLP – Multi-Layer Perception

MPE – Manager PE

MTTF – Mean Time to Failure

NM – Nanometer

NBTI – Negative Bias Temperature Instability

NI – Network Interface

NOC – Network-on-Chip

MPE – Manager Processing Element

ODA – Observe-Decide-Act

OP – Open Platform

OS – Operating System

OVP – Open Virtual Platform

P – Power

PBS – PE bin state

PC – Program Counter

PE – Processing Element

PID – Proportional Integral and Derivative

PMCA – Programmable Manycore Accelerator

R – Resistance

RA – Reliability Aware

RC – Resistor-Capacitor

REA – Reliability Estimator Accelerator

RL – Reinforcement Learning

RM – Reliability-aware Mapping

RMS – Root Mean Square

RTL – Register Transfer Level

SDF – Standard Delay File

SEUS – Single-Event Upsets

SF – Stress Factor

SG – Specific Goal

SI – Silicon

SIO_2 – Silicon Dioxide

SM – Stress Migration

SOC – System-on-Chip

SOFR – Sum-of-Failure-Rates

TASA – Threshold Accepting Simulated Annealing

TC – Thermal Cycling

TCF – Toggle Count Format

TCTS – Temperature Constrained Task Selection

TDP – Thermal Design Power

TDDB – Temperature Dependent Dielectric Breakdown

TEA – Temperature Estimator Accelerator

TPC – Task Power Category

TSP – Thermal Safe Power

V – Voltage

VF – Voltage-Frequency

VLSI – Very Large-scale Integration Systems

VRF – Vacancy Reliability Factor

W – Watt

CONTENTS

1. INTRODUCTION

In the initial phases of computing, single-core processors saw notable performance gains. However, they reached limits in instruction-level parallelism and encountered significant power and temperature issues with increased frequencies. With the advent of transistor miniaturization, as predicted by Moore in 1965 [Moore, 1965], the industry transitioned to a manycore approach [Sutter, 2005]. This shift addressed the growing need for enhanced performance, leading to the development of systems with numerous Processing Elements (PEs), known as manycore systems. Networks-on-Chip (NoCs) emerged as the favored method for intra-chip communication, offering superior performance and scalability compared to traditional bus or crossbar architectures in manycore contexts [Borkar, 2007]. These systems capitalize on task parallelism to boost overall performance.

The concept of *power density* in integrated circuits (ICs), defined as the power dissipated per unit area, is of critical importance. Dennard's scaling theory suggests that reducing transistor size should stabilize power density, as lower operating voltages proportionally decrease power consumption to the square of the voltage applied [Dennard et al., 1974]. However, this trend predicted by Dennard has not been sustained. The failure to scale down voltages proportionately with transistor gate length has increased power density, contrary to expectations of stability [Bohr, 2007].

An increase in power density leads to higher heat generation per unit area. This heat must be dissipated to prevent the chip from operating unreliably, aging prematurely, or experiencing permanent failure should the temperature surpass certain thresholds. Therefore, ICs must operate within a specified *power budget* to ensure safe functionality [Haghbayan et al., 2014]. In modern manycore systems, running all PEs at full capacity is not feasible without exceeding this power budget. As a result, certain parts of the chip must remain inactive. This inactive portion is commonly referred to as *dark silicon* [Esmaeilzadeh et al., 2011].

Additionally, a rise in power density can result in *hotspots*, a localized overheating. These hotspots can develop when multiple PEs are active in a concentrated area, which may arise even if the overall chip remains within its power budget. To address this issue, a patterning approach has been proposed [Liu et al., 2018]. This strategy involves configuring PEs in an alternating pattern of active and inactive (dark) cores, creating a chessboard-like layout. This arrangement helps in mitigating the creation of hotspots.

The described patterning approach, while effective in reducing hotspots, introduces disadvantages. These include the under-utilization of the system's resources [Wen et al., 2020] and an increase in hop count between tasks [Karkar et al., 2022], leading to higher application latency. Thus, these limitations motivated researchers to explore alternative strate-

gies. The objective of these new strategies is to maximize resource utilization while adhering to the thermal and power constraints imposed by the presence of dark silicon.

Dynamic Thermal Management (DTM) represents a prominent category of solution, as demonstrated by numerous research works [Pagani et al., 2017, Wang et al., 2018, Ranjbar et al., 2019, Rahimipour et al., 2020, Silva et al., 2020, Kim et al., 2020, Pourmohseni et al., 2022, Chen et al., 2023], among others. DTM techniques are designed to dynamically modulate system activity and performance in compliance with temperature constraints. This is achieved through strategies such as allocating tasks to PEs to prevent hotspot formation, relocating tasks away from PEs that are overheating, and managing the workload in real-time. Such approaches enable DTM to minimize system under-utilization while ensuring compliance with established thermal and power limitations.

However, the long-term impact on system reliability is often not considered. To address this issue, Dynamic Reliability Management (DRM) techniques have gained increasing relevance [Das et al., 2016, Sahoo et al., 2019, Wang et al., 2019, Namazi et al., 2019, Haghbayan et al., 2020a, Rathore et al., 2021, Zhang et al., 2023]. These DRM schemes integrate the consideration of a system's lifetime reliability into their strategies. They enable decisions that address immediate thermal problems and consider the overall lifetime of manycore systems. Incorporating reliability into the task management cost function poses a challenge since estimating circuit degradation requires intensive computation, and adding reliability sensors leads to an area overhead. As highlighted in [Sahoo et al., 2021], most studies have not effectively demonstrated the scalability of their proposed methods with the increase in the number of PEs.

1.1 Motivation

The pursuit of enhanced computational performance has given rise to manycore systems, which house numerous PEs on a single die. This advancement, however, introduces significant challenges in keeping system reliability and thermal efficiency. High temperatures can adversely affect the lifetime of these systems by accelerating the aging process in semiconductor devices. Consequently, developing robust and intelligent task management solutions has become increasingly crucial to improve these systems' lifetime and dependability.

Reinforcement Learning (RL) presents an effective approach to address the complexities associated with thermal and reliability challenges in manycore architectures. Unlike static models or predefined heuristics, RL promotes the creation of task management policies through iterative learning. These policies undergo continuous refinement during their training phase and are theoretically guaranteed to converge [Watkins, 1989, Tsitsiklis, 1994, Even-Dar and Mansour, 2003]. This convergence leads to optimized decision-making

in the inference phase, effectively maintaining operational temperatures within safe thermal limits and minimizing hardware wear-out.

The motivation for this Thesis is the challenge of devising a scalable approach for managing reliability and thermal issues in manycore systems using the RL technique. This Thesis integrates experimental simulations to assess and corroborate the effectiveness of the proposed RL-based task management strategy in enhancing the lifetime reliability of manycore systems. By exploring the interaction between thermal behavior and system reliability, this research aims to lay the groundwork for developing future resilient, adaptive, and efficient manycore architectures.

1.2 Thesis Statement

This thesis seeks to demonstrate the feasibility of developing scalable Dynamic Thermal Management (DTM) and Dynamic Reliability Management (DRM) heuristics for manycore systems with hundreds of PEs, executing a dynamic workload. The focus is on achieving this with minimal computational overhead, utilizing a machine learning technique of reinforcement learning (RL).

1.3 Objectives

The strategic objective of this Thesis is to specify, develop, validate, and optimize lightweight management strategies (DTM and DRM) for manycore systems, using application mapping and task migration as the main actuation strategies for these strategies.

The *specific goals* (SG) of the Thesis include:

SG1 Development of a manycore abstract platform. This platform should be parameterizable and designed to interconnect the PEs through a NoC. It must offer accuracy at the instruction level rather than at the clock level, ensuring that the behavior of the NoC closely mirrors that observed at the RTL level. A crucial requirement for this platform is efficient simulation time. This efficiency is imperative since evaluating temperature and reliability requires simulating applications over extended periods, typically measured in seconds.

SG2 Software environment for the platform that allows multitasking and dynamic workload (i.e., admission of new applications at any time during the simulation). For this purpose, a standard operating system, FreeRTOS, has been selected. This operating system received inter-task communication and system management modules.

SG3 Research and develop a method for applying RL to system management. This includes defining the state-action space, designing the reward function, and selecting the appropriate learning parameters to guide the RL algorithm through its training phase until it achieves convergence.

SG4 Development of a runtime management technique that utilizes the RL learned policy to guide decisions in application allocation and task migration. Application allocation involves two steps: (*i*) selecting a region of the system for executing the application, with the selection based on the temperature of the PEs, using this temperature as a cost function; (*ii*) mapping tasks within the chosen region, employing RL for this purpose. Task migration is the actuation mechanism to reduce hotspots, thereby increasing the system's lifetime. Task migration also leverages the RL technique to redistribute tasks in response to thermal events effectively.

SG5 Comparison of the proposed technique with state-of-the-art temperature and lifetime reliability approaches.

SG6 Evaluate the scalability of the proposal with systems containing up to 400 PEs.

1.4 Original Contribution

The original contributions of this Thesis are:

1. A novel abstract manycore platform, named Chronos-V, that simulates manycore systems with dozens of PEs. It can perform simulations lasting seconds, thus enabling system validation;
2. A novel RL-based reliability management technique called FLEA. It employs a lightweight policy lookup table trained by a reinforcement learning algorithm at the design phase. It utilizes the PE failure in time (FIT) to define a reward function to enhance the system's reliability.

Other contributions provided by this Thesis include:

- Adaptation of FreeRTOS to support dynamic workload and execution of applications described as communicating task graphs (CTG);
- Power calibration method on the abstract platform;
- Adaptation of the temperature estimator module, TEA, initially developed in RTL, to the abstract platform;
- Method of creating a *Q*-table (table used by the RL) for use in DTM and DRM;

- Lifetime reliability assessment considering a comprehensive set of aging effects;
- Implementation of state-of-the-art DTM and DRM techniques on the abstract platform.

1.5 Document Organization

This document is structured into seven Chapters. This first Chapter introduced the motivation, Thesis statement, objectives, and original contributions.

- Chapter 2 details the Chronos-V manycore model, which is the first original contribution of the Thesis. This Chapter establishes the basis for understanding manycore systems, the primary focus of this Thesis. It is positioned early in this document for two reasons: firstly, to enable the evaluation of temperature and reliability, a manycore model capable of long-duration simulations (seconds, and not milliseconds as in RTL platforms) is essential. Secondly, it separates the platform infrastructure from the DTM and DRM strategies. **This Chapter fulfills *SG1* and *SG2*.**
- Chapter 3 provides the background knowledge for understanding the DTM and DRM strategies. It covers topics such as Dark Silicon, runtime temperature estimation, and reliability concerns in integrated circuits.
- Chapter 4 reviews relevant literature related to both DTM and DRM. This Chapter positions the Thesis in relation to the current state-of-the-art and identifies existing gaps in the literature.
- Chapter 5 details the main original contribution of the Thesis, introducing a novel mapping technique named “Failure In Time-aware Learning Heuristic for Application Allocation” (FLEA). FLEA, based on reinforcement learning (RL), is designed to improve the lifetime reliability of manycore systems under dynamic workloads. **This Chapter fulfills *SG3* and *SG4*.**
- Chapter 6 comprehensively evaluates FLEA using the platform described in Chapter 2. This Chapter evaluates temperature behavior, hop count between communicating tasks, and reliability assessment, comparing FLEA with state-of-the-art heuristics. **This Chapter fulfills *SG5* and *SG6*.**
- Chapter 7 concludes the Thesis and outlines potential directions for future research.

2. CHRONOS-V MANYCORE MODEL

This Chapter presents the design of Chronos-V[1], *the first original contribution of this Thesis.* Chronos-V is a simulation environment for manycore systems-on-chip (MCSoC) architectures, combining a RISC-V instruction-set simulator with an abstract 2D-mesh Network-on-Chip (NoC) model. Its purpose is to aid developers in parallelizing hardware and software development cycles, thus accelerating the time-to-market for new products.

Chronos-V offers to developers a software-centric model that allows for code iterations and application testing in the early stages of product development. It accelerates software development cycles and tackles system management challenges in MCSoC designs. The platform provides developers with critical control features, including Dynamic Voltage and Frequency Scaling (DVFS) and task allocation. Chronos-V integrates runtime management techniques with observe-decide-act (ODA) capabilities to mitigate issues like dark silicon and reliability degradation, as will be discussed in Chapter 5.

This Chapter is based on the publication:

Iaçanã Ianiski Weber, Angelo Elias Dal Zotto and Fernando Gehm Moraes
Chronos-V: a Manycore High-level Model with Support for Management Techniques
Analog Integrated Circuits and Signal Processing, vol. 117, pages 57–71. 2023.
https://link.springer.com/article/10.1007/s10470-023-02190-8

The Chapter begins with Section 2.1, exploring related works on manycore frameworks such as Register Transfer Level (RTL) manycore platforms, FPGA-based platforms, and abstract platforms. Section 2.2 provides a comprehensive overview of the Chronos-V platform, explaining its three layers and their operation. Section 2.3 discusses the quantum-based simulation model. Section 2.4 presents the evaluation of the platform, comparing it with an RTL model. Section 2.5 concludes this Chapter.

2.1 Related Work on Manycore Frameworks

Due to the importance of simulation capability, several simulators with distinct simulation methodologies have been developed over the years. This section reviews and discusses manycore platforms for hardware exploration and software testing. The literature presents manycore platforms with several design goals and features. Table 2.1 provides a comparative summary of the efforts in the field, positioning our work in the last table row. The respective works have been categorized into six key areas for a comprehensive analysis:

[1] Available at: https://github.com/iacanaw/Chronos-RISCV

1. *Simulation Level*: This category reflects the degree of abstraction at which a certain platform operates, ranging from detailed hardware models to higher-level approximations.
2. *Platform Software*: This is subdivided into User-Level and OS-Level simulation capabilities. It highlights whether a platform is limited to simulating 'bare metal' applications (those running directly on hardware without an operating system) or if it supports simulation environments that include operating systems.
3. *Heterogeneity*: This indicates the system's ability to integrate and leverage diverse computational resources such as various processors or specialized peripherals, thus reflecting its adaptability to different types of computational tasks.
4. *Processor Architecture*: This category presents the processor architectures for which the framework has been tailored, showcasing the platform's versatility or focus regarding hardware design.
5. *Peripheral Support*: The framework can handle additional hardware components like accelerators, UART, network interfaces, etc., which are essential for a complete system simulation.
6. *Communication Infrastructure*: This aspect regards the methods by which PEs within the platform communicate, being a vital feature for assessing the efficiency and scaling capabilities of multi-processing environments.

Table 2.1: Related work on manycore frameworks.

Proposal		Simulation Level	Platform Software	Heterogeneity	Processor Architecure	Peripheral	Communication
RVNoC	[Elmohr et al., 2018]	RTL	-	Yes	RISC-V	Yes	NoC
Memphis	[Ruaro et al., 2019]	RTL	OS	Yes	Plasma	Yes	NoC
Savas et al.	[Savas et al., 2020]	RTL	User	Yes	RISC-V	No	NoC
Khamis et al.	[Khamis et al., 2022]	RTL	User	Yes	RISC-V	Yes	NoC
ProNoC	[Monemi et al., 2017]	FPGA	User	Yes	Multiple	Yes	NoC
HERO	[Kurth et al., 2018]	FPGA	OS	Yes	ARM and RISC-V	No	Bus and NoC
Chipyard	[Amid et al., 2020]	FPGA	User	Yes	RISC-V	Yes	Bus and Crossbar
ANDROMEDA	[Merchant et al., 2021]	FPGA	User	No	RISC-V	No	NoC
Kamaleldin et al.	[Kamaleldin and Göhringer, 2021]	FPGA	User	Yes	RISC-V (32/64 bits)	No	NoC
Uhlendorf et al.	[Uhlendorf et al., 2021]	FPGA	User	No	Nios II	No	NoC
Sniper	[Carlson et al., 2011]	Abstract	Emulated OS	No	x86	No	-
Subutai	[Cataldo et al., 2018]	Abstract[1]	OS	No	-	No	NoC
MPSoCSim	[Real et al., 2016]	Abstract	OS	Yes	Multiple	Yes	NoC[1]
One-IPC	[Uddin, 2017]	Abstract	User	No	Multiple	No	-
Prophet	[Zhang et al., 2017]	Abstract	OS	No	-	No	NoC[1]
Mack et al.	[Mack et al., 2020]	Abstract	User	Yes	Multiple	Yes	Shared Memory
FLECSim-SoC	[Hotfilter et al., 2021]	Abstract	OS	Yes	RISC-V	Yes	Crossbar
This Work		**Abstract**	**OS**	**Yes**	**RISC-V**	**Yes**	**NoC**

[1] cycle-accurate simulation

2.1.1 RTL Platforms

Elmohr et al. [Elmohr et al., 2018] have developed the RVNoC framework, a tool for constructing manycore models in synthesizable RTL language, leveraging the versatile RI5CY core and the RISC-V architecture to facilitate the creation of customizable network on chip (NoC) systems. RISC-V PEs are at the base of this framework, interconnecting them and peripheral devices via the AXI4 bus, ensuring efficient communication and extensibility. The RVNoC framework also employs configurable routers - adopting flit-based communication to efficiently manage network packets, which, the authors claim enhances the overall latency and power consumption.

The authors provided details about the Core Network Interface (CNI) as well as the comprehensive simulation environment they've established for performance analysis. Their implementation of CNI utilizes memory-mapped I/O for seamless core communication, without the need to modify existing architectures. Moreover, the inclusion of traffic generation tools and software controllers enables benchmarking across various network configurations. The authors conclude that it is a powerful framework that doesn't just facilitate the conception of MCSoC designs but also provides practical simulation data on critical performance metrics such as latency and throughput, thereby facilitating the development process for high-performance, low-power, NoC-based multiprocessor applications.

Ruaro et al. [Ruaro et al., 2019] introduce Memphis, a flexible and comprehensive Electronic Design Automation (EDA) framework with a manycore model designed for heterogeneous System-on-Chips (SoCs) at the RTL-level. Memphis employs PE tiles which include processor, network interface, router, and memory. Notably, its hardware model, designed in both SystemC and VHDL, is cycle-accurate, offering precise simulation and FPGA prototyping capabilities. The framework is equipped with an extensive suite of graphical debugging tools that provide an intuitive means for developers to gain insights into runtime computational and communication events. The author's goal is to position Memphis as a versatile tool for both research pursuits and educational contexts, demonstrated by its utility through various case studies that explore manycore generation, simulation, and debugging.

Memphis manycore model includes in its hardware configuration support for both general-purpose processing cores and specialized peripherals. Its decentralized management concept enables scalability by organizing the processing elements into clusters with a dedicated manager for each, optimizing resource allocation and application deployment. Additionally, the application model utilizes acyclic task graphs for defining communication flows, with support for both real-time and best-effort tasks managed by a task scheduler using the Least Slack Time algorithm. Furthermore, the dynamic application injection protocol ensures that new applications can be seamlessly integrated into the system at runtime.

Savas et al. [Savas et al., 2020] offer a robust approach for generating application-specific manycore architectures with enhanced RISC-V processors, addressing the performance demands of modern computing tasks that leverage parallelism and specialization. The framework allows PEs customization by adding instruction extensions and custom accelerators. It employs software tools to output synthesizable Verilog code, encompassing PEs, NoC, and accelerators and validates system performance using a cycle-accurate emulator. Developers must create applications using a high-level dataflow language. As outlined in their work, Savas et al.'s framework integrates these accelerators with a 2D mesh NoC using the open-source Rocket Chip generator. The use of a cycle-accurate emulator and the generation of synthesizable Verilog code ensure that the platform can be effectively validated for system performance.

Khamis et al. [Khamis et al., 2022] propose an emulation framework for exploring manycore architectures, supporting both 2D and 3D NoCs configurations with RISC-V processors. The framework automates the HDL and verification generation models. It encompasses processes such as compilation, simulation, synthesis, emulation, and performance reporting, facilitating the creation and verification of manycore systems. The authors use the AXI bus interface in the framework, offering connectivity for peripherals and allowing for the replacement of processor tiles. The framework can handle real traffic patterns, communicate through the AXI4 bus interface, and support scalability through its emulation co-modeling approach.

2.1.2 FPGA Platforms

Monemi et al. [Monemi et al., 2017] present ProNoC, an integrated framework designed for the rapid prototyping and validation of NoC-based MCSoC architectures specifically targeting FPGA devices. It incorporates advanced NoC features including support for virtual channels, and virtual networks, as well as implementing a variety of routing algorithms designed to optimize the NoC performance. Furthermore, ProNoC facilitates user interaction via a graphical user interface (GUI), streamlining the development process of MCSoC prototypes. ProNoC's design also includes an adaptable NI that works seamlessly with different processor cores and peripherals, a feature that promotes flexibility in MCSoC designs. The NI, integrated with direct memory access (DMA) channels and optional error-checking CRC32 codes, illustrates the authors' concern for reliable NoC-based MCSoCs. Additionally, the framework allows for integration with different peripherals, such as aeMB, LatticeMicro32, UART modules, and Ethernet modules.

Kurth et al. [Kurth et al., 2018] introduce HERO, a heterogeneous manycore platform on an FPGA. The platform includes a host processor (ARM Cortex-A) with a programmable manycore accelerator (PMCA) based on the RISC-V ISA to enhance process-

ing capabilities. The ARM processor, which works as the orchestrating host, manages the PMCA, which contains a 2D mesh NoC of RISC-V cores configured in a multi-cluster design to execute parallel operations with ultra-low power consumption. Through the interconnection, the host and PMCA share the main memory, promoting effective data exchange.

The HERO platform has a software stack that helps integrate the PMCA into the host environment. It allows for the customized generation of machine code for ARM and RISC-V ISAs from a single source, ensuring smooth operation. With the use of OpenMP programming and shared virtual memory, developers can offload computational tasks to the PMCA, resulting in fine-grained parallelism.

Amid et al. [Amid et al., 2020] introduce Chipyard, an integrated framework for the agile development of manycore systems. The toolset supports the assembly of customizable SoCs by providing composable, open-source intellectual property (IP) blocks and generating synthesizable RTL code. Chipyard's RTL code can be verified through FPGA-based or software-based simulation platforms, including VCS and Verilator, demonstrating the framework's practical applications for FPGA testing and simulation-based analysis. The Chipyard environment offers configuration, system validation, and backend chip design. The use of configurable RTL generators enables the creation of heterogeneous systems designed for specific uses.

Merchant et al. [Merchant et al., 2021] present ANDROMEDA, a framework designed for exploring design spaces in manycore systems. ANDROMEDA utilizes RISC-V processing elements (PEs) connected through a NoC, enabling early-stage system exploration and identification of application bottlenecks. The framework allows for experimentation with different configurations to observe performance trade-offs. The performance parameters of ANDROMEDA are verified using benchmarks like STREAM, matrix multiplication, and N-body simulations on the Synopsys HAPS-80D Dual FPGA platform for emulation. Helping designers to determine system configurations for efficient application execution. The baseline model of ANDROMEDA is a 16-node 2D mesh network with additional cores for network tasks. The manycore features a distributed memory model with configurable BRAM, optimizing local node memory access and minimizing external node interference. ANDROMEDA allows users to input system parameters, which automatically creates the desired multicore model. Users can change various elements, like cores, cache sizes, and network connections, among others. The authors claim that this approach enables quick prototyping and performance adjustments, making it suitable for early system development without automatic parameter tuning.

Kamaleldin et al. [Kamaleldin and Göhringer, 2021] describes a reconfigurable platform that can be modified to work with a variety of applications. Its structure allows for run-time reconfiguration to improve how well it performs based on what each application needs. Authors claim that its high platform reusability, regardless of specific applications. The platform uses different PEs that work with different RISC-V configurations, supporting

both 32- and 64-bit processors. These PEs are set up with shared and individual memory, connected to the processor cores, which makes different memory arrangements possible. The designer also may add custom accelerators using the bus interface. The platform capability for run-time reconfiguration is supported by a custom module, called dynamic partial reconfiguration controller designed for changing the PE configurations during operation. Initial evaluations performed on a Xilinx Virtex Ultrascale+ FPGA demonstrate the performance trade-offs and power efficiency that can be realized through varying the number of 32- or 64-bit RISC-V tiles.

Uhlendorf et al. [Uhlendorf et al., 2021] introduce an MPSoC platform to evaluate the performance of a NoC in FPGA contexts. The MPSoC contains 32-bit processors and an MPI library. This environment aids in generating and analyzing NoC traffic. The platform's architecture consists of a supervisor computer and an FPGA-based MPSoC, including a supervisor node, monitoring node, and several PEs, connected via a 2D mesh NoC. Additionally, a bus links the supervisor node and monitoring note to an external shared memory. Each PE features an embedded processor, local memory, network interface, and peripherals. The monitoring node monitors the traffic between PEs and employs a hardware mutex to regulate shared memory access. Communication uses an MPI library, allowing messaging between processors. The authors validate the platform using a Cyclone V FPGA.

2.1.3 Abstract Platforms

Real et al. [Real et al., 2016], introduced MPSoCSim, a simulator using a SystemC-based NoC for connecting heterogeneous MPSoC systems. The simulator provides a comprehensive tool for the modeling and evaluation of complex MPSoCs with an emphasis on clustered architectures. Its simulation structure divides the system into tiles, each containing an Open Virtual Platforms (OVP) processor with dedicated memory for executable code, stack operations, and dynamic storage management linked by a local bus system. These tiles are designed for flexibility, allowing for variation in processor models, and are connected through a common bus that grants access to shared memory used for communication within the cluster. The NoC is parameterizable, allowing the user to configure its switching mechanisms and routing algorithms. Dynamic execution capabilities are incorporated, utilizing compatible OVP processors that can operate different OS to simulate multi-application environments more realistically. These executions are further supported by control features at the processor level, which contribute to application scheduling analysis. The simulation capabilities have been corroborated through comparison with an actual hardware setup, confirming the simulator's capability in precisely representing sophisticated clustered MPSoCs.

Uddin et al. [Uddin, 2017] introduce a high-level simulation framework, called HLSim, for analyzing the performance of a microthreaded manycore architecture. Authors

claim that traditional cycle-accurate simulations for such complex architectures tend to run at a "glacial pace", making the analysis of applications on multi-core systems impractical. The authors address this by proposing a co-simulation environment that can simplify the interactions within the pipeline of microthreaded cores and between various hardware components, thereby creating an efficient design space exploration.

The HLSim is built in C++ and utilizes POSIX threads to execute a discrete-event simulation. It distinguishes itself by decoupling the application and architecture models, allowing independent modifications for performance optimization. The application model performs the execution of microthreaded programs on the host machine while generating events that estimate concurrency and computation workloads. These events are then processed by the architecture model, which advances simulated time based on the cycles required by the events, abstracting pipeline interactions. One special case of HLSim, the One-IPC HLSim, that assumes a simplified execution model where each core executes one instruction per cycle, regardless of the true instruction complexity or pipeline behavior. The authors validate HLSim's effectiveness by comparing its performance against a cycle-accurate simulation, noting that while HLSim greatly increases simulation speed, it comes at the cost of reduced accuracy.

Carlson et al. [Carlson et al., 2011] propose Sniper, which employs a different approach called interval simulation, combining the accuracy of cycle-accurate simulation and the speed of simpler one-IPC (Instruction Per Cycle) models. Interval simulation uses a high-level analytical model based on the concept of intervals—periods between miss events such as cache misses or branch mispredictions. Miss events are determined by simulators of the branch predictor, memory hierarchy, cache coherence, and interconnection network, while the analytical model calculates the timing for each interval. As a result, Sniper can simulate the interactions within a multi-core system more quickly than cycle-accurate models without significantly sacrificing accuracy, according to the Authors. Sniper is built on top of Graphite [Miller et al., 2010], which can perform multi-core processor simulations by running in them in parallel.

The GEM5 simulator, developed by Binkert et al. [Binkert et al., 2011b], is a comprehensive system architecture research framework that integrates the best features of M5 and GEMS [Martin et al., 2005] simulators to provide a highly configurable and flexible simulation tool. It supports various ISAs, including x86 and ARM, and it allows for running unmodified Linux operating systems, thus facilitating the simulation of realistic scenarios. The simulation framework is designed to balance speed and accuracy, with different CPU models and memory system models to cater to diverse research needs. It presents multiple capabilities such as system modes (System-call Emulation and Full-System), detailed models for cache coherence protocols, and flexible interconnection networks for on-chip network studies. Building on the capabilities of GEM5, Cataldo et al. [Cataldo et al., 2018] introduce

Subutai, a hardware/software architecture that optimizes synchronization primitives over a NoC to enhance the performance of parallel applications in MPSoCs.

Zhang et al. [Zhang et al., 2017] present Prophet, a parallel instruction-oriented simulation framework designed to cover the limitations of conventional cycle-oriented many-core system simulators. Based on the GEMS simulator [Martin et al., 2005] and BookSim [Jiang et al., 2013], Prophet utilizes an instruction-oriented model that concentrates on the effects of individual instructions on specific processor components. One of the core advantages of Prophet is its speculative simulation technique, which is used to decouple the simulation of private resources, such as core-specific caches, from that of shared resources, like shared caches. The framework speculatively predicts interactions with shared resources, thereby avoiding the need for constant synchronization and reducing the overhead that limits the cycle-by-cycle simulations. This novel approach only incurs extra computational penalties when the predictions fail to match the actual resource usage, known as mispredictions, requiring corrective recalculations. The authors claim that the Prophet framework is distinguished by its ability to scale efficiently, enabling the simulation of thousands of cores with minimal accuracy loss.

Mack et al. [Mack et al., 2020] present a flexible Linux-based emulation framework designed for rapid pre-silicon evaluation and development of for Domain-specific System-on-Chips (DSSoCs). The framework focus on challenges in DSSoC design encompassing accelerator integration, resource management, and application development. It provides a user-space runtime environment that enables the integration of novel accelerators, the implementation of custom scheduling heuristics, and the development of user applications. The emulation framework comprises three primary components: (*i*) an application handler, (*ii*) a workload manager, and (*iii*) a resource manager. The application handler starts task-graph representations of applications for the emulation. The workload manager maps tasks into the PEs, based on user-selected policies, while the resource manager instantiates test hardware configurations and coordinates task execution. The communication across different PEs is performed through shared memory. The authors also provide a novel compilation toolchain, enabling the automatic mapping of C code to DSSoC platforms.

Hotfilter et al. [Hotfilter et al., 2021] introduce FLECSim, a framework designed to simulate a SoC featuring dedicated accelerators, processor units, and memory components. The framework is flexible and allows the integration of new accelerator models. In FLECSim, accelerators are implemented in SystemC, which enables cycle-accurate execution. The system also includes a RISC-V processor capable of running a Linux operating system which can communicate with the accelerators for computation. FLECSim provides a virtual platform that executes at the instruction-level, using a ISS to simulate the processor. It employs deep neural network (DNN) accelerators described in SystemC. Each PE in the proposed system is a DNN accelerator. The PEs and the RISC-V core are interconnected through a crossbar switch that facilitates communication required for computation offloading

and control. The authors claim that FLECSim's ability to simulate various configurations and gather system metrics enhances its utility for exploring the design space of SoC architectures.

2.1.4 Discussion

In recent years, there have been significant advancements in the design and simulation of manycore systems, particularly focusing on RTL platforms, FPGA prototyping, and abstract simulator frameworks. Each of these strategies has distinct characteristics that address various aspects of manycore system development, from accuracy and complexity to scalability and speed of simulation.

RTL platforms exhibit the highest simulation accuracy compared to abstract platforms, allowing designers to measure system behavior precisely under specific workloads. Elmohr et al. [Elmohr et al., 2018], for example, developed the RVNoC framework that utilizes RISC-V processors and focuses on customizable NoC systems with features that enhance latency and power consumption. Memphis [Ruaro et al., 2019] is another RTL framework with a cycle-accurate hardware model and graphical debugging tools, that use a custom OS. Excluding Memphis, with its MIPS-like processor (Plasma), all other frameworks utilize RISC-V processors, highlighting the researchers' interest in this open-source architecture. These frameworks prioritize precise system measurements under specific workloads and support a range of peripherals. These platforms are unanimous in the use of NoCs as the communication infrastructure, with optimization being a primary objective. However, the prevalent challenge with RTL-based simulation is the extensive computational time required, which limits its practicality over extended periods [Calazans et al., 2003].

FPGA prototyping provides an alternative for validating manycore systems and has seen contributions from several authors, such as [Monemi et al., 2017, Kurth et al., 2018, Amid et al., 2020, Merchant et al., 2021, Kamaleldin and Göhringer, 2021, Uhlendorf et al., 2021]. HERO [Kurth et al., 2018] is particularly interesting with its host processor and PMCA, which supports multitasking by an OS. The frameworks that target FPGAs offer the exploration of user-level applications and often demonstrate the ability to integrate diverse peripherals, like Chipyard [Amid et al., 2020] and ProNoC [Monemi et al., 2017]. Some works, do not allow heterogeneous systems limiting the design-space exploration, that is the case for ANDROMEDA [Merchant et al., 2021] and Uhlendorft et al. [Uhlendorf et al., 2021]. Besides that, FPGA-based approaches suffer from the inherent limitation of computational resources which limits the creation of PEs for building large-scale systems.

On the other end of the spectrum, abstract simulators like Sniper [Carlson et al., 2011] and GEM5 [Binkert et al., 2011b] provide a compromise between accuracy and speed. While they allow some degree of accuracy degradation, they greatly accelerate simulation

speeds, enabling the execution of entire operating systems, complex workload scheduling, and system management. These simulators offer an effective platform for initial design space exploration and testing high-level concepts in manycore systems. For instance, FLECSim [Hotfilter et al., 2021] is designed to integrate new accelerator models and support OS execution. They also can emulate OS behavior at a high level by adding delays that mimic the real OS behavior in target systems [Carlson et al., 2011]. Nevertheless, abstract simulators often offer limited support for peripherals and may not be suitable for precise hardware-software co-verification due to their reduced accuracy.

In this Thesis, we introduce the Chronos-V platform, designed to simulate manycore systems with a *quantum* simulation method that achieves instruction-level accuracy. The system comprises RISC-V PEs interconnected through a 2D mesh NoC and supports peripherals such as hardware accelerators. On the software side, FreeRTOS has been enhanced by integrating a communication API. This API eases task communication via the NoC and incorporates system management support. The purpose of conducting long-term simulations is twofold: firstly, to facilitate software development in the early stages of design; and secondly, to investigate system management techniques. This investigation is conducted by extending FreeRTOS, adding a management task in combination with dedicated hardware accelerators.

2.2 Overview of the Chronos-V Manycore Layers

Figure 2.1 offers a general overview of the Chronos-V manycore layers.

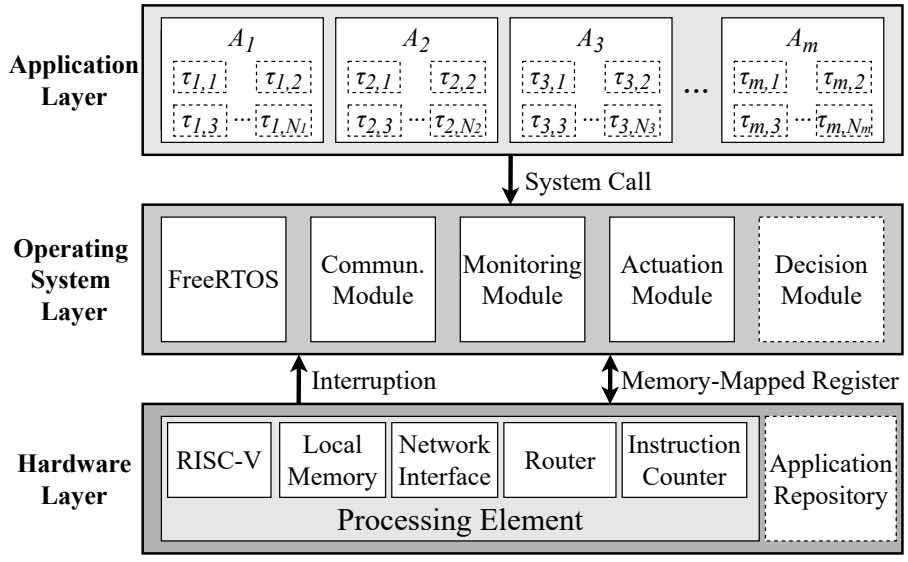


Figure 2.1: The layers of the Chronos-V manycore. Dotted borders indicate centralized modules, while continuous borders delineate modules replicated across the system.

The hardware layer consists of the physical components. The interface between the hardware and OS layers is done through interrupt signals and Memory-Mapped Registers (MMRs). The MCSoC comprises PEs connected via a NoC. Each PE contains a RISC-V processor, local memory, a network interface, a NoC router, and an instruction counter. A unique address identifies each PE's location within the NoC. The Application Repository (App_{rep}) is a peripheral that supplies the system with the required application binary code.

The OS layer employs FreeRTOS, an open-source real-time operating system responsible for managing tasks assigned to PEs. FreeRTOS received four additional modules: Communication, Monitoring, Actuation, and Decision. The Communication Module acts as an API for the Application Layer, facilitating MPI message exchange and communicating with the NoC via drivers. The other modules facilitate ODA management functions.

The application layer contains general-purpose applications, $\mathbb{A} = \{A_1, A_2, ..., A_M\}$. Each application A_m is defined by the tuple $\{D_m, S_m, I_m, P_m\}$, corresponding to the application deadline, start time, number of iterations, and period. Application A_m comprises N_m tasks $\{\tau_{m,1}, \tau_{m,2}, ..., \tau_{m,N_m}\}$. Each task is executed on a PE, with the PE address defined by a mapping heuristic. Applications employ the MPI for communication, accessible to application tasks via system calls that interact with the OS layer.

2.2.1 Hardware Layer

A manycore system can be classified as either heterogeneous or homogeneous, which includes symmetric and asymmetric subtypes. Heterogeneous manycores utilize cores with different architectures and organizations, such as general-purpose processors, graphics processing units, and dedicated hardware accelerators [Esmaeilzadeh et al., 2012]. Symmetric homogeneous manycores comprise systems with all cores having identical architecture and organization. Asymmetric manycores represent a subtype of homogeneous systems where the cores share the same ISA but differ in their organization [Hill and Marty, 2008].

Chronos-V is a heterogeneous manycore, with two regions. The first is a symmetric region called General Purpose Processing Elements (GPPE), featuring processors with the same architecture and organization dedicated to running general-purpose applications. The second region encompasses peripherals. Figure 2.2(a) illustrates the manycore architecture, highlighting the GPPE and peripheral regions.

Figure 2.2(b) details on the PE architecture. The PE comprises five main modules: (*i*) an RV32IM processor, a 32-bit RISC-V with an integer multiply/divide extension; (*ii*) local scratchpad memory; (*iii*) a NoC router; (*iv*) the instruction counter, which classifies and counts executed instructions, this data is available through MMRs; (*v*) and a network interface (NI) with direct memory access capability, which enables the processor to offload

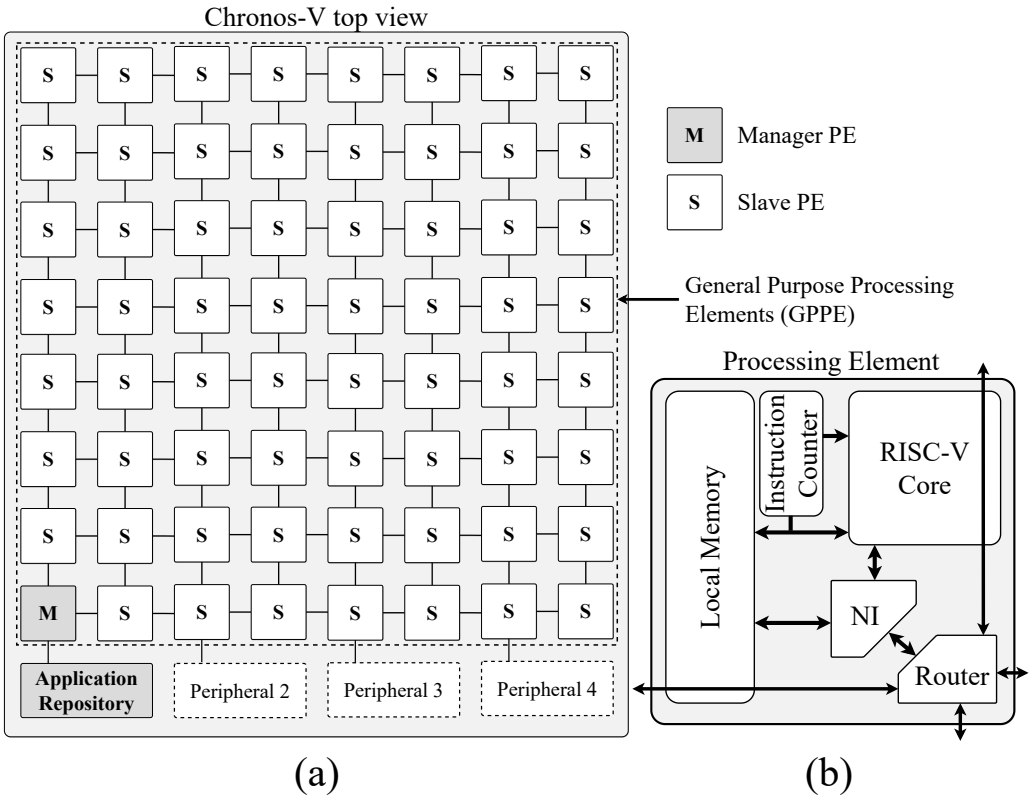


Figure 2.2: Chronos-V manycore architecture: (*a*) General Purpose Processing Elements (GPPE) and Peripherals; (*b*) PE architecture.

data transfers between the NoC and local memory. An API at the OS level configures the NI for packet reception and transmission, thus minimizing the processor's overhead for packet handling.

The dual-port local memory holds code and data, and its selection aimed to simplify system complexity and reduce power consumption by eliminating the need for cache controllers and the associated NoC traffic. The NoC router features wormhole packet switching, 2D-mesh topology, XY routing, round-robin arbitration, input buffering, and credit-based flow control.

The instruction counter module categorizes and counts the executed instructions to enable power and temperature estimations, as will be discussed in Section 3.2.2 and 3.2.3. This module is included in the system to support monitoring at the PE level when specific management strategies that require it are employed. Periodically, the OS retrieves the monitored information.

Peripherals provide specialized services for the system and applications, such as hardware accelerators. The system must include at least one peripheral: the App_{rep}. The App_{rep} informs the manager PE (MPE) when new applications must execute in the GPPE area. The MPE, which runs the same OS as the other PEs and performs decision-making functions like application mapping and task migration heuristics, decides each application mapping and then instructs the App_{rep} the address to transmit each task's binary code.

Subsequently, the App_{rep} transfers the tasks' binary code to the mapped PEs in the GPPE area. We also have implemented a peripheral for temperature estimation called *TEA* (Temperature Estimation Accelerator), initially developed at the RTL level by Da Silva et al. [Silva et al., 2019], to facilitate fine-grain thermal management.

2.2.2 Operating System Layer

Chronos-V adopts FreeRTOS [FreeRTOS, 2022], which allows tasks organized as a collection of independent threads. We have extended FreeRTOS with four system modules:

- **Communication Module** – enables communication among tasks and modules mapped to different PEs;
- **Monitoring**, **Decision**, and **Actuation Modules** – manage the system using the ODA control method [Hoffmann et al., 2013].

FreeRTOS schedules general-purpose tasks and the system modules that execute on the same PE. General-purpose tasks receive a lower scheduling priority compared to system modules. Thus, each PE runs its instance of FreeRTOS, managing only the PE on which it executes. The proposed architecture aims to be generic and modular at the OS layer, providing designers with the flexibility to conduct experiments related to system management by enabling easy replacement of system modules to suit their needs.

The **Communication Module** operates as follows:

- Upon receiving a packet, the module reads the packet header to identify its *service* (the action the packet requires) and provides the appropriate address to write the packet payload. A packet may contain user-level services, such as a request for data or data delivery, or OS-level services, such as task start/stop commands, task allocation, notification of task completion, and changes in the PE operating frequency, among others.
- This module configures the MMRs and signals the NI to inject the packet into the NoC. It allocates memory in the OS space to create the packet (including header, size, and payload) and, after the NI injects the packet into the NoC, signals that the NI can release the allocated memory to make space for new packets.

Figure 2.3 illustrates the communication from a *producer* PE to a *consumer* PE. When the producer PE is ready to send a packet, the `SendRaw()` function configures the NI to inject the packet into the NoC. The NI in the consumer PE interrupts the processor upon packet header reception. The OS then configures the memory address to write the incoming packet. Once written to memory, the packet becomes available to the task that is waiting for its reception.

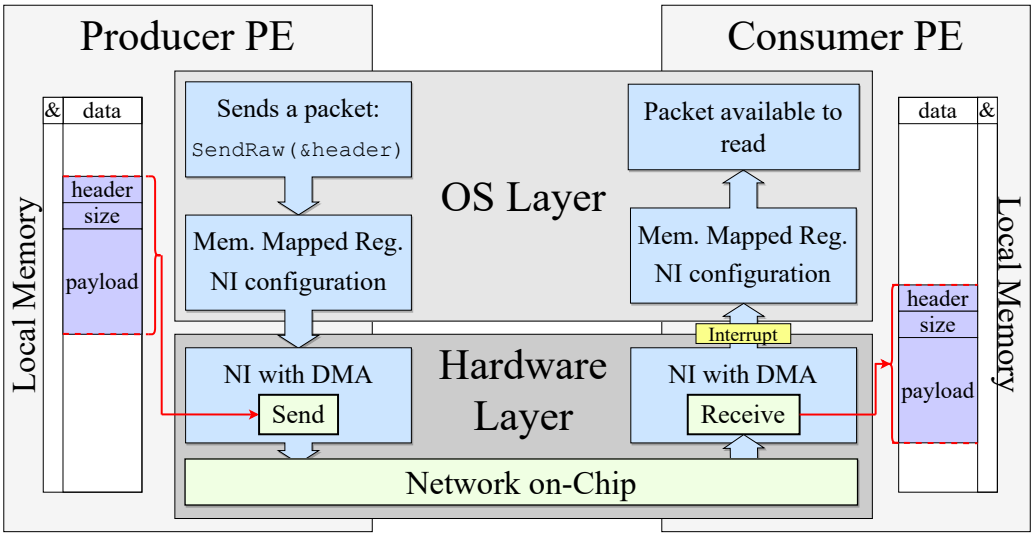


Figure 2.3: Communication from a producer PE to a consumer PE (OS view).

The **Monitoring Module** activates periodically to gather local data that informs the Decision Module. We define the monitoring window period at design time. Current implementation of Chronos-V feature two monitoring sources:

The *Instruction Counter Module* classifies and counts the number of instructions a PE executes during a monitoring window. We categorize instructions into seven groups (register, immediate, branch, load, store, jump, and PC), providing a distinct counter for each. This module also tracks the number of memory accesses. Embedded in the router, the *Router Counter Module* tracks the number of flits passing through during a monitoring window. Using this data, each PE periodically estimates its dynamic energy consumption, further explained in Section 3.2.3.

The PEs periodically send their estimated energy consumption to *TEA*. After estimating temperature, detailed at Section 3.2.2, *TEA* sends each PE temperature data to the MPE, granting fine-grained temperature monitoring.

The **Decision Module**, running in the MPE, receives periodic raw monitoring data from every PE. The management heuristic utilizes this data to make decisions in line with design objectives. For instance, if the module observes a temperature violation or a rising temperature trend in a PE, it may command the Actuation Module in that PE to migrate a task or adjust the operating frequency. Application mapping also falls within the Decision Module competences.

The **Actuation Module**, executing on each PE, including the MPE, performs actions as defined by the Decision Module. This includes task allocation, task migration, and DVFS.

Upon receiving a task allocation packet, the Actuation Module allocates memory for the incoming task, communicating the starting memory address to the NI for writing the

task's object code sent by App$_{rep}$. Following the code reception, the OS schedules the task for execution on the PE.

Figure 2.4 illustrates the task migration protocol. The Decision Module sends a `migration request` packet (event 1) to the source PE (2). The source PE continues running the task until it reaches a migration-safe state (3). The OS then stalls the task and transmits a `migration acknowledge` (4) packet to the MPE. The MPE stalls all other tasks from the same application (5-6) to prevent communications with the migrating task. Subsequently, the MPE directs the source PE to transfer the task's code and data to the target PE (7). Once received by the target PE (8), the OS allocates the task, and the PE notifies the MPE of the successful migration with a `forward complete` packet (9). Finally, the application resumes through a `task resume` packet sent to every application task. This mechanism introduces overhead to the application execution, as task suspension is necessary to preserve data integrity during migration.

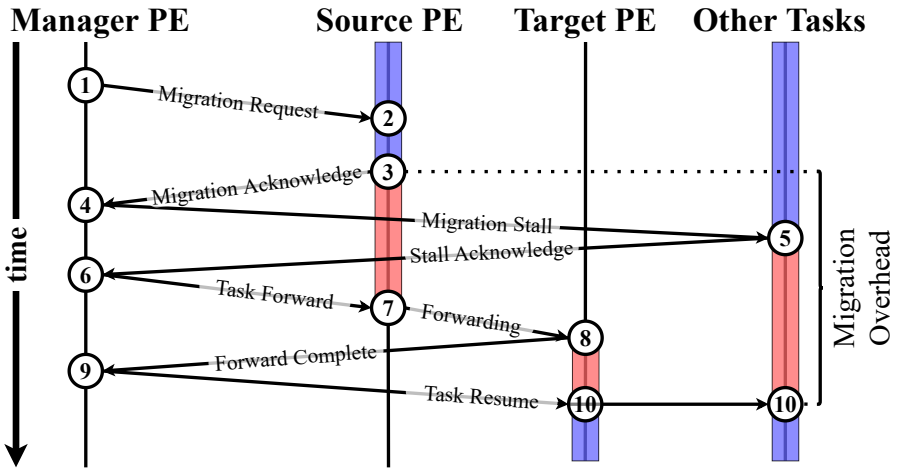


Figure 2.4: Events diagram of the task migration protocol implemented in the Chronos-V manycore.

DVFS is another actuation example supported by the platform. Modeled at an abstract level, frequency, and voltage changes are simulated by modifying the simulation *quantum* (explained in Section 2.3).

2.2.3 Application Layer

This layer corresponds to general-purpose applications. Communication Task Graph, $CTG(T,E)$, models each application. Each vertex $\tau_i \in T$ represents a task, and each edge $e_{ij} \in E$ represents communication from τ_i to τ_j. Assuming that edges e_{ij} are modeled implicitly in τ_i, an application is represented as a set of tasks $\{\tau_1, \tau_2, ..., \tau_{|T|}\}$.

Those applications must be position-independent executables to allow multitasking without virtual memory management support. The applications stored outside the MCSoC

computing fabric are deployed into the system through the App_{rep}. The App_{rep} communicates with the MPE, which executes the task mapping protocol. After transmitting all tasks to the selected PEs, the MPE releases the application execution. System calls make the interface between tasks and the OS layer, allowing, for example, inter-task communication.

Figure 2.5 presents the message passing protocol. Each message exchange requires two packets:

- `message request`, sent by the consumer task to the producer task, informing that the task is ready to receive a message. After sending this packet, the consumer task remains blocked, waiting for the message reception.
- `message delivery`, sent by the producer task after receiving the `message request`, contains the message payload. This packet is sent immediately after the `message request` if the message is already in the packet queue (Figure 2.5(a)). Otherwise, the OS registers the request (pending request), transmitting the message once the producer task generates it (Figure 2.5(b)).

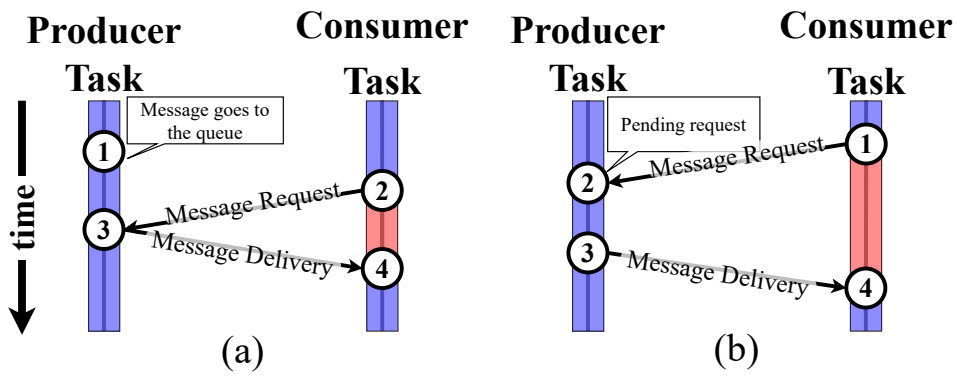


Figure 2.5: Event diagram of the message passing protocol implemented in the Chronos-V manycore.

When a task finishes its execution, the OS blocks its scheduling and waits for the consumption of packets stored in the packet queue. Next, the OS notifies the MPE and releases the task memory. An application is considered finished by the MPE when it receives the “concluded” packet from all tasks of the given application. This notification is important to release data structures used by the decision module.

2.3 Simulation Model

We utilize the Open Virtual Platform (OVP) [Imperas, 2021] to create and simulate the platform. We selected OVP for its scalability and flexibility. Scalability allows for the simulation of large systems (e.g., 20x20 PEs) within a reasonable time frame (on the order of hours), while flexibility simplifies the modeling of peripherals (e.g., hardware accelerators) or the swapping of processors. OVP includes over 70 processor models, each supported by

a corresponding toolchain. The Chronos-V model preserves the temporal and spatial traffic distributions when compared to the physical implementation [Lopes et al., 2021].

The simulation of the platform begins with a command line that includes eight parameters:

1. The simulation name, which facilitates the creation of a unique folder for the platform configuration that can be reused for various applications;
2. The number of PEs in the X-coordinate;
3. The number of PEs in the Y-coordinate;
4. The simulation time, indicating the time at which the simulation must stop;
5. The management algorithm, if any;
6. The migration algorithm, if applicable;
7. The clustering algorithm, if applicable;
8. The scenario file with the set of applications to be executed. For each application, it is possible to specify its start time, periodicity, and execution frequency.

Based on these parameters, the script configures the platform source code and hardware description and creates a self-contained folder. Subsequently, the system components are compiled. We can classify these components into two groups: (*i*) system components, such as the NoC router, the App$_{rep}$, and the NI, which emulates the behavior of their physical counterparts; and (*ii*) virtual components that facilitate system simulation, like the iterator (discussed later).

The script generates a `.tcl` file encompassing instances of system components and their interconnections, thus describing the system model. In this file, each processor connects to every component comprising a PE. The NoC routers link to adjacent routers, and the peripherals connect to the NoC edges. The *iGen* tool [Imperas, 2021] then processes the `.tcl` file, creating a *.c* file that contains the description of the system interconnections using the Open Platform (OP) API. Finally, it is compiled and linked to the system components and to the instruction set simulator (ISS) to generate the actual system model.

On top of that, there is the simulation manager, named *harness* [Imperas, 2021], responsible for instantiating the system model and providing stimulus to it (it acts as a testbench). OVP adopts a *quantum*-based simulation paradigm with instruction accuracy. This means that the simulation time is divided into pieces, called *quanta*. The simulation engine makes each processor execute the necessary instructions so that the processor advances the time by a *quantum*. After that, synchronization occurs between the processors and the simulation time is incremented by a *quantum*. Definition 1 defines the *quantum* parameter.

Definition 1. Quantum — the period that each processor executes. After the completion of all processors' *quantum* periods, the simulation advances the current time by one *quantum*, restarting the sequence with the first processor.

To compute the *quantum*, we assume that the number of cycles per instruction (CPI) is one. Equation 2.1 presents the computation of the *quantum* value.

$$quantum = \frac{Inst_Q}{Core_{IPS}} \tag{2.1}$$

where: $Inst_Q$: number of instructions executed during the *quantum* period; $Core_{IPS}$: processor frequency, as the CPI is set to one.

Processor communication occurs through synchronization after every processor has completed its *quantum*. To minimize data waiting time, $Inst_Q$ must be optimized to balance the trade-off between communication and computation. A small $Inst_Q$ slows the simulation due to frequent synchronizations, while large $Inst_Q$ results in processors stalling for data.

The number of instructions executed by one processor within the *quantum* dictates the processor frequency. When all PEs have the same $Inst_Q$, the system operates at a homogeneous frequency. Therefore, changing the number of $Inst_Q$ of a given PE allows to adjust its frequency. For instance, for a processor with $Core_{IPS}$=1GHz and $Inst_Q$=10,000, the *quantum* equals $1e^{-5}$ seconds. Reducing a processor's frequency to 500MHz requires a $Inst_Q$ equal to 5,000. This method simulates the effect of Dynamic Voltage and Frequency Scaling (DVFS).

Algorithm 1 shows the simulation loop pseudo-code as defined in the *harness*. The loop initiates by enabling each processor (P_{id}) to simulate a *quantum* period (lines 4 to 6). The `simulate()` function returns *True* on completion of a processor's execution and *False* otherwise. The processes can be parallelized, and the `join()` (line 7) waits for the completion of all processes. Note that within the *quantum* execution, processors can only access data in their local memory, including packets that the NI has received.

The *iterator* is an abstract system component that is essential in synchronizing routers after all processors have executed. The *iterator* behavior is expressed in the pseudo-code at lines 8 to 13. The process involves repeated calls to the `iterate()` function for each router (R_{id}), sequentially progressing through the iterations. Each iteration allows a router to send one flit forward to the next router in the path if it is available. The iterated loop concludes when flits can no longer proceed due to traffic or unavailability for transmission.

The NI and the NoC router are modeled in C and rely on the "iteration" to properly perform their behavior. When the router receives an "iteration" it evaluates if there is any flit available within input buffers. If available, the router selects a buffer according to a round-robin arbitration policy. The router executes the routing algorithm to find the destination port for the selected buffer. If the destination port is available, the router connects the input buffer to the output port. After performing the routing process, the router sends ahead one flit of each buffer that is connected to some output port (flits may be blocked due to NoC

Algorithm 1: Simulation loop pseudo-code

Input: *model*, $Inst_Q[N]$, *quantum*

```
1  SimulationTime ← 0
2  iteration ← 0
3  repeat
4      foreach P_id in model do
5          Finished_id = simulate(P_id, Inst_Q[id]).start())
6      end
7      join()
8      repeat
9          foreach R_id in model do
10             Syncronized_id = iterate(R_id, iteration)
11         end
12         iteration++
13     until Syncronized_id = True ∀ id
14     SimulationTime += quantum
15 until Finished_id = True ∀ id
```

congestion). Those sent flits are stored in the neighbor input buffers or the NI internal buffer if the packet destination is the local PE.

The module *iterator* is responsible for controlling the flit transmission, evaluating routers sequentially. An "iteration" enables flits to advance one hop. To avoid flits traversing more than one hop, when the flit is sent ahead, it carries the current "iteration". A new "iteration" starts after evaluating the entire NoC. This process repeats up to $Inst_Q$ times. The transmission stops when every router does not have flits to transmit or flits or is blocked for congestion reasons.

Figure 2.6 presents the NI and router interface with the processor, memory, and iterator. The communication API provides functions to send and receive packets using this infrastructure. The OS informs the packet address to be sent or received to the NI by using memory-mapped registers and releases the processor during the communication with the NoC. The following ports access these registers:

- *address*: at the system startup, the OS informs the address to store incoming packet headers to the NI. During the system execution, the OS uses this register to inform the NI about the address of a packet that has to be transmitted to another PE;
- *statusTX*: this register stores the transmission module status. If the read value is zero, the processor may start a packet transmission;
- *statusRX*: this register is used by the OS to notify the NI that a packet reception is complete, so a new packet can start to be received.

The NI rises interrupt signals to notify a packet transmission (*intTX*) or upon a packet reception (*intRX*). The transmission interrupt handler removes the transmitted packet from the local memory, opening space for new packets. This handler also checks if another

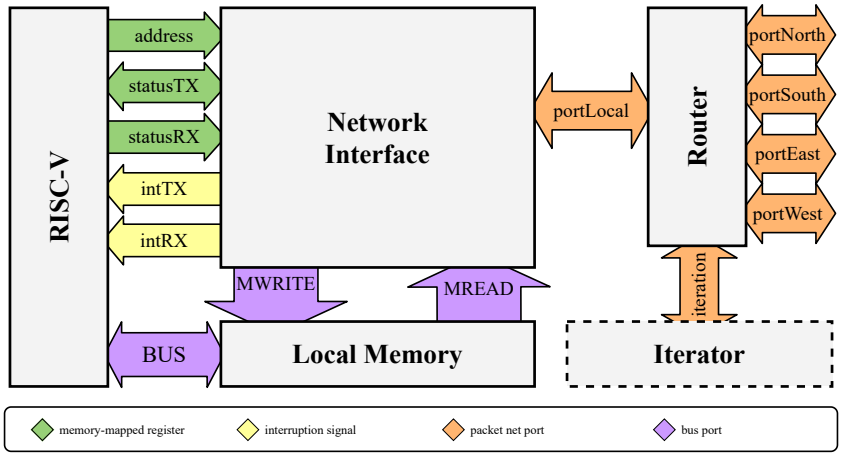


Figure 2.6: Iterator and internal PE components interface. The Iterator has one "iteration" connection to each router in the system.

packet is queued, configuring the NI to transmit it. The reception interrupt handler is responsible for reading the packet header and then identifying the packet service to perform the required action before returning to the application.

Figure 2.7 exemplifies a 3x3 system with three packets being sent during the same *quantum*: $R_6 \rightarrow R_7$; $R_0 \rightarrow R_7$; $R_2 \rightarrow R_4$. Each router is "iterated" sequentially, and, in the first turn, the first flit of each packet advances one router (Figure 2.7(a), continuous lines). In the next turn, R_1 selects one of the two available packets to route, following the round-robin policy, and forwards the flit from R_0 while blocking the flow from R_2. Also, NI_7 receives the packet from R_6 and blocks the R_7 local port. Thus, in the next turn the flow $R_0 \rightarrow R_7$ stops at the R_7 south buffer, and flow $R_2 \rightarrow R_4$ stays blocked at the R_1 east port (Figure 2.7(b)). At this point, no flit can be forwarded, and the simulation *quantum* ends by increasing the simulated time by one *quantum* (line 14 of Algorithm 1). In the next *quantum*, NI_7 interrupts the processor to get a memory address to save the incoming packet, releasing the R_7 local port.

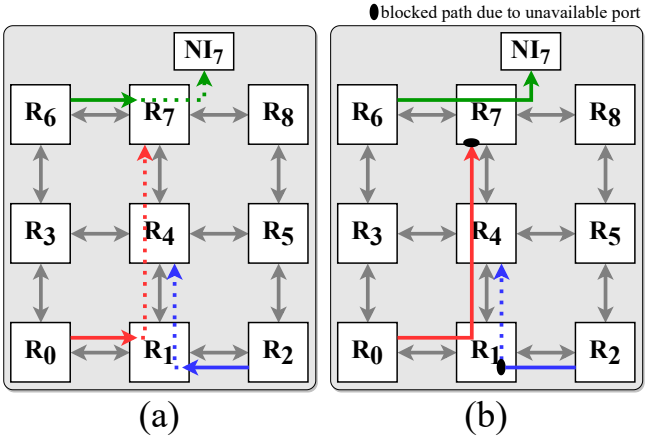


Figure 2.7: Example of packets behavior after being sent in the same *quantum*.

Harness also provides assessing functions using the OP API. For example, the "instruction counter" module is a monitor that generates a callback in the *harness* every time the processor fetches a new instruction from memory. Inside the callback, the instruction is identified and categorized. After that, it updates the instruction counter amount in the PE memory, which is accessed periodically by the monitoring module to estimate the processor energy consumption.

2.4 Results - Simulation Effort

In this section, we evaluate the simulation speedup by comparing the simulated performance of the proposed abstract model against that of an RTL simulation. We performed all simulations on a workstation equipped with an Xeon E-2246G@3.6 GHz processor (12 cores), 16GB DDR4 2666MHz dual-channel DRAM, and running Ubuntu 20.04.

How quickly can the manycore simulation run using an abstract platform model? We compare the Chronos-V platform, running FreeRTOS, with a similar platform modeled at the RTL level (SystemC) called *Memphis* [Ruaro et al., 2019], which runs an in-house operating system on MIPS-like processors. Despite this, both platforms share the same NoC, peripherals, and memory organization. This experiment measures the simulation effort, defined as the time (in *minutes*) required to simulate one second of the system model.

The experiment encompasses (*i*) 9 system sizes ranging from 4 to 100 PEs; (*ii*) execution times from 100*ms* to 1*s* across 10 scenarios; and (*iii*) a system load of 50%, meaning a 64-PE system executes 32 tasks, with one task per PE. We evaluated the Chronos-V platform across all 90 scenarios, while *Memphis* underwent evaluation in only 46 scenarios due to longer simulation times. We excluded migration or management algorithms from these experiments to focus solely on determining the simulation speedup of the Chronos-V platform in comparison to the RTL model.

Each simulated scenario features various applications, including realistic benchmarks such as MPEG encoding and decoding, pattern recognition, and Dijkstra's shortest path algorithm, besides three synthetic applications categorized by communication-intensive, computation-intensive, and balanced workloads.

Figure 2.8 shows the average simulation effort for executing the proposed scenarios. The Chronos-V simulation ranged from 2.36 $\frac{min}{s}$ to 92.46 $\frac{min}{s}$ for system sizes between 4 and 100 PEs, respectively. Memphis demanded between 147.9 $\frac{min}{s}$ and 3,731.39 $\frac{min}{s}$ for the same system sizes. The speedup diminishes with increasing system size due to the *quantum*-based simulation's requirement of synchronizing all processors at the end of each period (line 7 of Algorithm 1), a process that uses the 12 cores of the host machine in a multi-threaded simulation. To give a concrete example, OVP took 1 hour and 30 minutes, while

the RTL model took 62 hours and 13 minutes to simulate a system with 100 PEs running 50 tasks simultaneously.

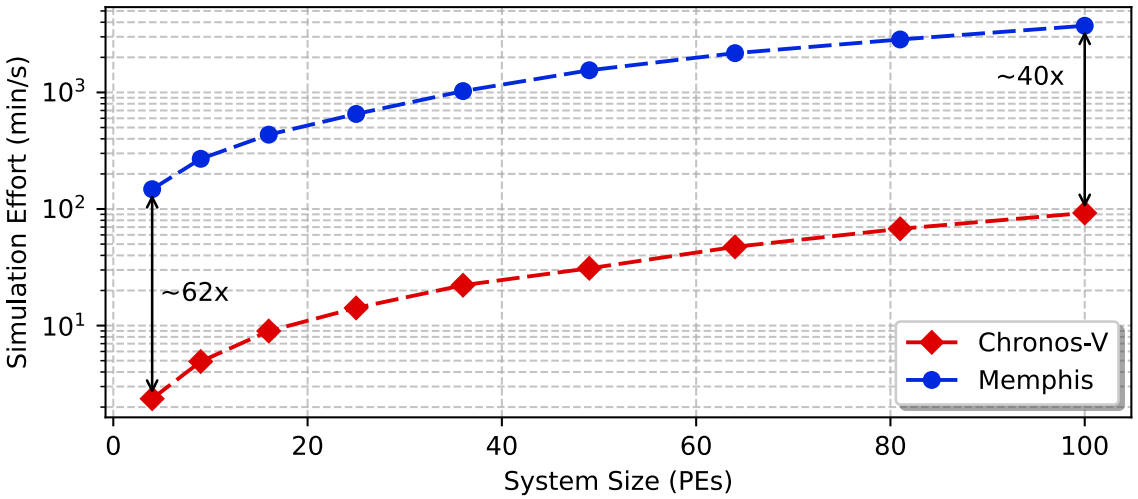


Figure 2.8: Chronos-V and Memphis simulation effort.

It is essential to recognize that these platforms serve different purposes. While RTL simulation is cycle-accurate and is primarily used to validate manycore hardware, abstract model simulation is intended for software development validation at both user and management levels. Chronos-V's unique characteristic is that it provides designers with high simulation speeds and NoC behavior closely resembling the RTL model, including path choices and congestion effects. This similarity allows designers to evaluate different scenarios, such as: (*i*) identifying hotspots to improve task mapping; (*ii*) studying security-related techniques, like detecting traffic anomalies that could indicate attacks; and (*iii*) exploring thermal management techniques.

2.5 Conclusion

This Chapter presented the Chronos-V platform, a high-level NoC-based manycore model. Several works in the literature present abstract models for manycore platforms, seeking hardware and software design space exploration at early design stages with reduced simulation time. Our work also meets these goals, *standing out* by the modeling that preserves the NoC behavior (routing and congestion) and by including management APIs loosely coupled to the kernel. The proposed platform demonstrates up to 62 times faster simulation time when compared to the RTL platform while still preserving the routing and congestion characteristics of communication based on NoC. Therefore, the proposed high-level model may help designers leverage the research in the field of NoC-based manycore systems, including software and management techniques.

3. BACKGROUND KNOWLEDGE

As introduced in Chapter 1, the enhancement of single-core performance encountered limitations due to the constraints in exploiting instruction-level parallelism and the penalties associated with increasing core frequency. To maintain the trajectory of performance improvement, designers moved towards a multi-core and, subsequently, a manycore paradigm. This evolution was facilitated by the ongoing reduction in transistor gate length, as highlighted by Moore [Moore, 1998].

Reducing the size of transistors diminishes their capacitance, enabling faster switching at a lower power cost. Additionally, the miniaturization of transistor dimensions allows their operation at reduced voltages, further decreasing power consumption. Dennard [Dennard et al., 1999] from IBM observed that both the power dissipation and area of transistors scale at a consistent ratio. This observation implied that newer generations of integrated circuit technology could keep a constant power density, defined as the power dissipated per unit area. However, in practical applications, the power supply voltage does not scale proportionally with device dimensions, diverging from the Dennard's "law". This discrepancy arises from various system-related constraints, such as hot electron effects and short channel effects, as discussed by Streetman [Streetman et al., 2016].

This Chapter is structured as follows.

1. Section 3.1 introduces the concept of Dark Silicon.
2. Section 3.2 discusses the need to estimate the temperature at runtime to ensure safe operating conditions of the system. This section is divided into three parts: Section 3.2.1 focuses on the use of sensors for temperature measurement; Section 3.2.2 presents the mathematical methods for temperature estimation; and Section 3.2.3 provides an overview of the energy estimation methodology, which is essential for the accurate computation of the temperature at runtime.
3. Section 3.3 explores reliability issues related to integrated circuits. Section 3.3.1 discusses the major effects that influence reliability. These effects encompass various failure mechanisms that can potentially compromise the integrity and performance of integrated circuits. Section 3.3.2 introduces the sum-of-failure-rates (SOFR) model to evaluate the overall system reliability.

3.1 Dark Silicon

The breakdown of the Dennard law has led to an increase in power density. A higher power density means more heat is generated within the same chip area, increasing

the chip's temperature. Exceeding a certain temperature threshold can cause unreliable operation, accelerated degradation, and potentially permanent damage to the chip, as noted by Kong et al. [Kong et al., 2012]. To guarantee safe operation without substantial alterations to the cooling system, it is imperative to keep the chip's temperature below a maximum safe limit. For this purpose, a thermal design power (TDP) budget is established, often conservatively estimated, potentially leading to underutilization of the chip's capabilities [Rahmani et al., 2016].

A portion of the chip must remain idle to maintain power dissipation within the TDP limit; this inactive part is known as dark silicon [Esmaeilzadeh et al., 2011]. While dark silicon limits the concurrent utilization of chip resources, it offers avenues for implementing temperature management strategies and extending the chip's lifetime [Gnad et al., 2015]. One strategy involves interleaving non-operational (dark) cores with active cores during task allocation, which helps in reducing power density and subsequently lowering chip temperature [Rathore et al., 2018].

3.2 Temperature

The industry provides a TDP for integrated circuits to provide a safe operating power level for systems. Cooling solutions are designed to dissipate heat up to this TDP value, ensuring that the heat sink is appropriately sized and operating at the TDP does not result in thermal problems. However, it is important to note that the TDP does not represent the maximum achievable power. Instead, this power budget is an abstraction that allows system designers to manage thermal violations indirectly [Rahmani et al., 2016].

In contrast to the static application of TDP, deploying Dynamic Thermal Management (DTM) techniques offers a more adaptive approach. DTM interacts with the system using various actuation mechanisms, such as clock gating, voltage and frequency scaling, and enhancing fan speed. This strategy requires direct temperature monitoring; if the system's temperature exceeds a specific threshold, DTM activates to mitigate the temperature increase [Jantsch et al., 2017].

3.2.1 Sensing

DTM techniques need temperature data, typically obtained through physical sensors integrated into the chip. This integration approach for temperature sensors in many-core systems is well-established and documented [Spiliopoulos et al., 2011, Benini et al., 2012]. However, each additional temperature sensor substantially influences the area of the chip and its power consumption [Ranieri et al., 2015]. Consequently, the deployment

of fine-grained physical sensing may pose a challenge in large systems encompassing over a hundred PEs, as such systems would require a significant number of sensors, leading to area and power demands scaling [Ditzel et al., 2021].

3.2.2 Temperature Estimation

An alternative to physical sensors for temperature measurement is using a thermal model. Numerous efforts in full-chip and compact thermal modeling for microelectronic systems have been made [Wang and Chen, 2002, Su et al., 2003, Li et al., 2004, Cheng et al., 1998]. However, these methods often oversimplify the thermal package modeling. Additionally, they are not entirely suitable for runtime estimation due to the significant computational effort they require. HotSpot [Huang et al., 2006] is a widely used tool that generates thermal models, addressing package modeling more accurately and offering computational efficiency. It is a methodology for creating Compact Thermal Models (CTMs) based on the prevalent stacked-layer packaging approach in modern Very Large-scale Integration (VLSI) systems.

CTMs enable relatively precise temperature predictions with minimal computational effort at various abstraction levels [Sabry, 2003]. These models draw on the analogy between thermal and electrical phenomena [Bergman and Lavine, 2017], where heat flow through a thermal resistance resembles electrical current flow, and the temperature difference is analogous to voltage. The thermal capacitance, dependent on the material's specific heat and heat absorption capacity, is analogous to electrical capacitance. A typical packaging scheme in modern VLSI systems involves multiple stacked layers of different materials, as illustrated in Figure 3.1. HotSpot's thermal models conform to this packaging structure.

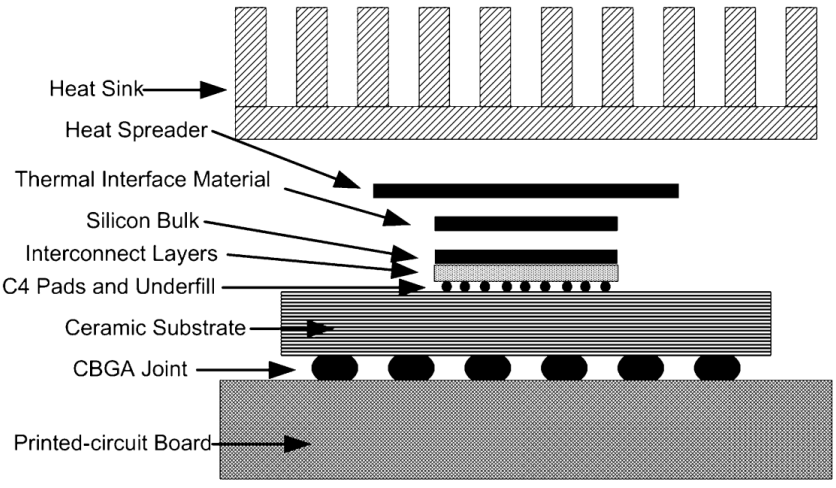


Figure 3.1: Stacked layers in a typical Ceramic Ball Grid Array (CBGA) [Parry et al., 1998].

To develop a CTM, HotSpot first identifies the different layers, their positions, and adjacencies. Each layer is segmented into blocks, and a thermal resistance (R) is applied between the center of a block (a node) and each neighboring edge to model heat transfer between nodes. The silicon substrate layer, for instance, can be divided according to architectural-level units or into regular grid cells, depending on the die-level design requirements. Larger layers, such as the heat sink, are partitioned as shown in Figure 3.2(a), with the central part divided similarly to the die and additional resistances to model heat transfer from the die's border to the material and from the material's border to the medium. In addition to lateral heat transfer ($R_{lateral}$), each block also has vertical heat transfer to the next layer modeled by a vertical resistance ($R_{vertical}$), as depicted in Figure 3.2(b). The thermal resistances and capacitances are calculated based on block geometry and material properties like specific heat and thermal conductance.

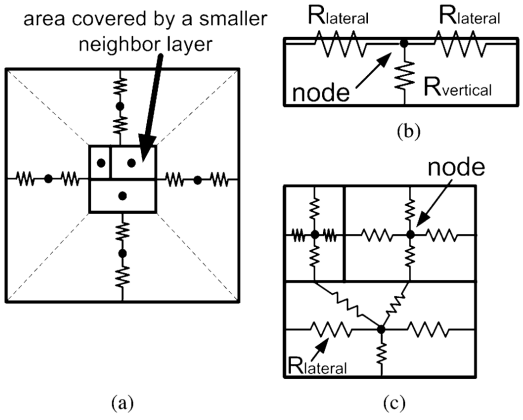


Figure 3.2: (a) Partitioning of large-area layers (top view). (b) One block with its lateral and vertical thermal resistances (side view). (c) A layer, such as the silicon die, can be divided into an arbitrary number of blocks if detailed thermal information is required (top view) [Huang et al., 2006].

For transient temperature calculations, HotSpot employs a fourth-order Runge-Kutta numerical method with an adaptive iteration count, providing quicker results than other thermal simulations and suitability for preliminary VLSI design estimations. However, the HotSpot algorithm does not offer real-time temperature estimation with the necessary time resolution for DTM use [da Silva, 2021].

MatEx [Pagani et al., 2015a], similar to HotSpot, uses a thermal model based on resistor-capacitor (RC) thermal networks to estimate chip temperatures based on power consumption. Unlike HotSpot, MatEx uses matrix exponentials and linear algebra to solve the first-order RC differential equations, providing a fast and accurate method for computing transient temperature peaks for runtime use. MatEx estimates transient and peak temperatures following changes in the system's power state, allowing for variable durations of each power state. It models the system's thermal behavior using Equation 3.1.

$$CT' + KT = P + T_{amb}K_{amb} \tag{3.1}$$

where:

- $C = [c_{i,j}]_{NxN}$ is the thermal capacitance between nodes i and j;
- $T' = [T'_i(t)]_{Nx1}$ denotes the first-order derivative of temperature at time t for each thermal node;
- $K = [k_{i,j}]_{NxN}$ is the thermal conductances between nodes i and j;
- $T = [T_i(t)]_{Nx1}$ is the temperature at time t for each thermal node;
- $P = [P_i(t)]_{Nx1}$ is the power consumption at time t by each thermal node;
- T_{amb} is the ambient temperature;
- $K_{amb} = [k_i]_{Nx1}$ is the thermal conductance between each thermal node and the ambient.

In the thermal model employed by MatEx, the system is represented by a network comprising N thermal nodes. These nodes are interconnected through thermal conductances (K) and thermal capacitances (C). The ambient temperature (T_{amb}) is assumed to be constant, and the power consumption (P) at each node represents the system's heat source. MatEx generates the capacitance and conductance matrices (C and K) based on the physical characteristics of the chip, its packaging, and the cooling solution. The characteristics adopted by this work are detailed in Table 3.1.

Table 3.1: Thermal characteristics adopted by this work.

Element	Characteristic	Value	Element	Characteristic	Value
Chip	Thickness (m)	160×10^{-6}	Heat Spreader	Side (m)	30×10^{-3}
	Conductivity (W/mK)	100		Thickness (m)	1×10^{-3}
	Specific Heat (J/m^3K)	1.75×10^6		Conductivity (W/mK)	400
Heat Sink	Convection Capacitance (J/K)	140.4		Specific Heat (J/m^3K)	3.55×10^6
	Convection Resistance (K/W)	0.1	Interface Material	Thickness (m)	20×10^{-6}
	Side (m)	60×10^{-3}		Conductivity (W/mK)	4
	Thickness (m)	6.9×10^{-3}		Specific Heat (J/m^3K)	4×10^6
	Conductivity (W/mK)	400	Temperatures	Ambient Temperature (K)	318.15
	Specific Heat (J/m^3K)	3.55×10^6		Initial Temperature (K)	318.15

The MatEx tool uses linear algebra to solve matrix exponentials using eigenvalues and eigenvectors. This process is computationally intensive, with a complexity of $O(n^3)$, but it requires execution only once for a given floorplan. In the context of the system proposed in Chapter 2, which has not been synthesized, there is an absence of floorplan data to characterize the system accurately. We conducted a literature review to address this issue, selecting the Celerity NoC-based manycore system [Rovinski et al., 2019] as a reference.

Celerity's tiles, encompassing a total area of 15.25 mm^2 and consisting of 496 tiles each with a 5-stage, in-order RV32IM core and 4 *KB* of memory, provided a basis for calculating the area per tile, which was found to be approximately 0.0307 mm^2, with memory representing $\sim$55% of this area. For this work, the tile area was considered as 0.070756 mm^2 which corresponds to an increment in the memory area of 3.4$\times$ when compared to Celerity's tile.

The hardware accelerator embedded in our system is named Temperature Estimator Accelerator (TEA) and was proposed by Silva et al. [Silva et al., 2021]. It was developed based on MatEx and is suitable for temperature estimation in complex manycore devices.

The standard MatEx algorithm calculates the steady-state temperature (T_{steady}) for a given power consumption using Equation 3.1. When the steady-state temperature is reached, the first derivative of temperature becomes zero. Consequently, the steady-state temperature is solely a function of conductance, as shown in Equation 3.2.

$$T_{steady} = K^{-1}P + K^{-1}T_{amb}K_{amb} \tag{3.2}$$

Once the steady-state temperature has been computed, it is possible to calculate the transient temperature (T) at any given time t using Equation 3.3.

$$T(t) = T_{steady} + e^{St}\left(T_{init} - T_{steady}\right) \tag{3.3}$$

where e^{St} represents the matrix exponential of S at the time interval t, and the matrix S is defined as:

$$S = -C^{-1}K \tag{3.4}$$

However, for runtime temperature estimation, we only require the temperature at fixed intervals, referred to as the monitoring window. In this scenario, for a fixed interval of t, the matrix e^{St} becomes constant. Consequently, the resulting equations that enable TEA to estimate the temperature are:

$$T_{steady_i} = \sum_{j=1}^{N} k_{i,j}^{-1} p_j + T_{amb} \tag{3.5}$$

$$T_i = \sum_{j=1}^{N} MWexp_{i,j}\left(T_{init_j} - T_{steady_j}\right) + T_{steady} \tag{3.6}$$

where:

- i and j are the indices of matrices and vectors;
- *MWexp* is the matrix exponential e^{St} for a fixed time interval, corresponding to the selected monitoring window.

At design time, we extract the matrices K^{-1} and e^{St} from MatEx. These matrices are loaded into the TEA's internal memory at startup. The required amount of memory is equal to $N_{PE}N_{TN} + N_{TN}^2$, where N_{PE} is the number of processing elements (PEs) and N_{TN} is the number of thermal nodes. Experimentally, Silva et al. [Silva et al., 2021] discovered that discretizing the matrix values into a 7-bit representation reduces the memory requirements by approximately 78% while maintaining the estimation error below 1% compared to MatEx. With the discretized values, TEA requires 119.72*KB* to perform temperature estimation for 81 PEs, and for 144 PEs, this value increases to 367.79*KB*. In addition to memory, TEA also requires a 32-bit multiplier-accumulator, enabling it to estimate the temperatures of up to 220 PEs with a monitoring window of 1 *ms* at a frequency of 1 *GHz*.

In this work, TEA was implemented at a higher abstraction level, modeled in C language as an OVP peripheral. At runtime, it performs the following actions: (*i*) periodically receives the power samples from the system's PEs; (*ii*) computes the current temperature of each PE using Equations 3.5 and 3.6; (*iii*) sends a packet with the estimated temperature of each PE to the Manager PE (MPE).

3.2.3 Energy Estimation

The energy samples are generated periodically at runtime by each PE. Each PE estimates its energy consumption based on three components: the processor, the router, and the memory, as shown in Equation 3.7. The size of these components represents more than 90% of the PE's occupied area [Rovinski et al., 2019].

$$E_{PE} = E_{processor} + E_{router} + E_{mem} \tag{3.7}$$

Each PE calculates its own energy consumption, E_{PE}, for the duration of the monitoring window, a period with a pre-defined fixed length. To determine its power dissipation, the PE multiplies E_{PE} by the time elapsed in the monitoring window. The resultant value quantifies the power dissipation of the PE during this interval. These power values, calculated by each PE, are then sent to TEA. The TEA uses this information to estimate the temperature of each PE, thereby providing an assessment of the system's thermal state.

Processor Characterization

The estimation of the processor energy consumption is based on the energy expended to execute each instruction during a designated monitoring window. As detailed in Chapter 2, each PE has an Instruction Counter (IC) module. This module is responsible for monitoring, categorizing, and counting the instructions fetched by the processor. To de-

termine the average energy consumption per instruction, we used current measurements provided in Fang's research [Fang, 2021] combined with the power budget presented in the Celerity design [Rovinski et al., 2019]. The resultant data, shown in Table 3.2, illustrates the average energy cost associated while executing different types of instructions at various voltage levels. The voltage/frequency configuration employed in this work is highlighted with a bold line in the table.

Table 3.2: Average energy cost to execute a given instruction in the processor RV32IM (28*nm*) executing with a certain voltage.

Voltage	**Max. Frequency**	**Energy per Instruction (*pJ*)**						
(*V*)	**(*MHz*)**	**BRANCH**	**REG**	**IMM**	**LOAD**	**STORE**	**JUMP**	**PC**
0,98	1400	34,72	22,00	21,60	29,14	29,55	34,70	20,80
0,94	1350	33,98	21,53	21,14	28,52	28,93	33,96	20,36
0,88	1300	31,82	20,16	19,79	26,70	27,08	31,79	19,06
0,86	1250	31,09	19,70	19,34	26,10	26,46	31,07	18,62
0,8	1200	28,92	18,32	17,99	24,28	24,62	28,90	17,32
0,78	1100	28,20	17,87	17,54	23,67	24,00	28,18	16,89
0,76	1050	27,48	17,41	17,09	23,06	23,39	27,46	16,46
0,74	**1000**	**26,75**	**16,95**	**16,64**	**22,45**	**22,77**	**26,73**	**16,02**
0,72	900	26,03	16,49	16,19	21,85	22,16	26,01	15,59
0,7	850	25,31	16,03	15,74	21,24	21,54	25,29	15,16
0,68	750	24,58	15,57	15,29	20,63	20,93	24,57	14,73
0,66	700	23,86	15,12	14,84	20,03	20,31	23,84	14,29
0,64	650	23,14	14,66	14,39	19,42	19,69	23,12	13,86
0,62	550	22,42	14,20	13,94	18,81	19,08	22,40	13,43
0,6	500	21,69	13,74	13,49	18,21	18,46	21,68	12,99

It is important to note that the energy consumption obtained for the target processor core (RV32IM) is comparable to the energy consumption reported by other works with similar technology in the literature. For instance, Zaruba and Benini [Zaruba and Benini, 2019] evaluated the instruction energy consumption in a 1.7*GHz* RV64GC in 22*nm* technology, and Silva et al. [da Silva, 2021] assessed the instruction energy consumption for a 1*GHz* Plasma (MIPS-like) in 28*nm* technology.

In this work, we categorize instructions into seven classes: (*i*) register instructions; (*ii*) immediate instructions; (*iii*) branch instructions; (*iv*) load instructions; (*v*) store instructions; (*vi*) jump instructions; and (*vii*) program counter (PC) instructions. Equation 3.8 specifies the dynamic energy consumption of each processor based on the data acquired at runtime.

$$E_{processor} = \sum_{k=1}^{n_{class}} \left(cI[k]_{x,y} \times E_{inst}[k] \right) \tag{3.8}$$

where:

- n_{class}: number of instruction classes;
- $cI[k]_{x,y}$: count of executed class-k instructions in PE x, y;
- $E_{inst}[k]$: average energy consumption to execute a class-k instruction.

Router Characterization

The internal components of the router include input buffers, a crossbar, and control logic. Consequently, router characterization involves stimulating all internal components and providing a payload with a significant Hamming distance between flits to induce high switching activity in the router's logic gates. The router energy characterization consists of four steps [Martins et al., 2014a].

The power dissipation of a router is a function of the reception rate in the input buffers [Ost et al., 2009]. Thus, the first step is to generate synthetic traffic (*step 1*) to analyze the switching activity of the input buffers. The generated traffic flow consisted of 1,000 packets, each containing 32 flits, with a Hamming distance between flits exceeding 80%. Subsequently, logical synthesis (*step 2*) of a 5-port router was performed, resulting in a netlist and a standard delay format (SDF) file. To account for the energy expended in data transmission between routers, the wire capacitance between routers was set at $200fF$ per $1mm$ of metal 5.

In a 3x3 NoC simulation (*step 3*), the central router was replaced with the netlist obtained in step 2. Each traffic scenario generated in step 1 produced a toggle count format (TCF) file, which contains the switching activity at the gate level. Finally, the power analysis (*step 4*) of the TCF file yielded the average power consumption for two reception rates: 100% for active mode and 0% for idle mode. During active mode, one flit is transmitted in each clock cycle, whereas during idle mode, the input buffer is either empty or the output is blocked, preventing any flit from advancing. Table 3.3 presents the router power characterization for both operation modes.

Table 3.3: Router average power. Library C28SOI_SC_12 (28nm), 1.0V@1GHz, 25ºC [Martins, 2018].

Operation Mode	**Input Buffer Power (μW)**	**Combinational Logic Power (μW)**	**Leakeage Power (μW)**
Idle	799.72	525.09	1.646
Active	1881.12	1896.38	

Based on these power values, and knowing the number of buffers present in the router (n_{buffer}), we can estimate the router's energy consumption when it is forwarding one flit (with one active buffer) using the following equation:

$$E_{active}^{cycle} = \left[(n_{buffer} - 1) P_{idle}^{buffer} + P_{active}^{buffer} + P_{active}^{comb} \right] T \tag{3.9}$$

When no buffers are active, the router is considered to be in an idle state, which affects the power consumption of the combinational circuitry. Thus, the energy expended by the router in the idle state is estimated using the following equation:

$$E_{idle}^{cycle} = \left[n_{buffer} P_{idle}^{buffer} + P_{idle}^{comb} \right] T \tag{3.10}$$

where:

- n_{buffer}: the number of buffers in the router; this number varies according to the PE's position in the network (central PEs have 5 buffers, edge PEs have 4, and corner PEs have 3);
- P_{idle}^{buffer}: the average power consumption of an idle buffer;
- P_{active}^{buffer}: the average power consumption of an active buffer while transmitting a flit;
- P_{active}^{comb}: the average power consumption of the combinational circuitry when active;
- P_{idle}^{comb}: the average power consumption of the combinational circuitry when idle;
- T: the period used to characterize the router.

As discussed in Chapter 2, flits are sent to their destination only at the end of each quantum. During the quantum execution, each sent packet remains waiting in the local buffer, and when the quantum ends, every packet is dispatched to its destination via the NoC. However, packets sent at the start of the quantum and those sent at the end will traverse the NoC simultaneously. Consequently, in our router energy estimation, we assume that each flit transmission occurs with only one active buffer, as described by Equation 3.11:

$$E_{router} = \left(E_{active}^{cycle} n_{flits}\right) + \left(E_{idle}^{cycle} \left(mw_{cycles} - n_{flits}\right)\right) \tag{3.11}$$

where:

- E_{active}^{cycle}: the amount of energy consumed by the router when transmitting flits;
- n_{flits}: the number of flits that have traversed the router in the past monitoring window;
- E_{idle}^{cycle}: the amount of energy consumed by the router while in idle;
- mw_{cycles}: the amount of cycles in one monitoring window.

Memory Characterization

Generally, a memory generator tool provides memories as black boxes within the technology design kit, without an RTL model. The CACTI-P [Li et al., 2011] tool models various memory types and generates estimations for access time, silicon area, and power. It also supports Dynamic Voltage Scaling (DVS). CACTI-P enables the characterization of the PE local memory's energy consumption, which is configured as a 64KB scratchpad memory with two ports. Table 3.4 displays the characterization data produced by CACTI-P. The CACTI-P tool allows for some technology options (such as cell and peripheral circuits) that differ from the industrial libraries of standard cells used previously. The technology

settings are calibrated to yield consistent results when compared with the processor and router. In Table 3.4, the access time corresponds to the period used to characterize the processor and router (1 ns), E_{load} is the dynamic read energy per access, and E_{store} is the dynamic write energy per access.

Table 3.4: Memory characterization. Library C28SOI_SC_12 (28nm), 1.0V@1GHz, 25ºC [Martins, 2018].

Access time (*ns*)	**Energy for Load** $E_{load}(pJ)$	**Energy for Store** $E_{store}(pJ)$
0.369	93.96	134.97

In the implementation of this work, the memory energy estimation is achieved by measuring the memory accesses made by the processor and the network interface with direct memory access capabilities. The processor memory accesses are monitored by the Instruction Counter.

Temperature estimation plays a central role in the proposed system's monitoring capabilities, facilitating the exploration of DTM techniques. However, as will be discussed in Chapter 5, the proposed mapping algorithm does not directly depend on the PEs' temperature. The primary objective of manycore thermal management is to maximize the system's performance while maintaining the chip within a certain temperature margin to prevent physical effects that could reduce the chip's reliability.

3.3 Reliability

Advancements in technology are hastening the emergence of reliability issues and contributing to the reduction of manycore system lifetimes. Reliability aims to ensure that the system's lifespan exceeds the targeted life expectancy and that the failure rate during the system's normal operational life remains below the targeted failure rate [Strong et al., 2009]. Various environmental factors can impact the reliability of integrated circuits, including voltage, temperature, rate of temperature change, current density, humidity, pressure, mechanical stress, process variability, and radiation. Ensuring long-term reliability is a critical objective for all manufacturers [Srinivasan et al., 2003].

System failures arise from errors in system operation, which can be broadly classified into two categories: *soft* and *hard* errors [Strong et al., 2009]. *Soft errors*, also known as transient faults or single-event upsets (SEUs), are caused by electrical noise or external radiation rather than by design or manufacturing defects. The architecture community has conducted extensive research to make manycores more resistant to soft errors. While most research on soft errors has concentrated on memory, recent studies have started to address errors in combinational logic. Soft errors can result in computation mistakes and data corruption, but they do not inflict permanent damage to the system and are not considered a

long-term reliability concern [Shivakumar et al., 2002]. On the other hand, *hard failures*, which can lead to permanent manycore failure, are caused by defects in the silicon or metallization of the manycore package. As the rate of hard failures increases, the lifetime of the manycore decreases inversely, indicating that hard failures are a significant determinant of manycore long-term reliability. Hard failures can be divided into two subcategories: *extrinsic* and *intrinsic* [Pecht et al., 2017].

Extrinsic failures typically decrease in frequency over time and are often the result of process or manufacturing defects. Contaminants on the silicon surface or surface roughness, for example, can lead to dielectric breakdown [Abadeer et al., 1999]. Extrinsic failures, such as shorts and open circuits in the interconnects due to incorrect metallization, are primarily caused by the manufacturing process. The micro-architecture of the system has minimal impact on the rate of extrinsic failures. Most of these failures can be detected early in the manycore's lifecycle. Burn-in and voltage screening are employed to identify manycores with extrinsic failures. After manufacturing, manycores undergo testing at elevated temperatures and voltages to accelerate the manifestation of extrinsic defects. This screening process removes extrinsic flaws before distribution, thus reducing the early life failure rate.

Intrinsic failures in manycore systems are linked to the gradual degradation of the hardware (*wear-out*), occurring during operation under specified usage conditions. These failures arise from factors inherent to the materials used in constructing the manycore, as well as process parameters, wafer packaging, and the design of the manycore itself. If the manufacturing process were devoid of any imperfections or errors, all failures in a manycore system would be classified as intrinsic. Such failures tend to increase over time and are typically caused by inherent defects in the materials of the manycore. It is crucial that these failures do not happen during the device's intended useful life when operated under specified conditions. Examples of intrinsic failures include time-dependent dielectric breakdown (TDDB) in the gate oxides, electromigration in the interconnects, and thermal cycling and cracking [Srinivasan et al., 2003].

The Bathtub Curve, shown in Figure 3.3, illustrates the long-term reliability of manycores as determined by the hard failure rate. It depicts the failure rate of manycores due to hard failures over time. Generally, $Z(t)$, the failure rate at time t, can be defined as the probability of a unit failing at time t, given that it has survived up to that point.

The Bathtub Curve is composed of three distinct sections associated with infant mortality (early life), useful life, and wear-out. Each section is characterized by different causes of failure. Early life failures are attributed to *extrinsic failures* and are primarily due to process and manufacturing defects, which decrease over time. Useful life failures are random and can be caused by various factors, but these tend to be infrequent, and the failure rate in this region is close to zero. Wear-out failures are *intrinsic* and result from material constraints, which increase over time. Burn-in and voltage screening aim to eliminate many-

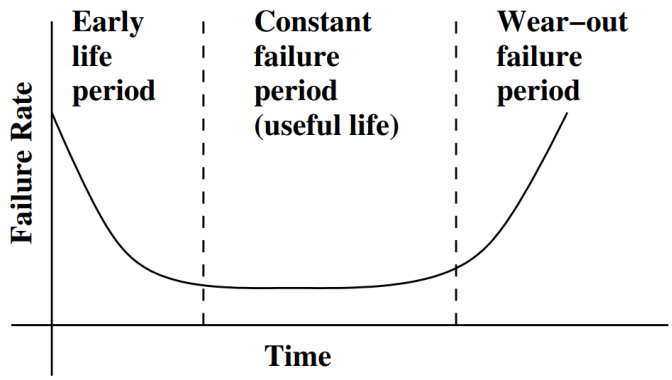


Figure 3.3: Variation of failure rate with time [Srinivasan et al., 2003].

cores with early life or extrinsic failures. Therefore, since the long-term reliability or lifetime of a manycore is mainly dependent on wear-out or intrinsic failures, these processes are critical for ensuring reliability.

There is considerable potential for enhancing long-term reliability from a management perspective. However, it is becoming increasingly challenging for manycores to satisfy performance and reliability margins, necessitating trade-offs among performance, cost, and reliability. Substantial research has been devoted to techniques that can optimize energy, thermal, and reliability performance by leveraging architectural features and adaptive capabilities (for instance, [Yu et al., 2015, Rathore et al., 2016a, Gou et al., 2018, Rathore et al., 2019b, Silva et al., 2020, Yoo et al., 2022]). One of the challenges encountered by studies that consider system reliability is the methodology used to evaluate the impact of the applied heuristic.

Studies often estimate system reliability using the mean time to failure (MTTF) metric [Ramachandran et al., 2008]. However, the calculation of MTTF in each study takes into account a variety of factors. For instance, [Yu et al., 2015] considers three effects: electromigration, gate oxide breakdown, and time-dependent dielectric breakdown, whereas [Rathore et al., 2019b] takes into account process variation along with three effects: negative bias temperature instability, electromigration, and hot carrier injection.

In our work, we will consider effects that are sensitive to the Arrhenius relationship [Arrhenius, 1889], which originates from the dependency of chemical reaction rates on temperature changes, thereby establishing a connection between manycore reliability and operating temperature. Assuming all other conditions are constant, the lifetime, $T_{failure}$, of a manycore resulting from a failure mechanism is determined by Equation 3.12.

$$T_{failure} \propto e^{\frac{E_a}{kT}} \tag{3.12}$$

where:

- E_a - activation energy of the failure mechanism in electron volts (eV);

- k - Boltzmann's constant ($8.62 \times 10^{-5} eV/K$);
- T - operating temperature in Kelvin (K).

The value of E_a is directly related to the specific failure mechanism under consideration. It is important to note that this model only accounts for the temperature dependence of failure mechanisms and is valid only when all other parameters remain constant. This approach cannot model failure mechanisms that are not influenced by temperature.

The Arrhenius model shows that the lifetime of a manycore decreases exponentially with an increase in temperature, indicating that temperature directly impacts manycore reliability. For example, hot spots on the manycore will lead to an accelerated rate of breakdown in those areas of the system [Yang et al., 2017a]. With the ability to estimate the temperature of each PE in the manycore simulator, as described in Section 3.2, we aim to identify major effects that are directly related to temperature behavior to evaluate our reliability-aware technique.

The metric used to evaluate reliability in our work is the mean time to failure (MTTF), which can be considered the average life expectancy of the manycore. We assume that all failure mechanisms have a constant failure rate in order to compute the MTTF. Although this assumption is not entirely accurate—as a typical wear-out mechanism, as depicted in Figure 3.3, will have a very low failure rate for an extended period before increasing sharply (excluding infant mortality) - it still enables us to combine different failure mechanisms and provide a unified MTTF. It is important to recognize that we are not modeling reliability as a function of time. Our work is capable of comparing the system reliability of different manycore management techniques and applications solely in terms of their MTTF.

Under the assumption of a constant failure rate, the mean time to failure (MTTF) can be calculated as the inverse of the failure rate. The standard method of expressing failure rates for semiconductor components is in Failures in Time (FIT) [Lala, 1996], which represents the number of failures per 10^9 device hours. Consequently, if the FIT is a constant, denoted as λ, then the MTTF, in hours, is given by:

$$MTTF = \frac{1}{\lambda} \tag{3.13}$$

3.3.1 Major Effects Affecting Reliability

Electromigration (EM)

Electromigration is a well-studied and understood cause of failure in semiconductor devices. Electromigration in aluminum and copper primarily occurs due to the movement of conductor metal atoms within the interconnects, driven by momentum transfer from the elec-

tron current. The electrons impart some of their momentum to the metal atoms in the interconnect, creating an “electron wind” that results in a net flow of metal atoms in the direction of electron flow. This can lead to a depletion of metal atoms in one area and an accumulation in others. Consequently, voids may form and grow at sites of depletion, potentially leading to open circuits, increased interconnect resistance, and other problems. Furthermore, extrusions can develop at accumulation sites, causing shorts between adjacent metal lines and leading to circuit failure.

The model used for the FIT due to electromigration (λ_{EM}) is based on Black’s electromigration equation [Black, 1969]. After isolating only the architectural variables for a given process, the FIT due to electromigration is modeled as:

$$\lambda_{EM} \propto \frac{e^{\frac{E_a}{kT}}}{V^n f^n p^n} \tag{3.14}$$

where:

- E_a - activation energy;
- k - Boltzmann’s constant;
- T - operating temperature;
- n - empirically determined constant, varying from 1 to 2 depending on the interconnect material; for our simulator, the value used is 1.1, which is relative to copper interconnections [Chandrakasan et al., 2001];
- V - supply voltage;
- f - clock frequency;
- p - switching probability.

Stress Migration (SM)

Stress migration is similar to electromigration, where metal atoms in the interconnects migrate due to mechanical stress. The primary causes of stress migration are intrinsic stresses, which originate from distortions in the crystal lattice of the semiconductor substrate, and thermo-mechanical stresses, which arise from the differing thermal expansion rates of materials within the device [Passage et al., 2019].

The model for stress migration is based on thermo-mechanical stresses generated by the varying thermal expansion rates of the device’s materials. The level of mechanical stress is proportional to the difference between the current temperature (T) and the stress-free temperature of the metal (T_0), which is the temperature at which the metal was initially

deposited. Thus, any temperature deviation from the metal deposition temperature will induce thermo-mechanical stresses. The equation used to estimate the FIT due to SM, λ_{SM}, is provided by [Srinivasan et al., 2003]:

$$\lambda_{SM} \propto |T_0 - T|^{-n} e^{\frac{E_a}{kT}} \tag{3.15}$$

where:

- E_a - the activation energy;
- k - the Boltzmann's constant;
- T - the operating temperature;
- T_0 - the stress free temperature;
- n - an empirically determined constant, varying from 2 to 3.

Time Dependent Dieletric Breakdown (TDDB)

Time-dependent dielectric breakdown (TDDB) is a critical failure mechanism in semiconductor devices. Over time, the gate dielectric weakens and ultimately fails upon the creation of a conductive path through the dielectric material. Once this conductive path forms between the gate and the substrate, it becomes impossible to regulate the current flow between the drain and the source using a gate electric field. Consequently, this renders the transistor device inoperative.

Manufacturers exercise extreme caution during the growth of the gate oxide to ensure its reliability and to avoid the incorporation of any impurities into the oxide. Historically, issues with gate-oxide breakdown were primarily extrinsic rather than intrinsic.

Gate oxide reliability is affected by factors such as temperature, the voltage applied at the gate, and the electric field present at the gate. Researchers generally accept that temperature significantly accelerates the degradation of gate-oxide reliability, and this degradation follows a relationship that is more severe than the Arrhenius model suggests. Various models have been developed to quantify TDDB degradation in terms of the electric field strength, its inverse, and the applied gate voltage. The model adopted for this work draws on the research by Wu et al. [Wu et al., 2002a, Wu et al., 2002b] from IBM. Wu et al. performed extensive analytical and experimental studies on TDDB, collecting experimental data across a broad spectrum of oxide thicknesses, voltages, and temperatures. Their work culminated in a unified TDDB breakdown model applicable to both current and future generations of ultra-thin gate oxides. According to the model proposed by Wu et al., the lifetime associated with TDDB for ultra-thin gate oxides not only depends on voltage but also suffers from accelerated degradation with increased temperature, exceeding an exponential

rate. In [Srinivasan, 2006], the author merges Wu et al.'s model with a traditional Arrhenius temperature-dependence approach, leading to the definition of TBBD FIT, λ_{TBBD}, model as follows:

$$\lambda_{TDDB} \propto \left(\frac{1}{V}\right)^{a-bT} e^{\frac{A+\frac{B}{T}+CT}{kT}} \tag{3.16}$$

where a, b, A, B, and C are fitting parameters based on data from [Wu et al., 2002a], we have:

- $a = 78$
- $b = -0.081$
- $A = 0.759\,\text{eV}$
- $B = 66.8\,\text{eV/K}$
- $C = -8.37 \times 10^{-4}\,\text{eV/K}$

Thermal Cycling (TC)

Thermal cycling (TC) can cause fatigue failures. Each cycle accumulates permanent damage until failure eventually occurs. Even the regular act of powering a device up and down can result in damage. Solder joints, which form the interface between the package and the die, are the parts of the device most susceptible to damage. The package undergoes thermal cycles that can manifest as large, low-frequency cycles, such as when powering up or down or entering low power or standby mode; or as small, high-frequency cycles that occur several times per second, resulting from changes in workload behavior and context switching. Researchers employ the Coffin-Manson equation [JEDEC, 2016] to model thermal cycles. This model allows the derivation of the FIT due TC equation:

$$\lambda_{TC} \propto \left(\frac{1}{T - T_{ambient}}\right)^{q} \tag{3.17}$$

where:

- T represents the operating temperature;
- $T_{ambient}$ is the ambient temperature;
- q stands for the Coffin-Manson exponent, which is an empirically determined constant with a value of 2.35 [JEDEC, 2016].

Negative Bias Temperature Instability (NBTI)

Negative bias temperature instability (NBTI) is an electrochemical reaction occurring in p-FETs when the gate is negatively biased relative to the source and drain. This usually happens when the gate input is low, and the output is high, leading to a buildup of positive charges in the gate oxide. This buildup increases the transistor's threshold voltage, thus reducing the gate overdrive (the difference between the supply voltage and the threshold voltage). As a result, it can slow down the gate's performance and may eventually cause processor failure due to timing issues [Zafar, 2007]. NBTI exhibits strong positive temperature and field dependence; it is exacerbated by higher temperatures on the chip due to scaling. Moreover, scaling causes the gate oxide to thin out, intensifying the reliability concerns associated with NBTI.

We based the adopted NBTI model on the research conducted by Zafar et al. from IBM; it is a physics-based model verified through both new and previously published NBTI failure data. Clearly, NBTI-induced lifetime degradation strongly correlates with temperature. To calculate the FIT due to NBTI, λ_{NBTI} at a specific temperature, we use the following equation:

$$\lambda_{NBTI} \propto \left[\left(ln\left(\frac{A}{1+2e^{\frac{B}{kT}}} \right) - ln\left(\frac{A}{1+2e^{\frac{B}{kT}}} - C \right) \right) \frac{T}{e^{\frac{-D}{kT}}} \right]^{\frac{1}{\beta}} \tag{3.18}$$

where A, B, C, D, and β represent fitting parameters derived from the data in [Zafar, 2007]:

- $A = 1.6328$;
- $B = 0.07377$;
- $C = 0.01$;
- $D = -0.06852$;
- $\beta = 0.3$.

3.3.2 Reliability Model

Up to this point, we have presented the models used to estimate the FIT of each specific failure mechanism considered in our system model. To assess the system's overall reliability or MTTF, we must combine the FITs of all individual failure mechanisms. In this thesis, we employ two techniques for this integration. The first technique utilizes a simpler method, the sum-of-failure-rates (SOFR) model, which facilitates its use during the training phase of our reinforcement learning algorithm. The second technique incorporates a

lognormal distribution model, which, despite being more complex and limiting its use during training, offers greater accuracy. This technique is employed to calculate the results presented in Chapter 6.

Sum-of-Failure-Rates - SOFR

Our manycore system's fundamental building block is the PE, and its operational integrity is required for the system's functionality. In the absence of fault-tolerance mechanisms, the potential failure of even a single PE makes the entire system inoperative. Consequently, we characterize the manycore system as a series system, where the reliability is dependent on the uninterrupted availability of all PEs. Assuming that the FIT of the *i*th PE of our manycore is represented by λ_i. Then the system FIT, λ_{sys}, is expressed express through the summation of the FIT of all individual PEs, Equation (3.19), as demonstrated in [Trivedi, 2016].

$$\lambda_{sys} = \sum_{i=1}^{n} \lambda_i \tag{3.19}$$

However, we do not possess the individual FIT for each PE; what has been estimated is the FIT for each aging effect. Therefore, just as manycore is a series system of PEs, we consider the PE to be a series system of its aging effects. In other words, if any of the five effects (EM, SM, TDDB, TC, and NBTI) produce a failure, we declare that the PE has failed. Let λ_{ik} represent the FIT for the *k*th aging effect on the *i*th PE. We can then express the FIT of the *i*th PE, λ_i, as the sum of the FITs for all individual wear-out effects, as shown in Equation (3.20).

$$\lambda_{ik} = \sum_{k=1}^{5} \lambda_k \tag{3.20}$$

Finally, we can combine Equation (3.19) and Equation (3.20) to derive the manycore FIT, λ_{sys}, as a function of the aging effects on individual PEs. Assuming that the lifetime follows an exponential distribution [Trivedi, 2016], we can conclude that the manycore MTTF, denoted by $MTTF_{sys}$, is calculated by Equation (3.21):

$$MTTF_{sys} = \frac{1}{\lambda_{sys}} = \frac{1}{\sum_{i=1}^{n}\sum_{k=1}^{5}\lambda_{ik}} \tag{3.21}$$

Equation 3.21 demonstrates that the manycore MTTF is inversely proportional to the sum of all failure mechanisms' rates for all components, which gives rise to the common remark that "a system is weaker than its weakest lin" [Trivedi, 2016]. During the simulation, we determine and store the system's FIT rate at each monitoring interval.

The MTTF estimates were integrated into the Thermal Estimation Accelerator (TEA) that was previously described in Section 3.2.2. Thus, after estimating each PE's temperature using the TEA at every monitoring interval, we also estimate the FIT for each PE. In Chapter 5, we will explore how we use this data to train a lightweight Q-learning table that assigns tasks to the system to enhance the manycore MTTF.

Lognormal Distribution

The SOFR model for computing the MTTF requires the simplification that considers every individual failure mechanism has an exponential lifetime distribution. However, this assumption is not accurate because typical wear-out failure mechanisms typically exhibit a low FIT rate at the beginning. The most accurate failure distribution for each mechanism is still under debate; however, lognormal distributions are widely applicable for a variety of wear-out mechanisms [Wan et al., 2020].

For calculating the total MTTF of our manycore system with a lognormal distribution assumption, we employed a method proposed by Srinivasan et al. [Srinivasan et al., 2005] known as RAMP 2.0. This method uses a Monte Carlo simulation to integrate the effects of individual lognormal distributions across all wear-out mechanisms and components. Given the complexity of the lognormal distribution combined with the large cross-product of components and mechanisms, analytically determining manycore reliability is computationally challenging. Srinivasan et al. suggest using a Monte Carlo simulation as a solution to this problem. The Monte Carlo simulation is an algorithm that solves problems by generating appropriate random numbers and noting the proportion of numbers that satisfy specific conditions or properties.

We employed this technique to calculate the system MTTF, $MTTF_{sys}$, which we present in the Results, Chapter 6, of this Thesis. Nevertheless, the following Chapter will first review and discuss management strategies from the literature that address reliability issues related to temperature fluctuations.

4. RELATED WORK

Dynamic Thermal Management (DTM) and Dynamic Reliability Management (DRM) are important aspects of the manycore design. DTM concentrates on the efficient management of heat generated by manycore circuits, whereas DRM aims at mitigating failures and malfunctions in system components to extend the lifetime of manycore systems. DTM and DRM are interlinked, as a robust thermal management system is imperative for ensuring overall system reliability. In this Chapter, we present and discuss relevant literature related to both DTM and DRM. Additionally, we explore their application in enhancing the performance and reliability of electronic components in manycore systems.

4.1 FoToNoC

In their work, Yang et al. [Yang et al., 2017a, Yang et al., 2017b] introduce a hierarchical management strategy for heterogeneous manycore systems using a folded torus Network-on-Chip (FoToNoC). In the FoToNoC architecture, cores are interconnected in a way that reduces communication latency and are organized into logically condensed virtual clusters. The proposed architecture consists of a 64-core NoC), arranged in an 8x8 array of tiles.

Figure 4.1(a) illustrates the physical arrangement of the cores on the platform, numbered from c_1 to c_{64}; (b) demonstrates how the connections between cores are arranged in the folded torus; and (c) shows the distinction between the physical view and logical view of the clusters.

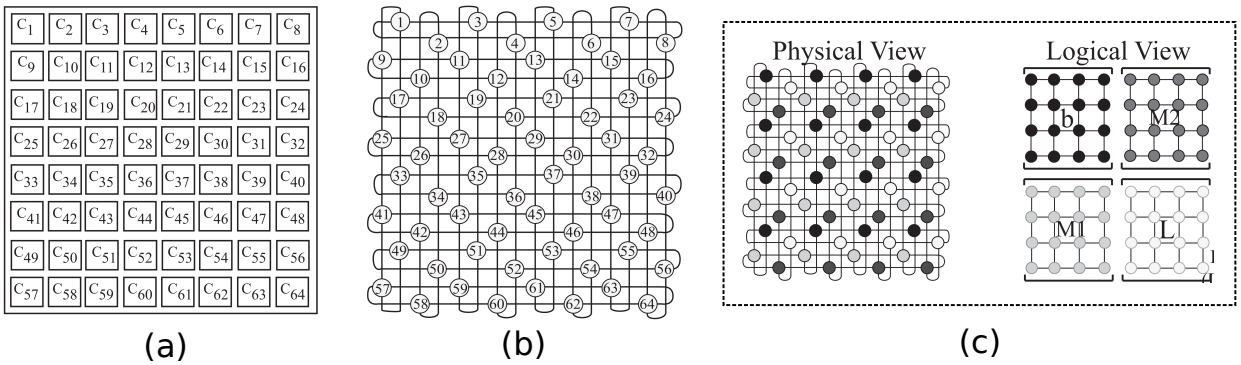


Figure 4.1: (a) Arrangement of 64 cores on physical floorplan.(b) Folded torus.(c) Physical and logical views [Yang et al., 2017b].

In the application mapping process, tasks are preferentially allocated to cores with less communication delay, such as cores 1, 3, 5, and 19 in Figure 4.1(b). This strategy en-

ables a better distribution of heat across physically dispersed cores on the floorplan, hence preventing the formation of hotspots.

In this architecture, there exist four types of cores. Each logical cluster contains one core type, namely *b* (*big* cores with enhanced performance), M_1, M_2, and L (*Little* cores with minimal energy consumption). These cores of different types are arranged physically and logically, as depicted in Figure 4.1(c). The authors assume that at any given moment, only one type of core or, in other words, one cluster will be powered on.

This study employs power models from the Alpha 21264 processor in 22-nm technology, as derived from *mcpat09* [Li et al., 2009]. The simulation of the microarchitecture uses *GEM5* [Binkert et al., 2011a]. Mapping decisions are based on runtime transient and peak temperature estimations provided by *MatEx* [Pagani et al., 2015a]. Results demonstrate improved heat distribution, superior to contiguous cluster mapping, and better communication delay compared to a standard torus network using decentralized mapping.

4.2 Dynamically Reconfigurable NoC

Liu et al. [Liu et al., 2018] propose a homogeneous manycore platform where routers can be reconfigured to enable multi-hop bypass on the NoC, thus reducing communication latency between non-adjacent cores. The authors also propose an Integer-Linear Programming (ILP) model based on this platform to investigate the network reconfiguration architecture that applies task mapping with minimum network contention. With the ILP model in place, they further introduce a heuristic, *TopoMap*, which is executed in polynomial time, with minimal impacts on communication and application performance.

Additionally, the authors propose a thermal-aware task mapping strategy, taking into account the Euclidean distance between cores. They assume that any processor located next to active cores will likely remain dark, as depicted with cores 11, 18, 20, and 27 in Figure 4.2(a). Furthermore, as long as there are available cores meeting this requirement, task mapping to cores is restricted to those at least 2-hop away from active cores, as shown in Figure 4.2(a). If no such cores are available, selection is among candidates that will not cause a temperature violation. The authors demonstrate that this pattern of application mapping can maintain thermal reliability in improved heat conditions, benefiting from the alternating arrangement of active and dark cores.

The primary disadvantage of implementing mapping applications in physically decentralized cores is the increase in the communication distance. Typically, a packet transmission's latency is directly proportional to both the distance it travels and the routing stages in the routers. To overcome the latency issue, the authors introduce a mapping algorithm with SMART NoC. SMART, a reconfigurable network, allows for single-cycle multi-hop bypass,

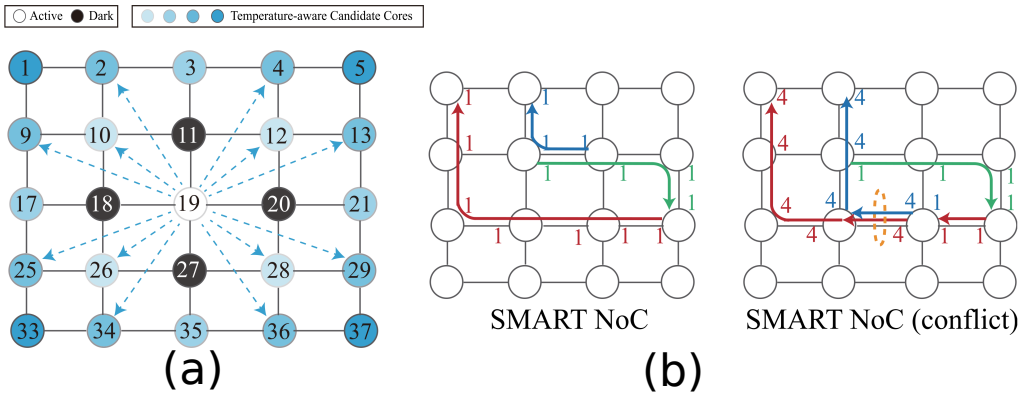


Figure 4.2: (a) Thermal-aware mapping strategy.(b) Contention in SMART NoC [Liu et al., 2018].

where flits can traverse multiple routers in one clock cycle, up to a HPC_{max}, representing the flit's maximum allowable hop count.

However, drawbacks persist, as SMART is susceptible to conflicts. Concurrent flits must stop and buffer, as illustrated in Figure 4.2(b), where arrows of various colors demonstrate data transmission between two processors, and a dotted ellipse in one link means that the two transmissions conflict. Consequently, the authors propose *Cont_Map*, a communication contention-aware task mapping where paths are analyzed to avoid transmission overlap among tasks.

Finally, the authors propose a thermal-aware mapping algorithm that prevents contention for SMART NoC, integrating both previously suggested mapping strategies. The algorithm defines the task's hop count from 2 to HPC_{max}, thus allowing the exploration of all candidate cores while simultaneously compressing the search space. To retrieve transient temperatures of all cores and calculate chip peak temperature at runtime, this mapping algorithm employs *MatEx* [Pagani et al., 2015a].

Simulation execution takes place using McPAT [Li et al., 2009] to gather power data, and GEM5 [Binkert et al., 2011a] to simulate both communication and microarchitecture. Results reveal a performance increase of up to 33.5% when contrasting the utilization of SMART NoC with a traditional mesh NoC, primarily attributable to decreased latency and contention rates. Additionally, authors noted a reduction in energy consumption when comparing the proposed architecture with a conventional mesh NoC and FoToNoC [Yang et al., 2017b]. This reduction is mainly due to enhanced execution speed and communication efficiency.

4.3 TSP: Thermal Safe Power

Pagani et al.[Pagani et al., 2017] introduced a novel power budget for manycore systems to maximize power efficiency. This new approach serves as an alternative to the conventional Thermal Design Power (TDP). TDP is a uniform power value, that serves to prevent system complications arising from excessive temperatures. However, TDP tends to lead to system under-utilization, especially when the mapping process results in system hotspots.

In contrast, the proposed metric, the Thermal Safe Power (TSP), considers the unique power capacity of each core and takes into account the arrangement of active/dark cores, rather than evaluating the system's power as a whole. Figure 4.3 presents two mappings of six active cores within a 4x4 manycore system, where the TSP is 80°C. In the worst-case mapping scenario, the maximum core power is lower than that in the best-case scenario due to the surrounding temperature. With the pattern mapping, active cores can operate at higher power levels as it reduces heat conduction from nearby active cores.

Notably, all cores do not necessarily reach the maximum temperature. Hence, there is potential for performance enhancement, implying that some cores might have the capacity to consume more power. While this adaptable technique is described as a function of power, it abstracts the specific method (e.g., DVFS) applied to adjust the power.

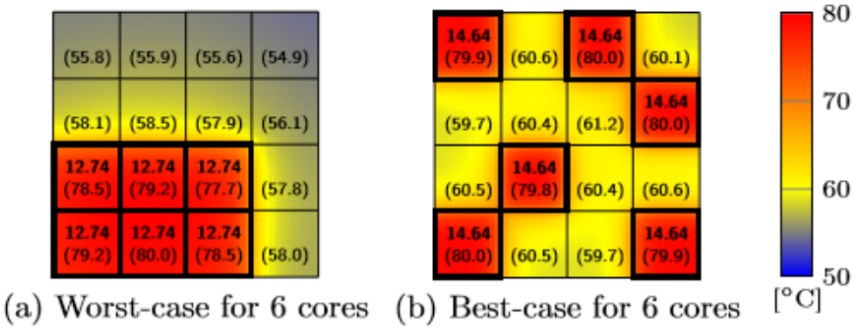


Figure 4.3: This example illustrates the best-case and worst-case mappings in relation to thermal constraints. The bold numbers at the top represent the power of the active cores, measured in watts. The numbers at the bottom, enclosed in parentheses, represent the core temperatures in ℃ [Pagani et al., 2014].

The study's evaluation process involves a simulation platform using GEM5 [Binkert et al., 2011a], McPAT [Li et al., 2009], and HotSpot [Huang et al., 2006]. The authors compared the computed TSP values for unique mappings with constant power budgets applicable to the entire system. The results demonstrated that utilizing TSP as a power constraint enhanced total performance in the majority of cases. However, the authors found that em-

ploying a single power budget for the entire system could yield superior results in some cases, primarily because cores may not have an exact speed step that fulfills the TSP value.

4.4 seBoost

Pagani et al. [Pagani et al., 2015b] introduced a selective boosting technique for heterogeneous manycore systems, called seBoost. The primary goal of this technique is to establish an efficient runtime boosting mechanism that satisfies the runtime requirements of applications. It aims to achieve minimum performance losses for the applications operating on the non-boosted cores. This technique is based on prior knowledge of power profiles for each thread and application across all voltage and frequency (VF) pairs. Utilizing this data, the algorithm aims to identify the optimal combination of VF pairs for both the boosted and non-boosted cores to maintain the system's temperature below the critical level. This approach requires runtime temperature monitoring as the temperature is required. The algorithm calculates both steady and transient temperatures to verify VF configuration amongst cores.

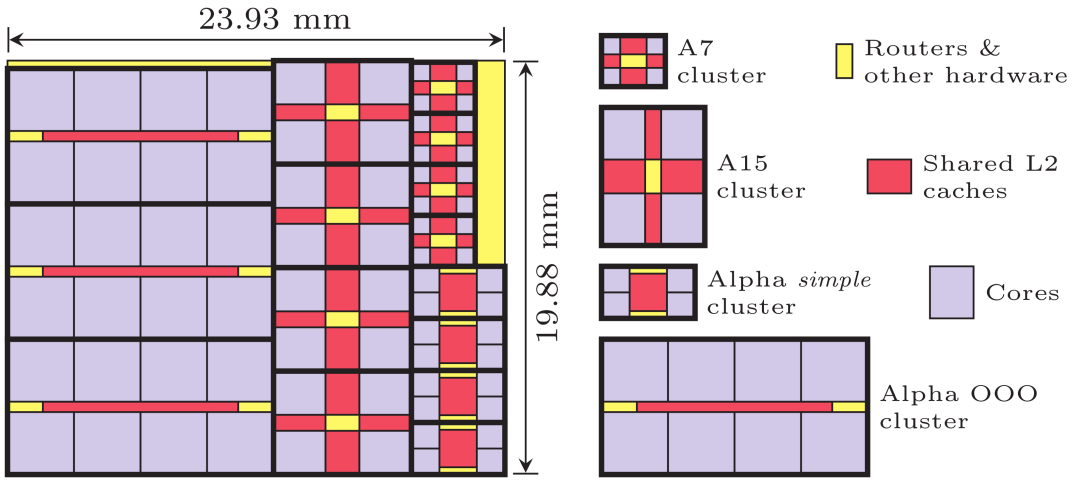


Figure 4.4: Heterogeneous manycore setup used to evaluate seBoost proposal [Pagani et al., 2015b].

This work employs a heterogeneous architecture with 72 cores, which are divided into 24 high-performance Alpha cores, 16 simple Alpha cores, 16 ARM A15, and 16 energy-efficient ARM A7. The evaluation of this architecture relies on the use of GEM5 [Binkert et al., 2011a], McPAT [Li et al., 2009], HotSpot [Huang et al., 2006], and the measurement of traces on a commercial platform which employs ARM's big.LITTLE architecture. The authors compare the proposed strategy with Intel's Turbo Boost technology and a simple boosting method that throttles down the non-boosted cores to their lowest frequencies.

The results indicate that if the performance boosting requirement is feasible, given an initial temperature vector, both the proposed technique and the simple boosting method

can meet the requirements 100% of the time. However, with the simple boosting technique, the non-boosted application may experience unnecessary performance losses. When using Turbo Boost, the non-boosted cores reach higher average performance, but in most cases, Turbo Boost doesn't meet the runtime requirements for the boosted cores throughout the entire boosting interval. Given that Turbo Boost has no knowledge of the application's performance requirement, it can set VF levels higher than necessary. This can lead to reaching the critical temperature before the end boosting interval, triggering the control mechanism and consequently degrading the performance.

4.5 M-Oscillating

Sha et al. [Sha et al., 2018a] propose a frequency oscillation-based technique to maximize the throughput performance of multi-core platforms while ensuring the peak temperature constraint. The proposed technique is based on two concepts: *step-up schedule* and *m-Oscillating schedule*. The proposed technique focuses on periodic schedules that can deliver steady and sustainable performance.

The paper discusses the importance of this technique by assuming that each processor features discrete running modes, and previous works focused on solving the throughput maximization problem by assuming that the speed of each core can be continuously and instantaneously varied. Based on previous work, one method to maintain the peak temperature constraint is to use the first discrete speed available below the calculated in the maximization problem, which is named in the paper as LNS (lower neighboring method). However, the results of this technique might be overly pessimistic when the available speed level is limited. To improve this technique, they propose an algorithm that searches all speed combinations to find one that can maximize the throughput without exceeding the temperature threshold, called EXS (exhaustive search). The main limitation of LNS and EXS is that each core can only execute one single speed, and the slack in temperature cannot be filled by raising the speed of any core due to the possible violation of the maximum allowed temperature.

Based on the limitations of LNS and EXS, the goal of this work is to maximize the throughput of a multi-core platform, where the processors have available discrete frequencies, using a periodic schedule of two or more frequencies to use the temperature slack to improve performance. To define this schedule, the Authors first define the step-up schedule, which is basically a periodic schedule with multiple state intervals where the processing speed (VF levels) increases at each interval. Based on a step-up schedule the authors define a m-Oscilating schedule, which is a method to determine the length of the period for the periodic schedule.

The m-Oscilating schedule is derived from a step-up schedule by scaling down each interval length by m. The Authors prove that the smaller the period is (higher m), the higher the throughput can be achieved. Considering different values of m, the total dynamic energy consumption and the average temperature remain constant, but the peak temperature tends to reduce with higher values of m. The main drawback of using higher values of m is the transition overhead to change VF settings on a processor, so the Authors take this overhead into consideration when defining the schedule.

This work uses its own temperature estimation model to define the stepup schedule which ensures that the temperature will not exceed the critical one. The algorithm constructs possible schedules and determines the highest temperatures to verify if the peak temperature constraint is guaranteed. The experimental results use different numbers of cores, up to 9, and possible voltage levels, up to 5. The proposed technique is simulated on hypothetical configurations of Alpha 21264 cores using power parameters from McPAT [Li et al., 2009] simulator and temperatures collected from HotSpot [Huang et al., 2006]. Results show an average improvement of 11% in performance when comparing the proposed technique with the EXS, which only allows the use of a single speed for each core.

4.6 TCTS: Temperature Constrained Task Selection

Li et al. [Li et al., 2015, Li et al., 2018] proposed a simplified thermal model, predicting steady temperatures in manycore systems given task-to-core mapping and power consumption rates of the tasks. They integrated a mixed-integer linear programming (MILP) model with the thermal model, for achieving the optimal task assignment that results in the lowest chip peak temperature. If the optimal task-to-core assignment stays within the safe temperature limit, they further propose a temperature-constrained task selection (TCTS) algorithm aimed at optimizing performance within the safe temperature threshold.

The mapping algorithm requires several inputs, including the computation demand of the tasks set for mapping, the available voltage and frequency levels of the system, the power consumption of the cores operating at each frequency, and the thermal model of the system. To achieve runtime temperature estimation, the authors propose a simplified version of the HotSpot [Huang et al., 2006] thermal model which necessitates scaling factors. These scaling factors need calibration through repeated executions of the HotSpot with random power samples to reduce the error caused by this simplified model. The algorithm uses the proposed thermal model to minimize the system peak temperature, searching for the best task-to-core assignment by solving the MILP formulation.

Finally, this work proposes the TCTS algorithm, exemplified in Figure 4.5. The purpose of this algorithm is to select the best set of tasks that can be completed in the system without crossing the safe temperature threshold. This precaution applies even when

the optimal mapping discovered by the MILP algorithm surpasses the peak temperature of the safe constraint.

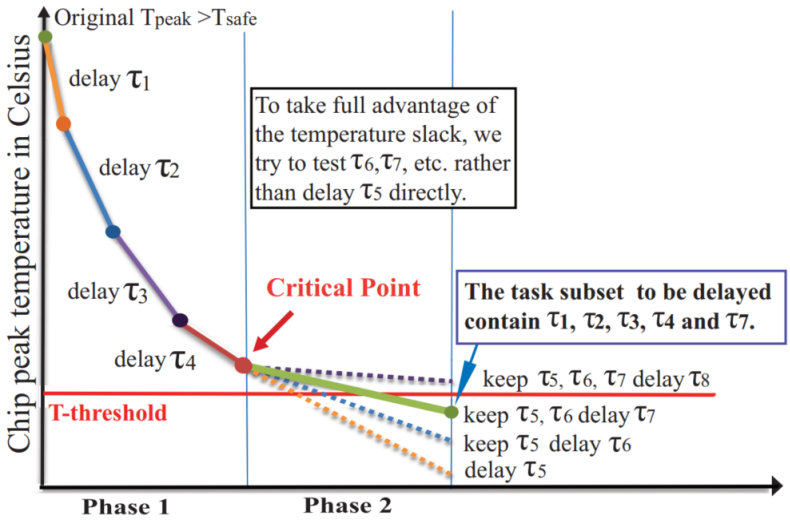


Figure 4.5: Illustration of TCTS algorithm [Li et al., 2018].

The methodology behind the TCTS algorithm relies on a greedy selection process. It delays tasks with higher computing demands until it reaches a critical point. This point, depicted in Phase 1 of Figure 4.5, is where all remaining tasks can be maintained within the thermal constraint. Beyond this point, the algorithm tries to discard tasks with lesser performance demands to maximize the system's throughput within the safe temperature limit. This is presented in Phase 2 of Figure 4.5. The authors provide evidence in the paper to prove that this greedy algorithm also determines the optimal solution to define which tasks require delay in order to use the system at its full capacity.

The results in this paper primarily illustrate the accuracy of the thermal model. They also serve to compare the MILP-based mapping with random and continuous mapping, by measuring the peak temperature achieved by each mapping for each computation demand. In comparison with results extracted from HotSpot [Huang et al., 2006] simulations, the proposed thermal model results show a maximum error of 0.31 ℃ in temperature prediction.

The mapping results have taken into consideration a system with 16 Alpha 21364 cores arranged in a 4x4 grid, with McPAT [Li et al., 2009] estimating the power consumption of tasks. These results indicate an average temperature reduction of 2.98 ℃ when comparing the proposed technique with random mapping, and 4.59 ℃ when compared with continuous mapping. This shows that TCTS has enabled the execution of more tasks within the safe temperature limit.

4.7 LF: Longevity Framework

Rathore et al. [Rathore et al., 2016b] propose a task mapping technique that constrains performance to enhance lifetime reliability by mitigating negative-bias temperature instability (NBTI) and taking into account on-chip process variation. The approach begins at *design time*, where the Threshold Accepting Simulated Annealing (TASA) algorithm [Dueck and Scheuer, 1990] is used to identify a Pareto-optimal mapping that maximizes the mean time to failure (MTTF) while satisfying throughput constraints. In simpler terms, the TASA algorithm helps to map tasks to cores in a way that not only extends the maximum MTTF but also meets throughput requirements.

At *runtime*, the strategy proactively reassigns tasks from aging hotspots to healthier cores that can meet the performance constraints, thereby accommodating them. Sensors in situ monitor core aging due to NBTI [Singh et al., 2011]. The researchers conducted experiments on a 16-core manycore system with process variations, and the results show that the design-time optimization leads to improvements of up to 54% in lifetime. Moreover, the runtime phase ensures that the MTTF remains above the levels achieved during the design-time phase.

Rathore et al. [Rathore et al., 2018, Rathore et al., 2019b] proposed HiMap, a dynamic, hierarchical mapping approach designed to maximize the lifetime reliability of manycore systems. This approach also satisfies performance, power, and thermal constraints. HiMap is both process variation-aware and aging-aware. It efficiently determines the optimal mapping and placement of dark cores to boost the system's lifetime reliability while adhering to performance, thermal, and power constraints, employing a two-level hierarchical strategy.

The *first level* identifies a cluster of cores suitable for mapping an application. At the *second level*, the approach ensures uniform aging within the cluster. It accomplishes this by assigning threads that cause more aging to relatively healthier cores. Additionally, the strategy utilizes dark cores interleaved within the cluster for thermal mitigation. The authors focus solely on aging due to electron migration (EM), as it represents one of the primary aging phenomena in manycore systems.

Rathore et al. introduce the Longevity Framework (LF) in [Rathore et al., 2021]. The LF combines the previously suggested hierarchical mapping approach, HiMap, with per-core Voltage-Frequency (VF) selection. The per-core VF selector utilizes the principle that selecting the minimum frequency level capable of meeting a task's performance requirements when mapping to a core optimizes for aging minimization and lifetime reliability maximization.

The manycore system consists of several cores grouped into blocks for managing the complexity and scalability of mapping exploration (see Figure 4.6(a), which depicts a 64-

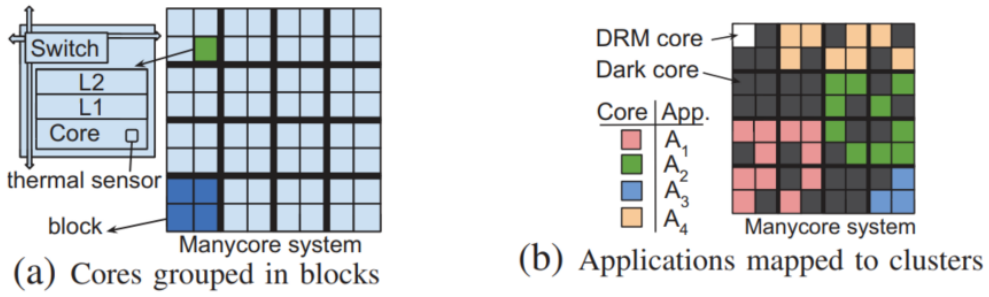


Figure 4.6: Illustration of Blocks and Application Clusters with DRM (Dynamic Lifetime Reliability Manager) [Rathore et al., 2018].

core system with 2x2 sized blocks). These blocks are organized hierarchically into clusters, and the system maps the threads of an application to these clusters (as shown in Figure 4.6(b), where 4 clusters have 4 applications mapped onto them). In-site sensors measure the temperature on each core. One of the cores runs the DRM (Dynamic Lifetime Reliability Manager) application, which conducts the mapping exploration at the end of each epoch, in parallel with application execution. The DRM evaluates aging and Mean Time to Failure (MTTF) at the close of every epoch, considering factors such as core process variation, temperature, and aging. Additionally, it maintains an aging database that includes per-core aging data, frequency, and health status.

The authors performed experiments using the Snipersim manycore simulator [Heirman et al., 2012], which was interfaced with the McPAT power simulator [Li et al., 2009], the Hotspot thermal simulator [Zhang et al., 2015], and tools for aging and MTTF assessment. They updated the Hotspot leakage model to include a process variation-aware adaptation of the temperature-dependent leakage model presented in [Chaturvedi et al., 2012]. They conducted the experiments on systems with 64 and 256 Nehalem cores and used applications from [Woo et al., 1995] and [Bienia et al., 2008]. They compared the results with two aging-aware mapping approaches: Hayat [Gnad et al., 2015] and Reliability-aware Mapping (RM) [Haghbayan et al., 2016]. The authors claim that their results for the 64- and 256-core systems validate HiMap's effectiveness and demonstrate significant improvements over the current state-of-the-art.

4.8 Hard and Soft Error-aware

Kim et al. [Kim et al., 2017] propose new techniques for optimizing energy use and extending the lifetime of emerging dark silicon manycore microprocessors, taking into account both long-term reliability effects, such as hard errors, and transient soft errors. They utilize a proposed physics-based electromigration (EM) reliability model [Huang et al., 2014] to predict reliability issues caused by EM. To enhance EM-induced lifetime and save en-

ergy, they employ an adaptive Q-learning-based method, which is well-suited for dynamic runtime operations due to its ability to provide cost-effective, yet high-quality solutions. Additionally, they use a mixed-integer linear programming (MILP) method, which often results in better solutions at higher computational costs. To increase the microprocessors' lifetime, they implement actions such as Dynamic Voltage and Frequency Scaling (DVFS) and power gating.

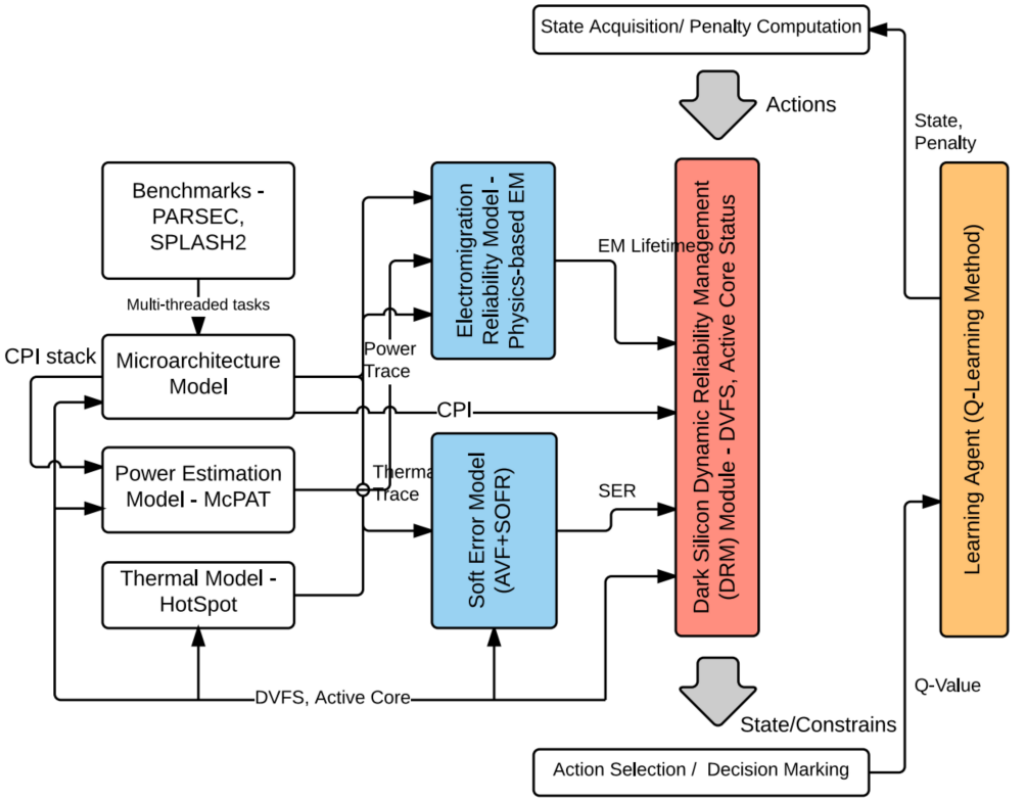


Figure 4.7: Q-Learning Model Utilizing a Reliability-Aware Dark Silicon Framework: AVF represents the Architecture Vulnerability Factor; SOFR denotes the Sum of Failure Rates; EM stands for Electromigration; SER indicates the Soft Error Rate; CPI means Cycles Per Instruction [Kim et al., 2017].

Figure 4.7 presents the Q-learning reliability-aware lifetime/energy optimization framework. The framework comprises an environment that houses the manycore system, along with the Q-learning algorithm that serves as the learning agent. This learning agent acquires the state of the environment, computes the penalty function, and then determines the subsequent action to take. The authors evaluated the proposal by using the Sniper simulator ([Heirman et al., 2012]) designed for manycore systems, with two benchmarks: PARSEC ([Bienia et al., 2008]) and SPLASH-2 ([Woo et al., 1995]). For power estimation, they utilized McPAT ([Li et al., 2009]), and to model thermal effects, they employed HotSpot ([Zhang et al., 2015]). The authors claim that their proposed methods effectively optimize performance and lifetime in a 64-core system while considering both soft and hard reliability constraints.

4.9 LBRM: Lifetime Budgeting Reliability Management

Wang et al. [Wang et al., 2018] propose a runtime application mapping scheme called Lifetime Budgeting Reliability Management (LBRM). This scheme employs a borrowing strategy to enhance the throughput of manycore systems within a given lifetime constraint. They begin by categorizing applications into two groups: communication-intensive and computation-intensive. Following classification, they map communication-intensive applications to tight, near-square regions to reduce communication costs. In contrast, they distribute computation-intensive applications across dispersed regions to prevent core overstressing (see Figure 4.8).

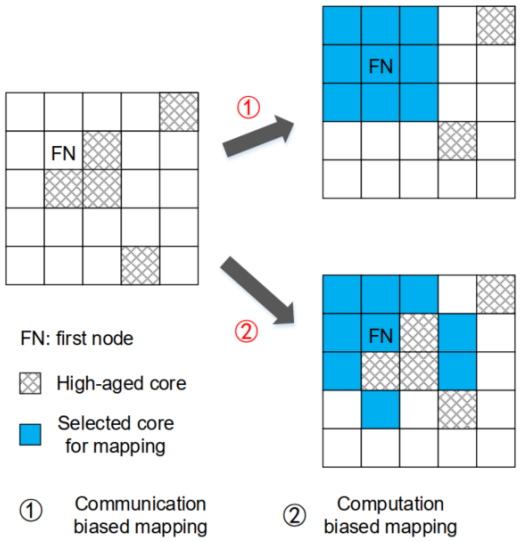


Figure 4.8: Diagram of neighborhood node allocation including two subroutines: (1) communication-biased mapping, suitable for communication-intensive applications, and (2) computation-biased mapping, ideal for computation-intensive applications [Wang et al., 2018].

The implementation of the borrowing strategy involves a two-level controller that uses application mapping to manage the lifetime reliability and performance. This controller decides whether the incoming application is mapped or enqueued. At the first level, the *long-term controller* organizes applications into batches, with each batch holding up to N applications. Upon the arrival of a new application, the controller adds it to the current batch if it has fewer than N applications; otherwise, the application goes into the next batch.

At the second level, the *short-term controller* determines the timing and method for mapping the next application in the current batch. Each controller bases its decisions on the lifetime budget allocated to each core. The long-term controller will only release the next batch if the system's total budget is positive. Meanwhile, the short-term controller considers each core's budget when selecting the mapping position and deciding when to release it.

The experiments were performed using an in-house manycore simulator. The power consumption was modeled with McPAT [Li et al., 2009]. They applied the HotSpot thermal model [Zhang et al., 2015], assuming the existence of in-situ thermal sensors. For calculating the aging rate, they utilized CALIPER [Bolchini et al., 2014]. To evaluate the proposed mapping schemes, they employed both synthetic (DAGGEN [Suter, 2013]) and realistic (video processing applications [Bertozzi et al., 2005]) task graphs. The LBRM proposal was compared with the state-of-the-art lifetime-constrained scheme named DSMR [Haghbayan et al., 2016], as well as the lifetime-agnostic runtime mapping called MAPPRO [Haghbayan et al., 2015]. Their experimental results revealed that, in comparison to the state-of-the-art lifetime-constrained mapping, the proposed mapping scheme enhances the throughput of manycore systems by an average of 26% for synthetic task graphs and by 20% for realistic task graphs, all the while maintaining lifetime reliability within the established constraints.

4.10 Run-time Resource Management for Multiple Aging Mechanisms

Haghbayan et al. [Haghbayan et al., 2020b] proposed a thermal-cycling-aware dynamic reliability management (DRM) approach for shared memory manycore systems running multi-threaded applications. This approach is novel because it incorporates Thermal Cycling (TC) awareness—unlike past approaches that focused solely on adhering to the power budget to prevent aging effects.

Haghbayan et al., in [Haghbayan et al., 2023], incorporate the Electromigration (EM) effect in their work. In this paper, the authors propose a lifetime reliability-aware runtime resource manager for conflicting requirements in both the short-term and long-term. The authors highlight that resource assignment decisions can have immediate effects on performance and power consumption, but may not have an immediate impact on lifetime reliability, which changes slowly over time. They also argue that mitigating only one aging mechanism, such as EM, may negatively affect other mechanisms, such as TC.

The architecture, shown in Figure 4.9, integrates with the operating system running on top of the manycore and includes four main modules: the RA Mapping unit, RA Scheduling unit, RA Dynamic Power Manager (DPM) unit, and the new Reliability Analysis unit. The Reliability Analysis unit estimates the aging status of cores based on temperature measurements and a stochastic lifetime reliability model. The other modules, referred to as Reliability-Aware (RA) units, use this information along with performance and power consumption considerations to make resource allocation decisions.

Reliability Analysis unit estimates the reliability status of each core in the architecture at the current time w.r.t. the two considered aging mechanisms (EM and TC). This module receives the temperatures of each core as inputs, provided by temperature sensors

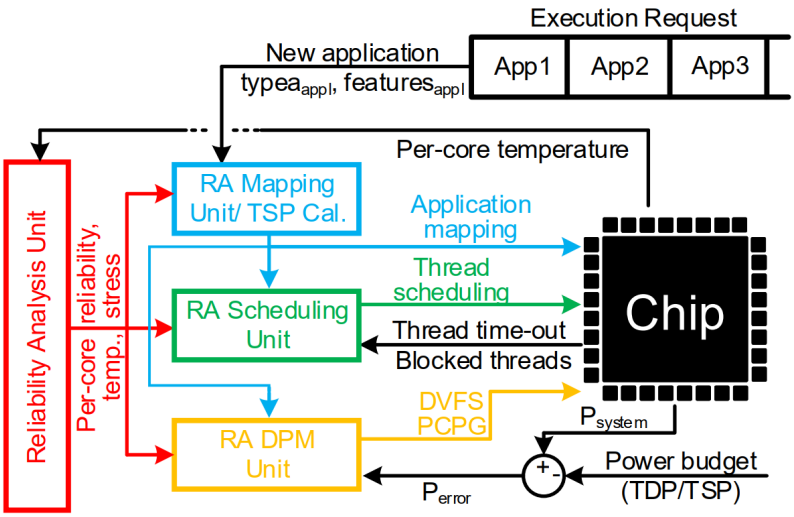


Figure 4.9: The run-time reliability-aware resource manager [Haghbayan et al., 2023].

at a fixed sample rate of 2 Hz. Reliability values are estimated once every long-term period (from 1 up to 2 hours).

The identifies a region of cores that are proximal in the grid to be used for the execution of a new application, associating this region to the incoming application, which is also known as a cluster. This is performed whenever there is a new application waiting in the request queue and some idle core is available. Otherwise, it is postponed until some running application terminates and releases occupied cores, making them available for an application waiting in the execution request list. The heuristic uses three affinity metrics to define the application cluster, (*i*) Stress Factor (SF), (*ii*) Vacancy Reliability Factor (VRF), and (*iii*) Memory Factor (MF).

The SF (Stress Factor) is a per-core metric that estimates the impact of aging on every individual core due to the execution of a newly incoming application. The authors calculate this value by performing a weighted sum that considers each aging mechanism. This calculation is based on the steady-state temperature estimation for the application that is about to execute on the system. To obtain the temperature estimation at runtime, the authors use a state-of-the-art technique [Zhang et al., 2018], a multi-layer perceptron (MLP). The MLP takes into account the current system map and the application's features, such as the number of integer and floating-point instructions and the number of accesses and misses in L1/L2 caches, among other characteristics known in advance. The VRF measures how evenly distributed the idle cores are within a candidate cluster; a lower dispersion signifies a preferable cluster configuration. Lastly, the MF is a metric designed to place memory-intensive applications close to the memory access routers. Less memory-intensive applications can be located farther away from these routers.

Experimental results demonstrate that the proposed approach improves the average Mean Time To Failure (MTTF) by at least 17% for EM and 20% for TC while maintaining the same performance level and ensuring adherence to the power budget.

4.11 Hot-Trim

Zhang et al. [Zhang et al., 2023] present a DRM and DTM framework, Hot-Trim, specifically designed for multicore processors. The framework aims to enhance thermal performance and reliability by considering hotspots developed at runtime due to task workload. The authors claim that traditional DTM techniques, which rely on on-chip thermal sensors, may not fully capture the true distribution of hotspots, leading to less-than-optimal resource management decisions. Hot-Trim addresses this gap by employing a machine learning-based model to map and predict the occurrence of hotspots in real-time, allowing better task allocation and migration decisions using the trained model.

The paper takes into consideration three VLSI wear-out effects: Electromigration (EM), Negative Bias Temperature Instability (NBTI), and Hot Carrier Injection (HCI). Authors consider these mechanisms to be central as they affect the overall lifetime of multicore processors. For accurate modeling and simulation of these effects, the authors use LifeSim [Rohith et al., 2018], an open-source tool.

Furthermore, the researchers employed a thermography system to measure the temperature of the processor cores, presented in Figure 4.10. This allows them to collect spatially precise thermal data that, when compared to sensing temperatures, reveals that sensor-provided temperatures can significantly differ from the hottest points within cores.

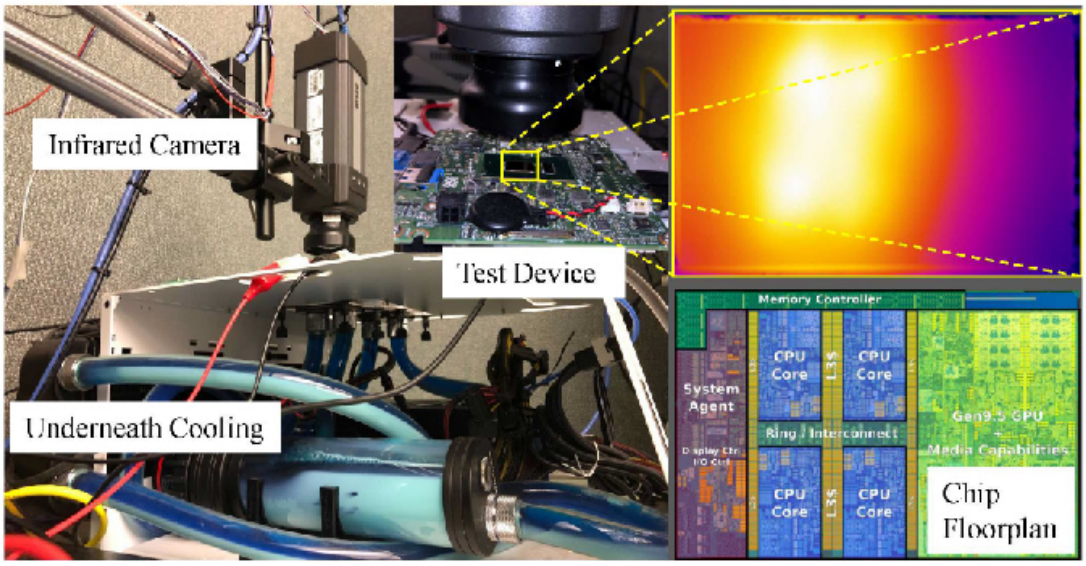


Figure 4.10: Infrared thermography system [Zhang et al., 2023].

The proposed Hot-Trim framework has two key components: a detector model for within-core hotspot prediction and a management controller that makes task allocation. The detector model uses metrics from Intel's Performance Counter Monitor (IPCM) and the thermography data to train the multi-layer perceptron (MLP) neural network to predict the power dissipation considering the monitoring data. To perform task allocation, Hot-Trim introduces a heuristic algorithm that distributes tasks to preserve core reliability and manage temperature. Tasks that induce higher power peaks at hotspots have the highest priority and are allocated to cores to minimize wear-out effects.

The authors validated Hot-Trim efficacy on an Intel Core i7 quad-core processor, with benchmark suites PARSEC-3.0 and SPLASH-2 driving the analysis. Compared to conventional methods, such as Linux's default scheduler and temperature-based task mapping, Hot-Trim can lower temperatures and enhance the reliability of the processor.

4.12 Lightweight Temperature Model

Castilhos et al. [Castilhos et al., 2016] propose a lightweight, software-based runtime temperature model for manycore systems. The model simplifies the HotSpot [Huang et al., 2006] model, enabling it to run on an embedded processor while aiming to minimize performance overhead. This model is designed for dynamic temperature management in manycore systems during runtime, prioritizing execution time, and accepting some loss of precision. The authors implemented and tested the model on a NoC based system providing cycle-accurate RTL description and featuring integrated power monitoring [Martins et al., 2014b].

The thermal characterization of the target platform comes from the HotSpot RC model. In this work, the temperature calibration process makes *four* key assumptions to obtain a simplified model. The calibration process begins with observing the heat flow when maximum power is applied to a PE. Figure 4.11 shows the measured temperature changes over time in a PE and its neighboring units. From the observed heat flow, researchers conclude: (*i*) a processor's thermal influence primarily affects its immediate lateral and diagonal neighbors, and (*ii*) thermal inertia is constrained within a 100 ms timeframe (20-time windows of 5 ms each). They additionally note that (*iii*) temperature and power have a linear relationship, allowing for the discretization of power values into intervals and assigning a corresponding temperature to each. Finally, they make the assumption (*iv*) to use only integer values rather than floating-point numbers, which enables faster computations.

They discretize and store the transient temperature and power behavior of each application in a lookup table (LUT) for the target PE and its direct neighbors. To estimate a PE's temperature, they record all the power consumption values over a transient effect window (20 samples) and then query the thermal behavior LUTs to determine the impact of

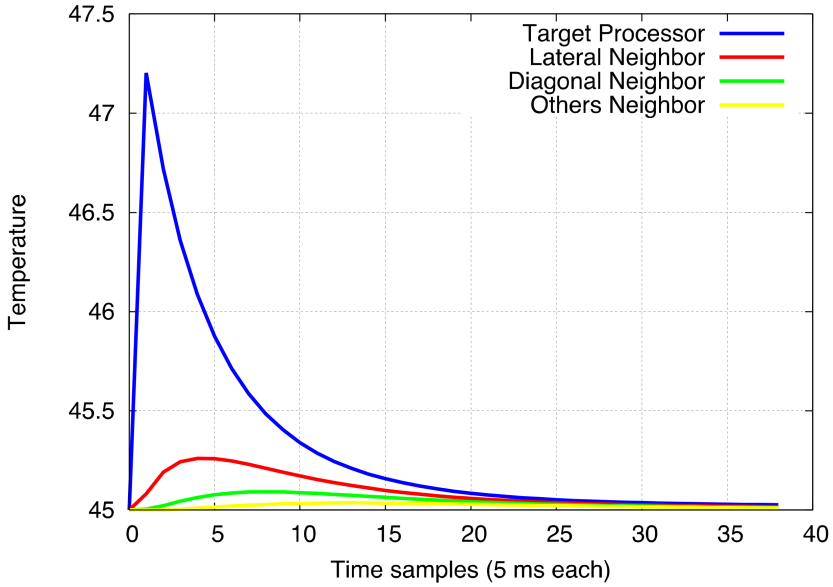


Figure 4.11: Effect of temperature (°C) over the time in a PE and its neighbors [Castilhos et al., 2016].

each sample. The author calculates the final estimated temperature for each PE by adding the transient effect of the target processor, which pertains to its power consumption, to the effects of all neighboring PEs, considering their power consumption.

The model produced is simple, requiring moderate memory, and capable of running fast enough, allowing real-time temperature monitoring in manycore systems. It has been tested using various configurations, up to 36 cores. Researchers simulated the proposed technique on a publicly available NoC-based MPSoC [Carara et al., 2009], incorporating a clock cycle-accurate model developed in SystemC. For a system with 36 PEs, the model can estimate temperatures in just 0.35ms when running at 100MHz. When compared with the HotSpot model, the proposed model has an average error of 3.52%, with the maximum error reaching 10%.

4.13 TEA: Temperature Estimation Accelerator

Silva et al. [Silva et al., 2019] proposed a fine-grained (core-level) temperature monitoring system for manycore systems, which includes a hardware accelerator to estimate the system's temperature. Each core's thermal estimation requires centralized computation due to the inherent data dependency caused by the influence of neighboring temperatures on the thermal models. Therefore, power monitoring samples from all cores must be collected simultaneously before estimating the temperature. The power monitoring method is hierarchical to minimize traffic on the NoC. Each core sends its power information to its respective manager, which then forwards it to the Temperature Estimation Accelerator (TEA).

The TEA computes the temperature of each core and sends the estimated temperature data to the global manager.

Figure 4.12 presents the TEA architecture, which contains six main modules. The Network Interface (NI) handles packet reception, identifies the sender, saves power data in the correct location, and assembles packets with the resultant temperature to send to the manager. The Temperature Registers (*Temp*) store the steady and current temperature of every thermal node, while the Power Registers (*Power*) store the received power samples. The Multiply-Accumulate (MAC) is an arithmetic module that performs the thermal estimation. The Matrices Memory stores the thermal model matrices. Additionally, TEA includes two finite-state machines that facilitate interaction with the NI and MAC.

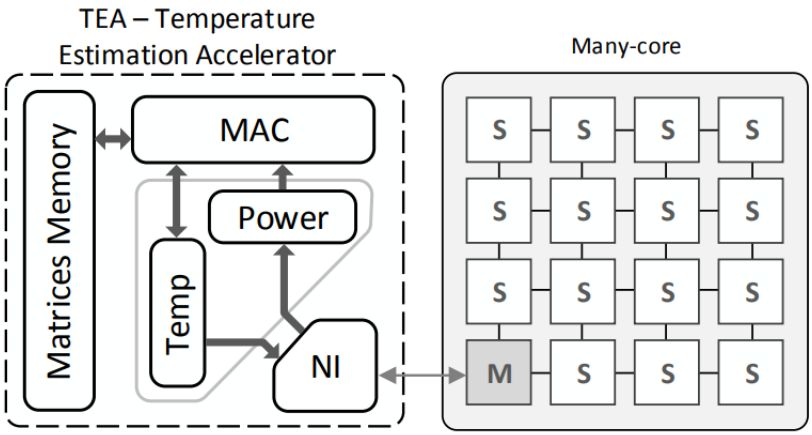


Figure 4.12: Overview of the TEA Architecture. NI stands for Network Interface, MAC for Multiply-Accumulate Unit, S for Slave, and M for Master [Silva et al., 2019].

The experiments were conducted on an in-house manycore system described with cycle-accurate precision in SystemC. The results were compared with HotSpot [Zhang et al., 2015], showing an average error of 0.02 degrees and a maximum error of 0.2 degrees. This technique paved the way for our work by enabling run-time accurate temperature monitoring.

In [Silva et al., 2020], the author proposes a dynamic thermal management (DTM) strategy based on a Proportional-Integral-Derivative (PID) control loop. This PID control loop uses temperature measurements provided by TEA. The author suggests three thermal-aware policies leveraging the PID: (*i*) application admission, (*ii*) task mapping, and (*iii*) task migration. These policies aim to prevent hotspots during application execution. The author evaluates their approach against patterned mapping [Yang et al., 2017b, Liu et al., 2018] and MORM [Martins et al., 2019], showing that their proposed heuristics achieve a reduction of up to 6.8% in the peak system temperature for scenarios involving high workloads—a notable gain in managing temperature challenges.

4.14 State of the Art Discussion

Most of the works reviewed propose strategies to mitigate power density issues in manycore systems. Five of these studies [Pagani et al., 2017, Yang et al., 2017a, Yang et al., 2017b, Liu et al., 2018, Mohammed et al., 2020] introduce architectures that employ a dark silicon patterning approach. This technique involves mapping tasks in a chessboard pattern, which enhances heat transfer between dark (inactive) and active cores. Additionally, these studies describe a communication strategy that helps minimize the impact of spread mapping on network latency. However, these works depend on complex temperature predictions calculated by MatEx to make real-time mapping decisions. This dependency may slow down the scheduler by tens of milliseconds. While the patterning approach can lead to system underutilization, the results presented in Chapter 6 of this Thesis demonstrate that our proposal successfully reduces the peak average temperature compared to the patterning approach.

Boosting techniques also appear in research concerning thermal management, as seen in [Pagani et al., 2015b]. Moreover, M-Oscilating [Sha et al., 2018b] is a solution that mitigates the problem of having a limited number of speed steps to control the power consumption of tasks. In this work, no boosting technique is applied because our method inherently considers the highest voltage-frequency (VF) combination. This approach enhances the application performance, and our mapping and migration ensure a uniform thermal distribution.

Control theory, as presented by [Silva et al., 2020], offers a viable approach to DTM. The PID control loop allows the system manager to take into account the current temperature, historical temperature, and temperature trends of each PE. By incorporating these elements into the heuristic for task allocation/migration, the system manager can reduce peak temperatures and, consequently, enhance system reliability, according to the authors' claims. However, the main drawback of this approach is that it requires continuous computation of a PID score for each PE at runtime, thereby compromising the scalability of the technique.

Several related studies suggest the use of dynamic reliability management (DRM) [Wang et al., 2018, Rathore et al., 2019b, Liu et al., 2019, Rathore et al., 2021, Haghbayan et al., 2020b, Haghbayan et al., 2023, Zhang et al., 2023]. DRM strategies employ the expected lifetime, derived from a theoretical model, as the primary metric for system management. Some DRM research accounts for thermal cycling as a controllable metric [Haghbayan et al., 2020b], while others focus on electromigration [Kim et al., 2017], and others also consider the time-dependent dielectric breakdown [Liu et al., 2019, Haghbayan et al., 2023] or on negative bias temperature instability [Rathore et al., 2019b, Zhang et al.,

2023]. Only one work accounts for process variation [Rathore, 2020] and one for hot carry injection [Zhang et al., 2023].

It is noteworthy that [Liu et al., 2019] recommended an offline mixed-integer linear programming (MILP) method, and [Haghbayan et al., 2020b, Haghbayan et al., 2023] claim for a more conventional heuristic approach to address thermal cycling issues. On the other hand, three works have introduced learning techniques for system management [Kim et al., 2017, Rathore et al., 2019b, Zhang et al., 2023]. Zhang et al. propose the use of an MLP neural network to predict the hotspot formation due to task consumption of resources. Rathore et al. propose a Q-learning approach with a fixed search space; her heuristic selects if the application will be mapped into fast or slow cores in a heterogeneous system to reduce the aging without penalizing the performance. The contact with those authors is the main source of inspiration for the FLEA heuristic, presented in Chapter 5.

Other related works do not focus on DRM or DTM proposals [Castilhos et al., 2016, Silva et al., 2021], which have collaborated for this study. Castilhos et al. [Castilhos et al., 2016] introduce a lightweight temperature estimation model that is also used in application mapping. However, this model is not scalable because its execution time is $O(n^2)$, where n represents the number of Processing Elements (PEs). Within the premises of the estimation model, Castilhos et al. further observe heat dissipation around a given PE; a topic that receives additional discussion during the FLEA motivation in Section 5.1.1. Additionally, Silva et al. propose a key component utilized in this study: the Thermal Estimation Accelerator (TEA), which estimates the temperature of each core at runtime.

Table 4.1 summarizes the works reviewed and classifies them according to the design goals for power, temperature, or reliability management. The first column lists the authors and their references. The second column details each work's design goal. The third column differentiates how the proposed architecture acquires temperature data. The fourth column itemizes the management techniques employed to achieve these objectives, identifying whether each technique utilizes DVFS, mapping, and migration. The fifth column lists the reliability-related effects considered in the work. Lastly, the sixth column outlines the experimental setup that the authors employed to simulate the architecture for their experiments.

Most studies use high-level architectural models with tools such as GEM5, Noculator, or Snipersim, which emulate the instruction set and communication structure of the target architecture, and McPAT, to evaluate processor power consumption. These high-level models are prevalent in computer architecture research; however, they make assumptions that can limit their effectiveness in power management strategies. Notably, these models do not account for the variable power consumption behavior of each task, even when operating at the same voltage and frequency. This oversight can lead to significant errors in power prediction, as noted by [Xi et al., 2015]. Additionally, the accuracy of power consumption data related to processors and NoC is a concern with these models. To mitigate these ac-

Table 4.1: State-of-art summary (DVFS: includes power- and clock-gating techniques).

Autor	Design Goals	Temperature		Actuation			Aging Effects	Modeling
		Sensor	Estimation	DVFS	Mapping	Migration		
Lei Yang 1 [Yang et al., 2017a] [Yang et al., 2017b]	Peak Temperature Communication Latency		X	X	X	X	-	McPAT(Power) GEM5(Architecture) MatEx(Temperature)
Weichen Liu [Liu et al., 2018]	Peak Temperature Communication Latency Communication Contention		X		X		-	McPAT(Power) GEM5(Architecture) MatEx(Temperature)
Santiago Pagani [Pagani et al., 2017] [Pagani et al., 2015b]	Peak Temperature Power Bugdet		X	X	X		-	Custom Thermal Model McPAT(Power) GEM5(Architecture) HotSpot(Temperature)
Shi Sha [Sha et al., 2018a]	Temperature Constraint Performance		X	X	X		-	Custom Thermal Model McPAT(Power) HotSpot(Temperature)
Mengquan Li [Li et al., 2015] [Li et al., 2018]	Temperature Constraint		X	X	X		-	Custom Thermal Model McPAT(Power) HotSpot(Evaluation)
Vijeta Rathore [Rathore et al., 2016b] [Rathore et al., 2018] [Rathore et al., 2019b] [Rathore et al., 2021]	Lifetime reliability Power Budget Performance	X		X	X	X	Process Variation NBTI EM	McPAT(Power) Snipersim(Architecture) HotSpot(Temperature)
Taeyoung Kim [Kim et al., 2017]	Lifetime reliability Energy	X		X			EM	McPAT(Power) Snipersim(Architecture) HotSpot(Temperature)
Liang Wang [Wang et al., 2018]	Lifetime reliability Performance	X			X		EM	McPAT(Power) CALIPER(Aging) HotSpot(Temperature)
Mohammad Haghbayan [Haghbayan et al., 2020b] [Haghbayan et al., 2023]	Lifetime reliability Performance		X	X	X	X	EM TC	Noculator(Architecture) McPAT(Power) Multi-layer Perception(Temperature)
Jinwei Zhang [Zhang et al., 2023]	Temperature Management Lifetime Reliability	ML Model			X	X	EM NBTI HCI	Real System IPCM(Power) Thermography LifeSim(Reliability)
Guilherme Castilhos [Castilhos et al., 2016]	Lightweight Software for Temperature Estimation		X		X		-	Custom Thermal Model Cycle-accurate simulation Low level power analysis HotSpot(Evaluation)
Alzemiro da Silva [Silva et al., 2019] [Silva et al., 2020]	Temperature Monitoring Thermal Management		X	X	X	X	-	Cycle-accurate simulation Low level power analysis
Proposal	Temperature Constraint Performance Lifetime reliability		X	X	X	X	EM SM TDDB TC NBTI	Abstract simulation (OVP) Long time simulation Accelerator(Temperature) MatEx(Reference Model)

curacy concerns, our platform was developed based on Memphis [Ruaro et al., 2019], an in-house manycore platform at RTL-level, allowing for the characterization of the router and instructions energy consumption.

In terms of obtaining real-time temperature data, most DTM research relies on some form of temperature estimation. At the same time, studies focusing on DRM tend to assume that temperature is available through in-situ sensors. We observe two exceptions; firstly, [Zhang et al., 2023] employs a thermography system capable of precisely acquiring temperature from the die. And in [Haghbayan et al., 2023], that employs a multi-layer per-

ception (MLP) solution to estimate the temperature of PEs in software. The major issue with temperature estimation is that it is often not scalable. Even the MLP solution has scalability limitations, with an estimated overhead of 0.57 ms for each prediction, resulting in a total delay of 344.1 ms to perform the 600 predictions necessary to allocate a single application, as reported in the example scenario [Zhang et al., 2018]. This constant thermal status monitoring, which plays a crucial role in system management, is frequently overlooked in related literature. As will be discussed in Chapter 5 and shown in Chapter 6, our proposal, once trained, can perform mappings without considering temperature, showing similar results to management techniques that have temperature information. However, temperature estimation is still necessary to enable migrations.

After reviewing the current state-of-the-art for DTMs and DRMs, we identified the following gaps in the existing literature:

- Although some studies suggest a patterning approach that could avoid system utilization, others depend on complex heuristics intended for design-time execution, which require deep foreknowledge of the workload. We found no studies offering a lightweight heuristic capable of efficiently handling workloads that occupy more than 50% of the system while interleaving active and inactive cores cannot seamlessly avoid hotspot creation.
- The works we reviewed specialize either in thermal management or in reliability management. None of the works propose thermal management with a lifetime reliability analysis. Furthermore, other studies consider on average only two aging effects.

The following Chapter introduces the heuristics we employ to manage applications within the system. These heuristics aim to maintain the system temperature within the prescribed limits and reduce chip degradation caused by various wear-out effects, thereby addressing the gaps identified earlier.

5. REINFORCEMENT LEARNING-BASED TASK MAPPING

This Chapter presents the *second and main original contribution of this Thesis*. The proposed mapping technique, **F**ailure In Time-aware **L**earning H**e**uristic for application **A**llocation (FLEA), is based on reinforcement learning (RL) and aims to enhance the lifetime reliability of manycores when executing dynamic workloads. FLEA employs a lookup table trained at design time to facilitate the task-to-core assignment, thereby addressing the critical aspect of lifetime reliability. The system manager uses FLEA during application mapping and task migration processes. It is worth noting that while other mapping strategies discussed in [Silva et al., 2020] and [Rathore et al., 2019a] can also handle dynamic workloads, they need continuous computation to update the PID scores or tables that guide the mapping strategy.

Part of this Chapter was published at:

Iaçanã Ianiski Weber, Vitor Balbinot Zanini and Fernando Gehm Moraes.
FLEA - FIT-Aware Heuristic for Application Allocation in manycores based on Q-Learning.
In Proceedings of the Brazilian Symposium on Computing Systems Engineering (SBESC) 2023.
http://dx.doi.org/10.1109/SBESC60926.2023.10324296

This Chapter is structured as follows. Section 5.1 provides insight into the research motivation, presenting the driving forces behind the development of the FLEA technique. Section 5.2 formally defines the research problem that FLEA seeks to address. In Section 5.3, the methodology employed to solve the identified problem is comprehensively described.

5.1 Motivation

Managing the manycore system temperature is crucial for enhancing system lifetime reliability [da Silva, 2021]. Several proposals have emerged in the literature that use machine learning techniques, particularly Reinforcement Learning (RL), for Dynamic Thermal Management (DTM). Das et al. [Das et al., 2014] employ Q-learning to build a Q-table that indexes system states representing all possible task arrangements within the system. However, a significant challenge arises due to the exponential growth of the action space as the system size increases, rendering their proposal unscalable. In contrast, Rathore et al. [Rathore et al., 2019b] present an alternative approach by creating a Q-table with a fixed action space. They classify applications based on power, temperature, and computation requirements, and RL is used to train the Q-table for optimal decision-making regarding whether an application should run on a high or low-frequency core to maximize system health. Nonetheless, Rathore's method relies on embedded sensors in each core to pro-

vide real-time data to the learning algorithm, potentially limiting scalability by requiring these hardware components and potentially hindering the scalability of the manycore system.

These works have inspired our research question: *Is it feasible to develop a lightweight and scalable reliability-aware mapping algorithm that operates independently of sensing data?* To answer this question, our solution has been driven by three fundamental considerations:

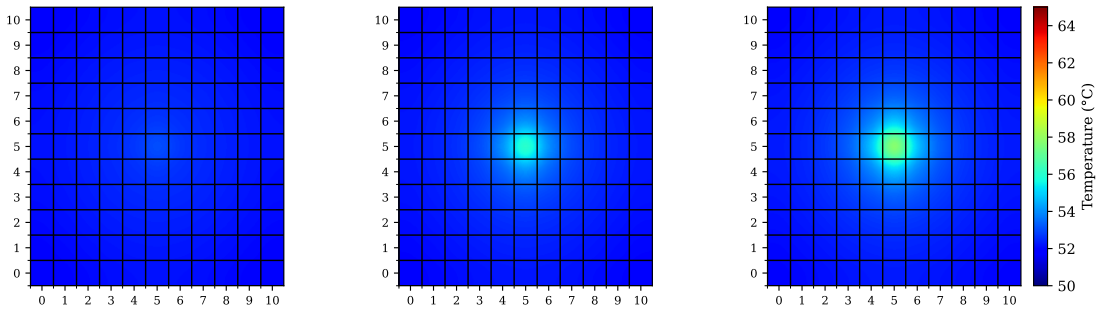
I. the heating behavior (Section 5.1.1) within a specific Processing Element (PE) can be attributed directly to two primary factors: (*i*) the amount of power dissipated by the PE, and (*ii*) the thermal conduction emanating from neighboring PEs.
II. the numerous wear-out effects (Section 5.1.2) are intricately linked to the system's temperature.
III. the use of reinforcement learning (Section 5.1.3) as a scalable approach to establish a connection between power consumption and task mapping, focusing on enhancing system reliability.

5.1.1 The Heating Behavior

The heat generated by a specific PE during the execution of a given task is a function of the power dissipation of that PE. We periodically estimate the energy expended by each PE, resulting in the power dissipation of the PE, which is determined by three distinct components: (*i*) the energy consumed during memory accesses; (*ii*) the energy expended on local communication with the NoC router, and the transmission of packets through the PE, and (*iii*) the energy consumed by the RISC-V core responsible for executing task instructions. Figure 5.1 illustrates the resulting temperature after 20 seconds of executing three tasks ($\tau a, \tau b$, and τc) mapped to the central PE ($x = 5$, $y = 5$) within an 11x11 manycore configuration. It is important to note that the relative average power consumption of each task follows the order: $P_{\tau(a)} < P_{\tau(b)} < P_{\tau(c)}$.

Figure 5.1(a) reveals a subtle temperature difference, wherein the central PE registers an approximate 1.5℃ increase in temperature compared to the surrounding PEs. This temperature distinction becomes more relevant in Figure 5.1(b), where the central PE exhibits an average temperature elevated by 5℃ relative to its counterparts. Finally, in Figure 5.1(c), the central PE demonstrates an average temperature of 8℃ higher than the surrounding PEs. Those scenarios demonstrate that the temperature is directly affected by the task power consumption being executed in the PE.

It is essential to consider that the heat generated by a PE during task execution dissipates in three dimensions. This thermal behavior is captured by Equation 5.1, which



(a) (b) (c)

Figure 5.1: Thermal shots of an 11x11 manycore system. Each thermal map corresponds to the execution of τa (a), τb (b) and τc (c). The colors indicate the temperature of the PEs, with warmer colors corresponding to higher temperatures.

characterizes the heat dissipation of a homogeneous body emitting thermal energy [Lienhard, 2006].

$$\frac{1}{\alpha}\frac{\partial u}{\partial t} = \left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2}\right) + \frac{1}{k}q \tag{5.1}$$

where:

- $u = u(x, y, z, t)$ is the temperature as a function of space and time;
- $\frac{\partial u}{\partial t}$ is the rate of change of temperature at a point over time;
- $\frac{\partial^2 u}{\partial x^2}$, $\frac{\partial^2 u}{\partial y^2}$, and $\frac{\partial^2 u}{\partial z^2}$ are the second spatial derivatives (thermal conductions) of temperature in the x, y, and z directions respectively;
- $\alpha \equiv \frac{k}{c_p \rho}$ is the thermal diffusivity, which is a material-specific quantity depending on the thermal conductivity k, the specific heat capacity c_p, and the mass density ρ;
- $q = q(x, y, z, t)$ is the heat generation function for each point in the space and time. This value is positive and nonzero when the PE is turned on and zero if the PE is in an off state.

The thermal model of a manycore system is not homogeneous and must consider different parts, such as the heat spreader, the heat sink, the interface material, the heat generated at each PE, the temperature exchange with the environment, and other elements that affect the temperature. Each of those parts is considered a homogeneous body, modeled by Equation 5.1. Huang et al. [Huang et al., 2006] propose a solution based on RC thermal networks to solve the first-order differential equation that models the iterations between the manycore system and its cooling solution. Pagani et al. [Pagani et al., 2015a] further enhanced this solution, introducing a fast polynomial-time algorithm to efficiently compute all transient temperatures from input power traces for any given time resolution without losing

accuracy. Finally, Alzemiro et al. [da Silva, 2021] proposed a hardware peripheral called Temperature Estimator Accelerator (TEA). TEA considers a fixed sampling rate for power traces, allowing it to solve the temperature problem for manycore systems at execution time with an estimated error as low as 1%. As discussed in Chapter 2, we can simulate the manycore and feed it with temperature estimations at runtime with the TEA embedded into our simulator.

Several works [Pagani et al., 2015a, Pagani et al., 2014, Li et al., 2015, Li et al., 2018, Silva et al., 2019, Yang et al., 2017a] highlight that when tasks are mapped to contiguous regions, it may generate a hotspot in the system. Figure 5.2 presents a 11x11 system executing one single task that is mapped in the central PE. The task is executing a simple Bubble sort algorithm, without any communication. Warmer colors mean higher temperatures. The central PE, that is executing the task, is running at 1GHz and every other PE is in an idle state, running at 100MHz. The thermal shot presents the temperature of each PE after executing for 20 seconds.

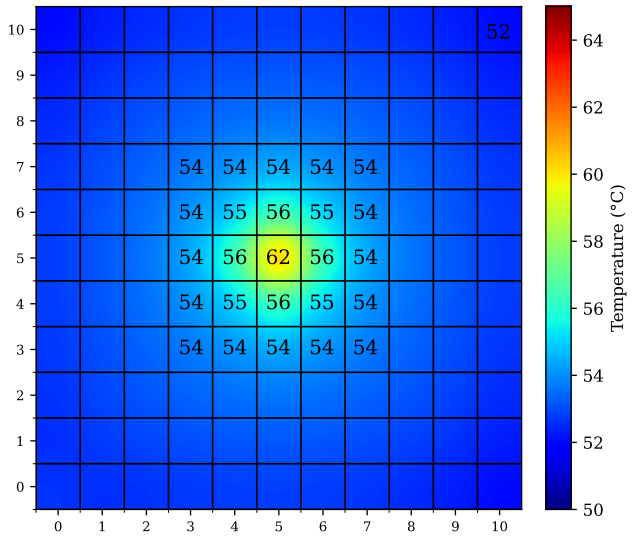
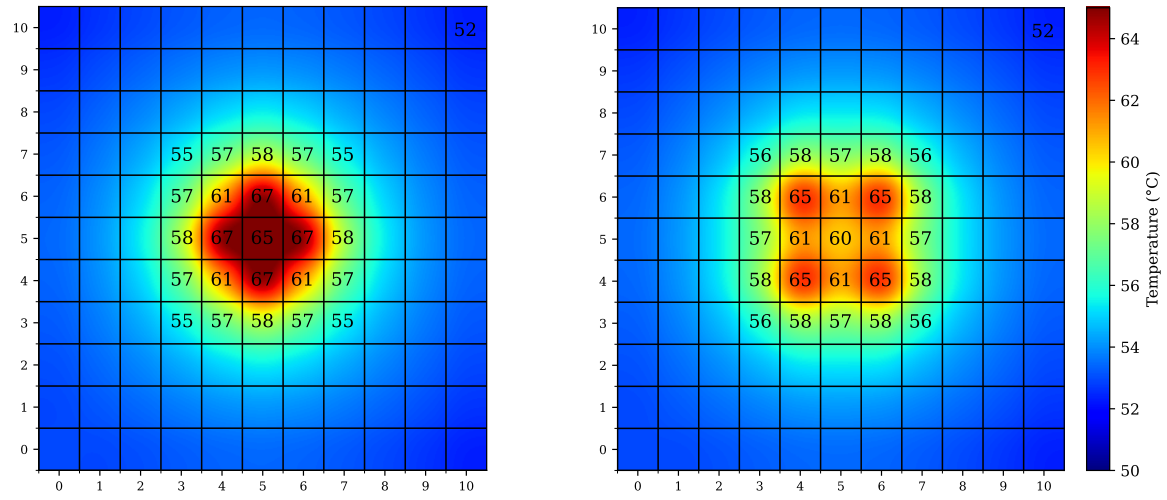


Figure 5.2: Thermal shot of an 11x11 manycore system executing a single task mapped to PE(5,5), after 20 seconds. The colors indicate the temperature of the PEs, with warmer colors meaning higher temperatures.

The heat generated by the central PE is transferred through thermal conduction to the adjacent PEs. This phenomenon becomes evident when we observe PEs far from the central PE, such as PE(10, 10), which has a temperature of 51.98℃. PEs closer to the central PE have higher temperatures, such as 56.26℃ for the four orthogonal neighbors. We observed a smaller thermal difference when comparing the temperatures of the PEs located diagonally to the central PE. This behavior is expected, as the distance between the midpoints of the diagonal PEs is greater than that between the midpoints of the orthogonal PEs.

Figure 5.3 provides a visual representation of the thermal effects caused by four tasks, using the Bubble sort task, which was previously used. These tasks are allocated



(a) (b)

Figure 5.3: Thermal shot of an 11x11 manycore system executing four tasks. In (a), the four tasks are orthogonal to central PE. In (b), the four tasks are diagonal to the central PE. The colors indicate the temperature of the PEs, with warmer colors meaning higher temperatures.

around the central PE in two distinct configurations: (a) $\{[6,5],[5,4],[5,6],[4,5]\}$ with tasks mapped orthogonally to the central PE, and (b) $\{[4,4],[4,6],[6,6],[6,4]\}$ with tasks mapped diagonally to the central PE. These two scenarios have a temperature difference of 5.25℃ in the central PE. In scenario (a), the central PE's temperature surpasses that depicted in Figure 5.2, even if the central PE is in an idle state. This difference arises due to the heat conducted from the four tasks surrounding the central PE. In scenario (a), these tasks form a contiguous area in contact with the central PE, promoting significant heat conduction. Conversely, in scenario (b), where the tasks are diagonally positioned relative to the central PE, the area of contact with the PE emitting heat is reduced, resulting in a comparatively lower temperature increase. *This observation motivates the exploration of techniques to optimize the topological distribution of tasks within the manycore architecture to enhance its thermal performance.*

5.1.2 The Heating Influence on Reliability

In Section 3.3.1, we introduced the five wear-out effects considered in this work. These effects encompass Electromigration (EM), Stress Migration (SM), Time-Dependent Dielectric Breakdown (TDDB), Thermal Cycling (TC), and Negative Bias Temperature Instability (NBTI). Each of these effects is characterized by a mean-time-to-failure (MTTF) equation in which temperature directly influences the wear-out rate. To show this correlation, we evaluated both scenarios presented in Figure 5.3 and assessed a PE operating in an idle

state without any load in its vicinity. By employing the aging models associated with these effects, we derived estimations for the failures in time (FIT) attributable to the respective aging mechanisms. The outcome of this analysis is presented graphically in Figure 5.4.

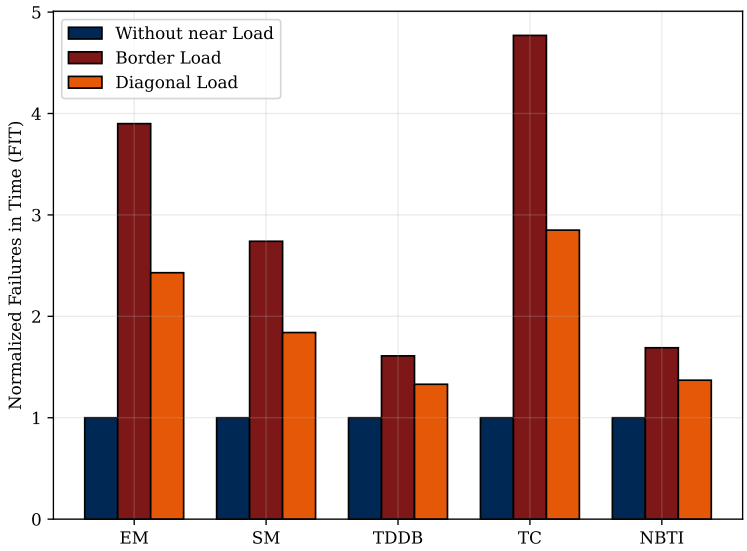


Figure 5.4: FIT estimation for the central PE ($x = 5$, $y = 5$), in idle state, in three different conditions: (*i*) without any task being executed near it; (*ii*) with tasks allocated in the PE border (Figure 5.3(a)); (*iii*) with tasks allocated diagonally to the PE (Figure 5.3(b)).

Figure 5.4 illustrates three distinct scenarios. The first scenario (*i*) depicts the manycore system operating in an idle state without executing tasks. Scenarios (*ii*) and (*iii*) correspond to the configurations outlined in Figure 5.3, with four tasks executing near the central PE, positioned either orthogonally or diagonally to it, respectively. The graph demonstrates that the increase in temperature directly impacts the number of failures experienced by the device. The effect of the temperature increase varies, with a 28% increment for TDDB and a 192% increase for TC when comparing scenarios (*ii*) and (*iii*). Even if the evaluated PE remains in an idle state, the heat generated by nearby PEs influences its reliability degradation. However, estimating the FIT at runtime may represent a significant cost to the manycore manager as every effect must be applied to each PE at every monitoring window. *Even with FIT information available, to our knowledge, there is no management technique available that considers the FIT value in its cost function.*

5.1.3 Scalability of Reinforcement Learning

Artificial neural networks (ANNs) have been explored in the context of manycore systems; however, the task mappings obtained through supervised learning have shown limited scalability. This limitation arises from the requirement of building a training dataset, which can be particularly challenging to acquire for manycore systems. In contrast, unsuper-

vised learning is not a suitable alternative in this context, as its primary application is data classification rather than task mapping. As a promising alternative, Q-learning emerges as a model-free reinforcement learning technique that does not hinge on the availability of a training dataset, thereby offering more flexibility in the learning process. Furthermore, Q-learning allows for optimizing reward functions, a feature well-suited for addressing manycore aging concerns. Nevertheless, the scalable implementation of *Q*-learning for manycore systems needs further exploration [Rathore, 2020].

Several RL-based resource management methods have been proposed in the literature, such as those discussed in Wang et al. [Wang et al., 2011], which present an RL-based dynamic power management technique of selective sleep and running state of the cores. Additionally, Lin et al. [Lin et al., 2014] proposed an RL-based technique for managing power in a hybrid electric vehicle. Among these, Das et al. [Das et al., 2014] and Rathore et al. [Rathore et al., 2019a] are the only ones that address the maximization of lifetime reliability and perform task mapping. Therefore, they are the most relevant to the proposed technique and thus were selected for motivational analysis.

Reinforcement Learning (RL) maintains a policy with the state-action pairs and an associated objective score for each pair. In Das et al. [Das et al., 2014], the authors consider the potential mappings as the action space, which can rapidly increase in size given its exponential relation to the number of tasks and cores in a manycore system. To mitigate this, Rathore et al. [Rathore et al., 2019a] consider the state space comprising a set of application categories. With this arrangement, the action space is given by two times the number of application categories, a constant figure.

Therefore, as the manycore system size increases, the solution proposed by Das et al. [Das et al., 2014] grows exponentially with the number of cores because it creates a new state-action pair for each possible mapping available. On the other hand, the solution proposed by Rathore et al. [Rathore et al., 2019a] remains constant, at a small value. It does not prune the action space, being smaller simply by construction. Unlike Das et al., it does not define the mappings as its actions; instead, it considers certain mapping heuristics as the actions.

Unlike both solutions, our proposal also does not generate all possible mappings, as proposed by Das et al. because the scale is not feasible. Furthermore, our proposal also differs from that presented by Rathore et al., as our Q-table will not say which heuristic should be applied. Instead, our idea is based on classifying the available PEs for mapping according to the task we want to map and the number of tasks allocated closer to the PE.

In this context, this Thesis combines hardware and software approaches to develop a dynamic management solution. This solution incorporates a policy table, trained using Q-learning, and a temperature monitoring system. The solution aims to efficiently manage the manycore systems with dynamic workloads to increase reliability by decreasing system aging.

5.2 Research Problem

As we discussed in Section 3.3, the Mean Time to Failure (MTTF) is an indicator of system reliability. There are several methods for computing the MTTF. In this Thesis, we employ two methods: (*i*) the sum of failure rates (SOFR), which considers the Failures In Time (FIT) to be exponentially distributed over time, detailed at page 65, and (*ii*) the lognormal distribution of FIT, detailed at page 66. Despite the differences in computation, both methods take into account the FIT of each PE. In either method, the MTTF is inversely proportional to the FIT of the PEs.

The **objective of our management technique** is to develop mapping and migration heuristics aimed at minimizing the FIT for each PE, thereby maximizing the system's lifetime reliability, denoted as $MTTF_{sys}$. These heuristics must also satisfy the temperature constraint $T_{x,y} \leq T_{threshold}\ \forall(\ x \in [1, X], y \in [1, Y])$, where $T_{threshold}$ is the threshold temperature defined by the system designer.

5.3 FLEA Deployment

This section introduces the original contribution of this Thesis, designated as **F**IT-aware **L**earning H**e**uristic for application **A**llocation (FLEA). The Q-learning algorithm is employed as the Reinforcement Learning (RL) method, selected for its simplicity and model-free characteristic, meaning it operates without requiring a model of the environment. Comparative analysis reveals that Q-learning outperforms other model-free RL algorithms like Monte Carlo control and SARSA in key metrics: it has the lowest Root Mean Square (RMS) error, highest speed of convergence, and largest average total reward return, according to Li et al. [Li, 2023].

FLEA adds a lookup table in the manager PE, named *Q*-table. The *Q*-learning populates the *Q*-table. The mapping and migration heuristics use the *Q*-table values (named *Q*-values).

The *Q*-table is a matrix with r rows and c columns used to select the location of a given task for the mapping and migration heuristics. Each row is addressed by the power consumed by the task, or "task power category" (TPC), according to Definition 2. Each column of the *Q*-table is indexed by a value defined by the thermal influence of the neighbors of a given PE, named PE bin-state (PBS), according to Definition 3. The *Q*-learning training phase determines the $r \times c$ *Q*-values.

When a task arrives for allocation or is selected for migration, during the inference phase, the manager PE traverses the manycore, calculating the PBS for each $PE_{x,y}$. The manager PE accesses the *Q*-table using the task TPC as row index and the $PE_{x,y}$ PBS as

column index. While it is traversing, it finds the available *PE* with the highest *Q*-value, which is selected as the task destination.

Definition 2. *Task Power Category* (**TPC**) - Tasks are grouped into discrete classes according to their average power profile. The power profile is obtained by running each application multiple times in the manycore without any other applications running in the system and without multitasking.

Figure 5.5 exemplifies the process of defining the TPC of each task. In the x-axis, we have each task available, named $\tau_{m,n}$ where m is the application number (in this example, the application set has seven applications), and n is the task number (in this example, each task has up to eight tasks). The y-axis presents the average power consumption measured in Watts (W). The graph shows the average power consumption of a set of 38 tasks.

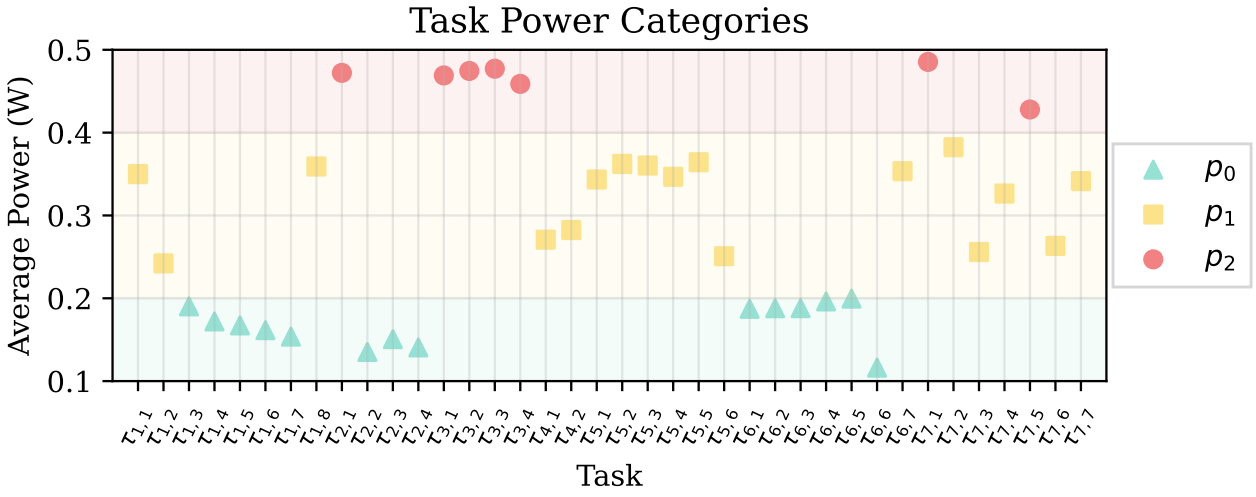


Figure 5.5: Average power consumption of tasks ($\tau_{m,n} \mid m \in [1,7], n \in [1, N_m]$). Tasks are classified into three categories, p_0 comprises tasks with average power consumption up to $0.2mW$, p_1 tasks range from $0.2mW$ up to $0.4mW$ and p_2 includes tasks that consume over $0.4mW$.

In our example, tasks are grouped by the average power consumption in three categories named p_0, p_1, and p_2. Tasks with an average power consumption of up to $0.2mW$ were assigned to p_0, tasks with average power consumption ranging from $0.2mW$ up to $0.4mW$ comprise the p_1 category, and tasks p_2 encompass all tasks that consume over $0.4mW$.

The **key idea** of grouping tasks is to generalize tasks with similar power consumption requirements into a single entity, reducing the model complexity [Krishnan et al., 2022]. For the sake of simplification, in the previous example, the TPC groups were defined without any particular method. However, during the experiments, we employed the *k*-means clustering [Hartigan and Wong, 1979] to define the TPC groups. This way, each task's TPC is defined as the group with the nearest power consumption mean.

Definition 3. *PE bin state* (PBS) – is an index in the *bin state table* that represents the distribution of tasks within the sphere of influence around a given PE. The sphere of influence comprises the neighbor PEs surrounding a given PE.

The bin state table is defined at design time. It is constructed by generating the set of every possible distribution of task power categories (TPC) within the defined sphere of influence. Exemplifying, the construction of the bin state table considered in our experiments was done as follows: Firstly, the sphere of influence was defined to encompass only the PEs that border the $PE_{x,y}$, because they represent the dominant source of outside heat in the $PE_{x,y}$ (as observed in Section 5.1.1). Therefore, in our example, the sphere of influence of $PE_{x,y}$ considers the set: $\{PE_{x+1,y}$, $PE_{x,y+1}$, $PE_{x-1,y}$, and $PE_{x,y-1}\}$. Next, every possible arrangement of tasks executing within the sphere of influence is generated, discarding equivalent arrangements. Equivalent arrangements aggregate combinations of tasks that irradiate the same amount of heat to the PE, i.e. the heat influence of a task executing at north, south, east, or west is the same. One example of equivalent arrangements is $\{p_0,\lambda,\lambda,\lambda\}$, $\{\lambda,p_0,\lambda,\lambda\}$, $\{\lambda,\lambda,p_0,\lambda\}$, and $\{\lambda,\lambda,\lambda,p_0\}$, (where λ means a idle PE) therefore, in any of those situations the PBS will be considered the same. Finally, with a sphere of influence considering the four bordering PEs and with three TPCs, we have a total of 35 possible PBS, every one of which is listed in Table 5.1.

Table 5.1: Example of PE Bin State (PBS) Table

$\sum p0$	$\sum p1$	$\sum p2$	***PBS***	$\sum p0$	$\sum p1$	$\sum p2$	***PBS***
0	0	0	**0**	1	0	3	**18**
0	0	1	**1**	1	1	0	**19**
0	0	2	**2**	1	1	1	**20**
0	0	3	**3**	1	1	2	**21**
0	0	4	**4**	1	2	0	**22**
0	1	0	**5**	1	2	1	**23**
0	1	1	**6**	1	3	0	**24**
0	1	2	**7**	2	0	0	**25**
0	1	3	**8**	2	0	1	**26**
0	2	0	**9**	2	0	2	**27**
0	2	1	**10**	2	1	0	**28**
0	2	2	**11**	2	1	1	**29**
0	3	0	**12**	2	2	0	**30**
0	3	1	**13**	3	0	0	**31**
0	4	0	**14**	3	0	1	**32**
1	0	0	**15**	3	1	0	**33**
1	0	1	**16**	4	0	0	**34**
1	0	2	**17**	-	-	-	-

Table 5.1 presents the *PE bin state table*. The first three columns contain the sum of neighboring PEs for TPCs p_0, p_1, p_2. Given that each PE has a maximum of 4 neighbors per PE, the maximum sum of these three columns is equal to 4. The fourth column is the PE bin state index, or *PBS*.

Figure 5.6(a) presents a manycore with 25 cores arranged in a 5x5 mesh. Each PE is colored gray if it is idle (not executing any task) or with a color according to its TPC. Each PE contains its address (x, y) and current PBS. Consider $PE_{2,2}$ in Figure 5.6(a). The PBS of $PE_{2,2}$ is 10 because $\{\sum p_0, \sum p_1, \sum p_2\} = \{0,2,1\}$. In Figure 5.6(b), we may observe the effect in PBS after the allocation of a new task with TPC $p1$ at $PE_{1,2}$, changing the $PE_{2,2}$ PBS from 10 to 13 (now $\{\sum p_0, \sum p_1, \sum p_2\} = \{0,3,1\}$). Besides that we also observe changes in the PBS of $PE_{1,3}$ (19 $\rightarrow$ 22), $PE_{0,2}$ (5 $\rightarrow$ 9), and $PE_{1,1}$ (9 $\rightarrow$ 12).

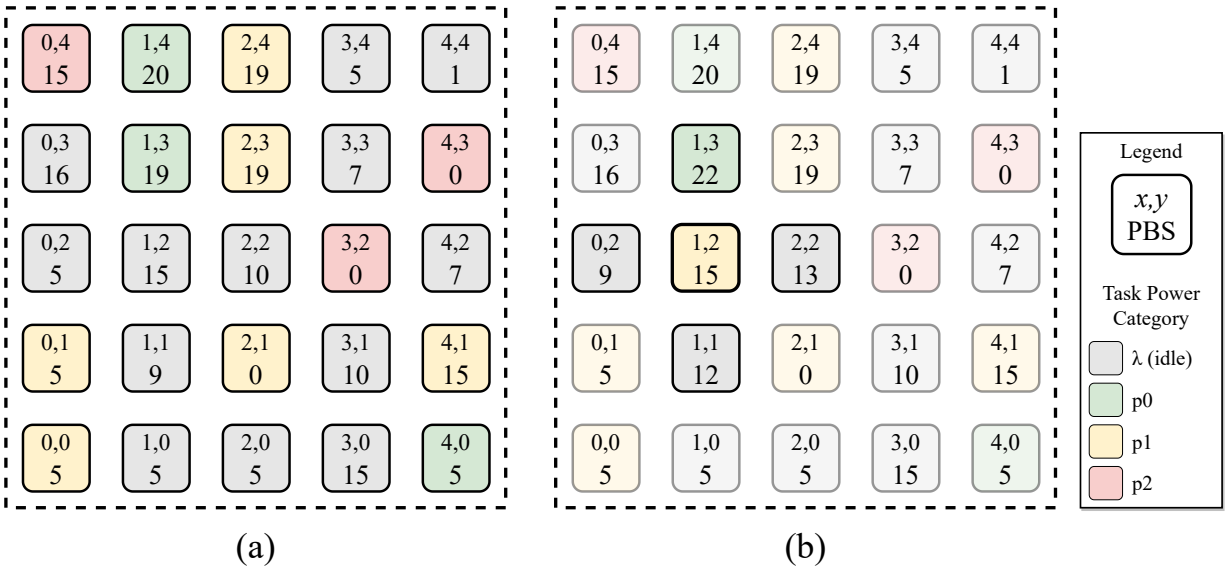


Figure 5.6: PE bin state (PBS) examples. The PE color indicates the task power category (TPC) that is executing. (a) presents the PBS of each PE; (b) highlights the PBS change after the admission of a new task $p1$ in the $PE_{1,2}$.

5.3.1 Training

During the training phase, the Q-learning algorithm updates its Q-values. Upon completing this phase, the Q-table contains the scores for the most favorable places to allocate tasks and is ready for the inference phase. Equation 5.2 corresponds to the update function used during the training phase.

$$Q^{new}(p,s) \leftarrow (1-\alpha)Q(p,s) + \alpha[r(\Delta FIT_{x,y}) + \gamma \max_{P} Q(P,s)] \tag{5.2}$$

where the new value ($Q^{new}(p,s)$) for the pair TPC (p) and PBS (s) is given by the sum of three factors:

- $(1-\alpha)Q(p,s)$: represents the current value for the pair (p,s);
- $\alpha r(\Delta FIT_{x,y})$: the reward is a function of the FIT variation in the last evaluated period (weighted by the learning rate α - the learning rate controls how fast the method modifies its estimates).

- $\alpha\gamma Q(p,s)$: the maximum value of Q for tasks with the same TPC ($p = P$) (weighted by the learning rate α and discount factor γ - the discount factor determines the importance of future rewards).

The learning rate (α) in the adopted approach operates in two phases. Initially, it is set to a high constant value ($\alpha = 0.1$) to facilitate rapid adaptations in the Q-table. Once the table reaches a predetermined number of iterations, α is linearly reduced to zero. It is important to note that each update to a specific score in the Q-table, indexed by a {TPC, PBS} pair, does not provide insights about other pairs, underscoring the importance of exploring a diverse set of mappings, also known as the trained population [Powell, 2022]. As α reaches zero, the algorithm ceases to learn, consolidating the Q-table and thereby concluding the training process. The need to gradually reduce α arises from the necessity for the Q-table to converge in stochastic environments [Even-Dar and Mansour, 2001]. The environment is inherently stochastic due to the exclusion of several variables from the Q-table model; identical actions may yield different outcomes. For instance, assigning two similar tasks to PEs with identical PBS values will produce different results if the initial temperatures of the PEs differ, such as $50°C$ and $60°C$.

The discount factor (γ) serves as a balancing mechanism to weigh the significance of immediate versus future rewards, thereby influencing the degree to which future outcomes affect present decisions. Generally, the setting of γ is context-dependent, tailored to the level of environmental uncertainty. In environments characterized by high uncertainty, a higher γ value is usually adopted, signifying a diminished influence of current actions on future outcomes [Ris-Ala, 2023]. In our training framework, we set γ at a constant value of 0.35.

The reward function, denoted as $r(\Delta FIT_{x,y})$, is presented in Figure 5.7. It is designed to boost the selection of PBSs that lower the PE FIT. Specifically, PBSs causing a decrease in FIT (x-axis) receive higher rewards (y-axis), promoting their corresponding Q-values. On the contrary, those who cause an increase in FIT get reduced rewards, maintaining their associated Q-values at lower levels.

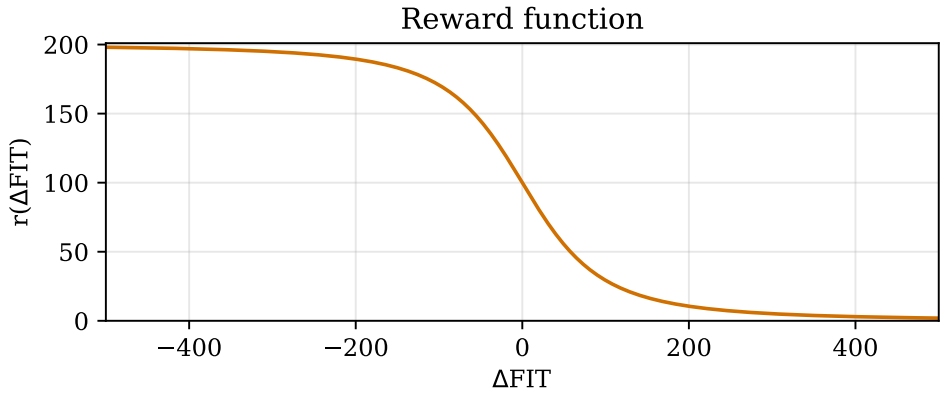


Figure 5.7: The reward function $r(\Delta FIT)$ used during the Q-table training.

Training occurs in the design phase using an epsilon-greedy learning policy [Powell, 2022], as depicted in Figure 5.8. Applications arrive at the system randomly, and the manager PE determines a mapping policy for each task. With an ε probability, the manager PE allocates the task to a random PE; with a $1-\varepsilon$ probability, the manager PE accesses the *Q*-table to select the optimal PE available. This involves computing the PBS for every available PE and then scanning the *Q*-table for the highest *Q*-value based on the TPC and PBS. Using the ε factor ensures comprehensive exploration of the state space. Once the location has been chosen, tasks are assigned to the designated PE. Every 1 ms, the platform evaluates each PE's temperature and FIT as outlined in Section 2. Concurrently, the manager initiates the *Q*-table update process. A PE is considered stable if its PBS remains unaltered over the preceding 50 ms, indicating that no tasks were placed within its sphere of influence. If stability persists over this duration, rewards are allocated based on FIT variation, leading to the computation of $Q^{new}(p,s)$, where *s* represents the PBS and *p* denotes the TPC executing in the PE (Equation 5.2).

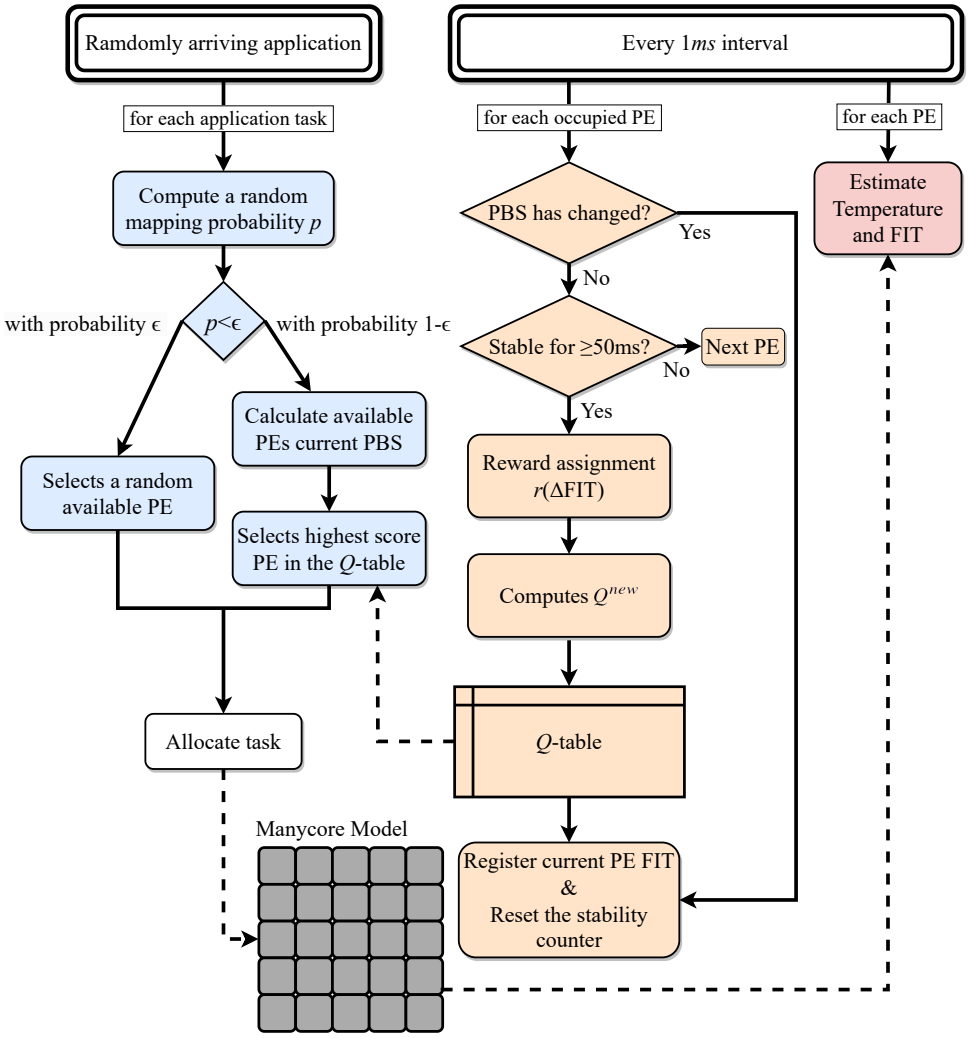


Figure 5.8: Training flow diagram. The diagram is divided into three main processes; the *blue* path presents the task-to-core selection process; the *orange* path is the *Q*-table update process; and the *red* presents the temperature and FIT estimation process.

The Chronos-V platform, as described in Chapter 2, can conduct *Q*-table training. However, to facilitate extensive simulations of thermal behavior over prolonged periods and to enable comprehensive experimentation with the training process, we developed an auxiliary platform in C++ that yields a simulation speed-up factor of up to 20,000 compared to Chronos-V (OVP platform). This specialized training platform[1] performs temperature and FIT estimations based on the average power consumption metrics obtained from the OVP platform. Importantly, the average power consumption used in these estimations is consistent with that used in the Task Power Category, as defined in Definition 2. An overview of the training platform's simulation loop is presented in Figure 5.9.

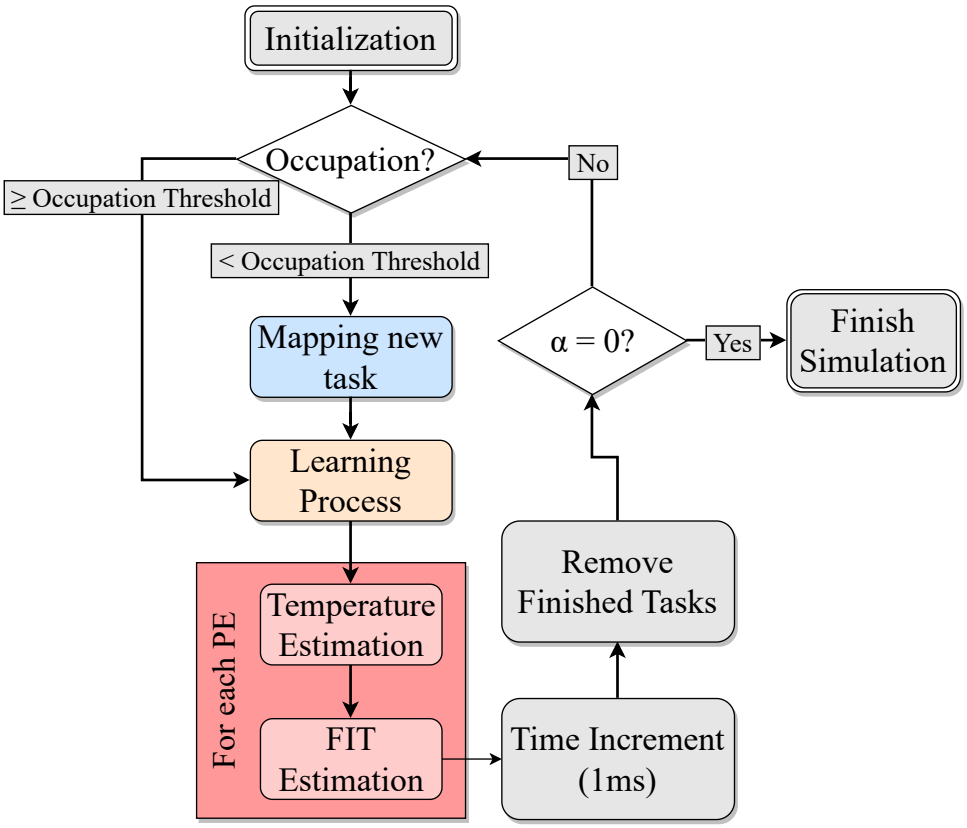


Figure 5.9: The training platform loop - colored tiles correspond to the paths with the same color in Figure 5.8.

The training platform is essentially a 2D array with an average power value at each PE's position. If the PE is occupied, the average power value of a task is placed in that position; if the PE is idle, the average power of in idle state PE is considered. After initialization, a verification is conducted to identify whether the system has the desired occupation. If not, a new task is mapped using the epsilon-greedy allocation (*blue* path depicted in Figure 5.8). Once the new task has been mapped or if the occupation has reached the threshold, the *Q*-learning processor begins, which follows the *orange* path indicated in Figure 5.8. Subsequently, the temperature and the FIT are calculated for each of the manycore PEs, and the simulation time is incremented by 1 ms. If the learning rate reaches zero ($alpha = 0$),

[1] Available at: https://github.com/iacanaw/Chronos_HighLevel

the simulation stops as the training is concluded. Otherwise, the loop restarts. After completing the training, the *Q*-table is saved, and we can import the *Q*-values into the system, which is ready to execute workloads using the learned *Q*-table in the mapping and migration heuristics.

Figure 5.10 depicts the evolution of rankings during the training phase, displaying 35 PE bin-states (PBS) for Task Power Category (TPC) 0. These PBS are ranked from the least favorable (rank 1) to the most favorable (rank 35). The figure includes 30 evenly spaced snapshots of these rankings to illustrate their progression throughout the training period. Notably, considerable shifts in rankings occur during the initial 20 snapshots. Subsequently, the ranks largely stabilize and converge towards fixed positions until the end of the training. This observed behavior correlates with the two-stage learning rate (α), which initiates its decrease around the twentieth snapshot.

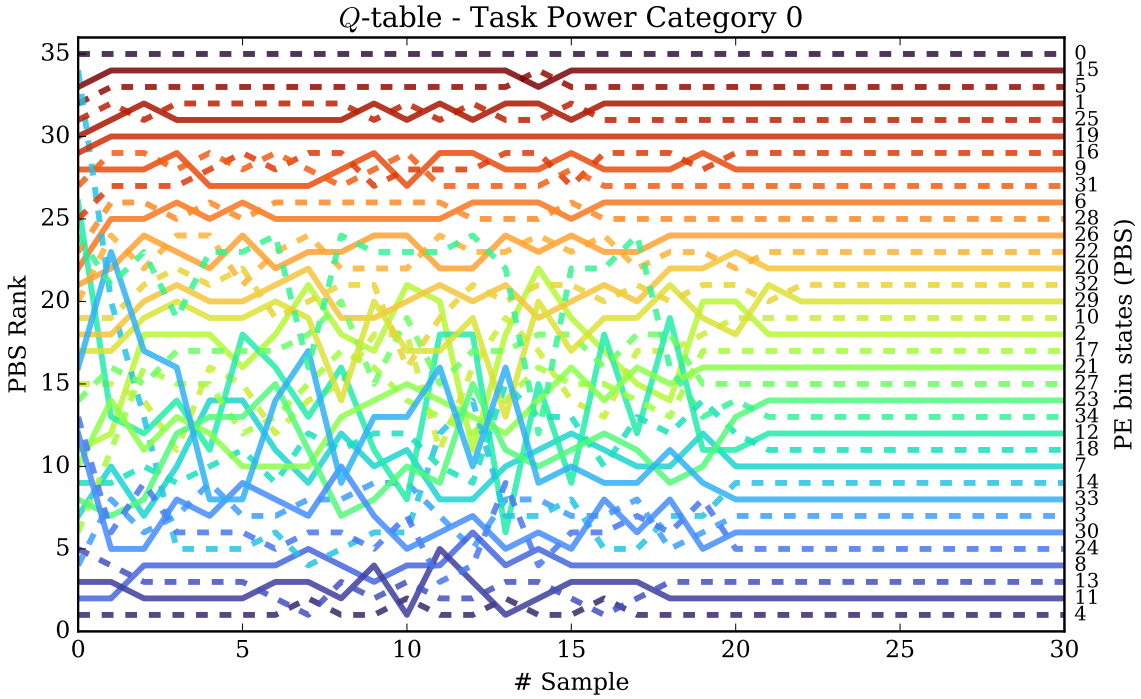


Figure 5.10: Training ranking evolution of each PBS in the *Q*-table indexed by TPC 0.

5.4 Actuation Mechanisms and Decision Heuristics

The actuation mechanisms are the means through which FLEA interfaces with the system. Understanding the actuation mechanisms is crucial to understanding the decisions in the development behind the FLEA heuristics. It should be noted that these mechanisms are distributed across various levels of hierarchy in alignment with the hierarchical structure of the reference platform. The actuation mechanisms use protocols presented in Section 2.2.2.

5.4.1 Application Admission

Application admission is an actuation mechanism encoded in the manager PE software. Incoming applications reach the system through the Application Repository (App_{rep}) peripheral. The manager PE receives a service packet from the App_{rep} informing that an application needs to be allocated. This triggers the application admission process, which will search for a cluster with available resources for the incoming application. When resources are scarce, the selected cluster may encompass the entire system. If the manager PE can find a cluster, the application is admitted into the system.

The cluster search process seeks a continuous region with an amount of available PEs (i.e., with resources to run a task) at least equal to two times the number of application tasks. The rationale for this choice is to reduce the search space to select where the application will execute and at the same time, reduce the number of hops between communicating tasks. Mapping communicating tasks close to each other reduces spatial fragmentation, avoiding performance degradation due to increased hop count and traffic congestion [Li et al., 2019]. The cluster is not exclusively allocated to a single application, meaning other tasks may already be running on it. The selection of the cluster is based on which has the lowest temperature. As already detailed, the TEA provides this information to the manager PE. However, in a manycore system where this information is unavailable at run-time, the heuristic may select a cluster based on a different criterion, such as the number of tasks already running on the cluster. Algorithm 2 details the application admission procedure.

Algorithm 2 receives two inputs: (*i*) the application reference, denoted as `app`, and (*ii*) `PE_info`, which contains data related to the PEs. Initially, the algorithm computes the minimum cluster side size (line 1) to ensure a space at least twice the size needed for the application's tasks. It then assesses the availability of system resources to receive the incoming application. If there is no available space for the application, the candidate cluster size is set to zero (line 3), and the algorithm ends (line 37), indicating the inability to find a suitable cluster and, therefore, rejecting the application admission.

If the search window size exceeds any system dimension, the search ends, returning an infinite cluster side size, which means that the cluster encompasses the entire system (lines 7-9). The algorithm uses a sliding window search, initializing the candidate temperature to infinity (line 11). If the candidate temperature remains infinite after both inner loops, it indicates an absence of an appropriate cluster, prompting the search to increase the window side (line 34) and continue.

The algorithm's main loops (lines 12-13) identify the bottom-left corner of the search window. Before any window analysis, the temperature and available PEs are reset (lines 14-15). Subsequent nested loops (lines 16-17) analyze each PE within the window, adding temperature data (line 18) and quantifying available PEs (lines 19-21). If the analyzed window

Algorithm 2: Application Admission

```
    Input: app, PE_info
    Output: cl_candidate
 1  window_side ←ceil(√(2app.n_tasks))
 2  if Σ_{x=0}^{DIM_X−1} Σ_{y=0}^{DIM_Y−1} PE[x][y].slots ≤ app.n_tasks then
 3  |  cl_candidate.side ← 0
 4  end
 5  else
 6  |  do
 7  |  |  if window_side ≥ DIM_X ∨ window_side ≥ DIM_Y then
 8  |  |  |  cl_candidate.side ← ∞
 9  |  |  |  break
10  |  |  end
11  |  |  cl_candidate.temp ← ∞
12  |  |  for x ← 0 to DIM_X − window_side do
13  |  |  |  for y ← 0 to DIM_Y − window_side do
14  |  |  |  |  window_temp ← 0
15  |  |  |  |  window_slots ← 0
16  |  |  |  |  for i ← x to window_side do
17  |  |  |  |  |  for j ← y to window_side do
18  |  |  |  |  |  |  window_temp+ = PE_info[i][j].temp
19  |  |  |  |  |  |  if PE[i][j].is_available() then
20  |  |  |  |  |  |  |  window_slots++
21  |  |  |  |  |  |  end
22  |  |  |  |  |  end
23  |  |  |  |  end
24  |  |  |  |  if window_slots ≥ 2app.n_tasks then
25  |  |  |  |  |  if window_temp < cl_candidate.temp then
26  |  |  |  |  |  |  cl_candidate.temp ← window_temp
27  |  |  |  |  |  |  cl_candidate.x ← x
28  |  |  |  |  |  |  cl_candidate.y ← y
29  |  |  |  |  |  |  cl_candidate.side ← window_side
30  |  |  |  |  |  end
31  |  |  |  |  end
32  |  |  |  end
33  |  |  end
34  |  |  window_side++
35  |  while cl_candidate.temp = ∞
36  end
37  return cl_candidate
```

meets the spatial requirements and has a lower temperature than any previously identified candidates, it becomes the new candidate cluster (lines 24-31).

After examining all windows, the window size is incremented (line 34) to execute the next iteration, as no window could receive the required number of tasks. However, if the candidate cluster's temperature is finite, it indicates that a suitable cluster has been identified, thereby terminating the search (line 35) and returning the located cluster (line 37).

It should be emphasized that the decision criterion for selecting clusters in the algorithm was the cluster temperature. However, when temperature data is unavailable to the system manager, other alternatives may be used for cluster formation. These alternatives may include the occupation of the cluster or the total period during which the PEs remained idle.

5.4.2 Task Mapping

The allocation of application tasks to the PEs within a cluster is called Task Mapping. The manager PE executes this actuation mechanism, using the trained policy table, *Q*-table. After the task mapping heuristic, the task allocation protocol starts, as previously described in Section Section 2.2.2.

The FLEA task mapping starts after the admission step of the application. The task mapping algorithm is repeated for each application task. Algorithm 3 presents the FLEA task mapping. The algorithm receives as input: (*i*) `task`; (*ii*) `q_table`; (*iii*) `app` and (*iv*) `PE_info`.

The mapping process starts by defining the highest score as zero (line 1). After that, the two loops will iterate over every position within the cluster (lines 2-3). If a given PE is available (line 4), its score is obtained by consulting the *Q*-table (line 5), indexed by the PE bin state (from Definition 3) and the task power category (from Definition 2). If the PE score exceeds the prior highest score (line 6), then the highest score and the target PE are updated (lines 7-8). If more than one PE presents the same highest *Q*-value (line 10), the manager PE selects the PE that minimizes the hop count between other application tasks (lines 11-22). After repeating this verification for each PE in the cluster, the algorithm returns the PE with the highest *Q*-table score.

5.4.3 Task Migration

The actuation mechanism of moving a task from a source PE to a target PE is defined as task migration. This mechanism is also executed by the manager PE and uses the trained *Q*-table. The migration protocol adopted in our platform was previously detailed in Section 2.2.2.

The task migration decision process is triggered at each monitoring window (1 ms), using the temperature estimation from TEA. The migration heuristic actively looks for a thermal violation considering a pre-defined thermal threshold T_{th}. Upon detecting a thermal violation, the manager PE identifies the PE with the highest temperature above T_{th}, selecting the task running on it for migration. The inputs for Task Migration (Algorithm 4) are: (*i*) `task`; (*ii*) `q_table`; (*iii*) `app`; (*iv*) `PE_info`; (*v*) `T_th` - temperature threshold.

Algorithm 3: Task Mapping

```
   Input: task, q_table, app, PE_info
   Output: target_PE
 1 high_score ← 0
 2 for x ← app.cluster.x to app.cluster.x + app.cluster.side do
 3     for y ← app.cluster.y to app.cluster.y + app.cluster.side do
 4         if PE_info[x][y].is_available() then
 5             score = q_table[PE_info[x][y].PBS][task.TPC]
 6             if high_score < score then
 7                 high_score ← score
 8                 target_PE ← {x, y}
 9             end
10             else if high_score = score then
11                 hop_count_A ← 0
12                 hop_count_B ← 0
13                 foreach allocated_task from app do
14                     hop_count_A += Δ(x, allocated_task.x)
15                     hop_count_A += Δ(y, allocated_task.y)
16                     hop_count_B += Δ(target_PE.x, allocated_task.x)
17                     hop_count_B += Δ(target_PE.y, allocated_task.y)
18                 end
19                 if hop_count_A < hop_count_B then
20                     high_score ← score
21                     target_PE ← {x, y}
22                 end
23             end
24         end
25     end
26 end
27 return target_PE
```

After selecting the task that has to migrate, the heuristic starts by creating a list (`available_PE`) with all the PEs that have a task slot available within the expanded application cluster (lines 1-6). The expanded application cluster (`e_cluster`) considers PEs that are in the cluster's outer border (e.g. a cluster starting at {3, 4} with side equal to 3 encompasses 9 PEs, its expanded cluster starts at {2, 3} and has side equal to 5, encompassing 25 PEs), which are considered to create additional migration opportunities. The PE list is sorted by temperature (line 9) and the algorithm starts to search for the PE with the highest score in the *Q*-table (lines 11-17). Similarly to the task mapping process, the score is obtained by consulting the *Q*-table (line 12), indexed by the PE bin state (from Definition 3) and the task power category (from Definition 2). The algorithm evaluates the 50% available PEs (line 11) looking for the one with the highest score. This method corresponds to making a trade-off between current temperature and *Q*-table score. If more than one PE presents the same highest score, naturally select the PE with the lowest temperature.

Algorithm 4: Task Migration

```
Input: task, q_table, app, PE_info
Output: target_PE
1  available_PE ← ∅
2  for x ← app.e_cluster.x to app.e_cluster.x + app.e_cluster.side do
3      for y ← app.e_cluster.y to app.e_cluster.y + app.e_cluster.side do
4          if PE_info[x][y].is_available() then
5              available_PE.add(PE_info[x][y])
6          end
7      end
8  end
9  available_PE.sort_by_temp()
10 high_score ← 0
11 for i ← 0 to available_PE.length/2 do
12     score = q_table[available_PE[i].PBS][task.TPC]
13     if high_score < score then
14         high_score ← score
15         target_PE ← {available_PE[i].x, available_PE[i].y}
16     end
17 end
18 return target_PE
```

Note that each migration implies an execution overhead for the application because the migration protocol requires multiple synchronization packets to transfer the task memory content to its new PE. Thus, a mapping heuristic that can minimize the total number of migrations is essential to optimize the application performance.

6. RESULTS

This Chapter presents the results of the proposed heuristics applied in the reference OVP architecture (Chapter 2). Section 6.1 outlines the experimental setup employed in the evaluations. Section 6.2 presents an improvement in the FLEA configuration - **FLEA+**. The core of the Chapter, Section 6.3, includes a detailed evaluation of the temperature behavior for the heuristics (Section 6.3.1), the hop count between communicating tasks (Section 6.3.2), reliability assessment (Section 6.3.3), and a scalability analysis (Section 6.3.4). Section 6.4 concludes this Chapter.

6.1 Experimental Setup

To evaluate the heuristics, we used a set of benchmarks available on the Memphis platform [Ruaro et al., 2019] and developed four synthetic applications during this work.

The applications adapted from the Memphis platform include:

1. Dijkstra (DIJ), with 7 tasks;
2. Dynamic Time Warping (DTW), with 6 tasks;
3. Synthetic producer/consumer application (PRODCONS), with 2 tasks.

We designed new applications as a set of pipeline tasks where each task performs a varying amount of computation. Figure 6.1 illustrates these applications:

1. Synthetic application α (ALPHA), with 4 tasks;
2. Synthetic application β (BETA), with 4 tasks;
3. Synthetic application γ (GAMMA), with 8 tasks;
4. Synthetic application double pipeline (DPIPE), with 7 tasks.

We created sixteen scenarios to evaluate the thermal and reliability behavior associated with different management techniques. Table 6.1 presents the scenarios. Each scenario has a **unique tag** for easy reference during this Chapter. We classified the scenarios according to three criteria:

- Manycore size: 64, and 196 PEs;
- Workloads: computation-intensive or a mix of computation- and communication-intensive tasks. Computation-intensive scenarios include mainly CPU-bound tasks, while mixed scenarios include an equal distribution of communication- and CPU-bound tasks;

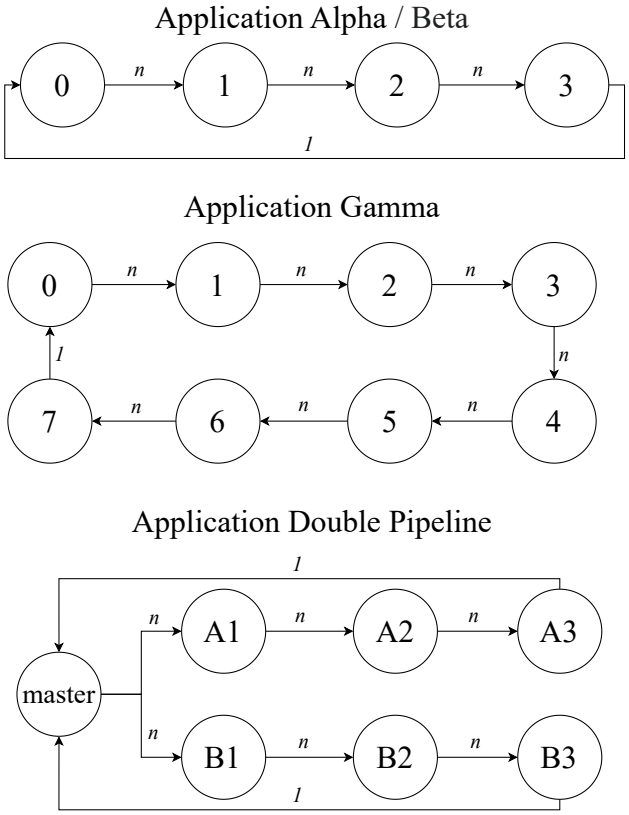


Figure 6.1: Task communication graphs for the new synthetic applications, where *n* represents the number of iterations in application execution.

- Occupancy: system occupancy refers to the percentage of PEs in use at any time during the scenario's execution, with levels ranging from 50% to 90%.

Table 6.1: Reliability and thermal evaluation scenarios.

	Scenario			Applications							
Manycore Size	**Workload**	**Occ. (%)**	**Tag**	**DIJ**	**DTW**	**PRODCONS**	**ALPHA**	**BETA**	**GAMMA**	**DPIPE**	**Total**
8x8	**MIXED1**	**50**	**A5**			4	3	3			**10**
		70	**A7**			6	5	4			**15**
		90	**A9**			7	7	4			**18**
	MIXED2	**50**	**B5**	2		5	2				**9**
		70	**B7**	2	1	7	2	1			**13**
		90	**B9**	3	1	9	3				**16**
	COMPUTATION	**50**	**C5**				8				**8**
		70	**C7**				9		1		**10**
		90	**C9**				10		2		**12**
14x14	**MIXED**	**50**	**D5**	5		5		1		7	**18**
		70	**D7**	6		10		3		9	**28**
		90	**D9**	6		13		8		11	**38**
	COMPUTATION	**50**	**E5**				24				**24**
		70	**E7**				34				**34**
		90	**E9**				44				**44**
	RANDOM	∼	**F**	3	3	3	3	3	3	3	**21**

We also proposed an additional scenario in which applications arrive randomly into the system. In the “random” scenario a maximum of three applications of the same type can execute at any time. In this scenario, the occupancy fluctuates over time due to applications having varying numbers of tasks.

These scenarios were executed in the Chronos-V platform (Chapter 2). We compare our management strategies with a Grouped mapping, a patterning [Liu et al., 2018] mapping, and a PID approach [Silva et al., 2020]. Grouped and patterning manage applications without considering any feedback from the system. PID approach considers the temperature in its cost function, using task migration dynamically.

6.2 FLEA+

Before comparing FLEA to other DTM and DRM strategies, this Section details a modification to the FLEA heuristic and evaluates its impact – **FLEA+**. We increased the number of PE bin states (PBS) to capture additional optimization opportunities previously dismissed by the algorithm.

Originally, the selection process had an inherent bias towards PEs on the manycore edge. These edge PEs were more likely to be in a beneficial PBS due to their reduced number of neighboring PEs. The initial set of 35 states, detailed in Table 5.1, considers only the four directly adjacent PEs to the target PE. These adjacent PEs are essential as they most significantly influence the target PE’s temperature. However, this approach had limitations. For instance, a PE with all eight surrounding positions unused is a better candidate than one with four diagonally used positions. The original 35-PBS framework did not distinguish between such scenarios.

To consider this scenario, we expanded the number of PBS to reflect a wider range of PE configurations, including those with diagonal neighbors. This enhancement enables the heuristic to recognize different spatial arrangements of PEs, increasing the options for task allocation decisions.

Initially, our plan included accounting for all eight surrounding PEs, categorizing them into diagonal and lateral PEs. This modification would increase the number of possible states from 35 to 1,225. Such an increase, while comprehensive, would require significantly higher training times due to the complexity of randomly reaching many of these states.

Considering these constraints, we opted for a different approach, categorizing the PEs into three groups. This new strategy aims to balance the need for different spatial arrangements of PE configurations with training time and algorithmic complexity.

1. The first category addresses scenarios where two or more tasks with high power consumption (TPC 2) are diagonally placed in relation to the center PE.

2. The second category focuses on situations with two or more tasks with medium power consumption (TPC 1) located diagonally to the PE.
3. The third category considers the edge PEs of the manycore system.

Adding these three new categories into the PBS enhances the FLEA heuristic, resulting in 140 states. This total comprises the original 35 states and an additional 35 for the three newly introduced categories. This expansion enables the recognition of new spatial configurations surrounding each PE, thereby allowing the learning algorithm to optimize task allocations more effectively and accurately.

Chapter 5 detailed the methodology for the training process, which is conducted on a high-level training platform. It is important to note that introducing these changes in the PBS does not modify the training process. The training process still applies the same principles and mechanisms, remaining adaptable to these modifications.

6.2.1 FLEA+ Evaluation

This section compares the original FLEA heuristic, as detailed in Chapter 5, and the modified version, referred to as **FLEA+**, which incorporates the expanded PBS table. This comparison aims to assess the impact of the expanded PBS on the heuristic's performance and effectiveness in task allocation and system management. Both heuristics were evaluated using the scenarios presented in Table 6.1.

Performance was assessed using three metrics: (*i*) Mean Time to Failure (MTTF), (*ii*) average temperature, and (*iii*) average peak temperature. MTTF is a crucial metric indicating the expected duration before a potential failure, which is particularly relevant as both FLEA and FLEA+ were optimized for MTTF maximization through reinforcement learning. The average temperature metric evaluates the overall thermal load of the system. In contrast, the average peak temperature metric reflects the highest temperatures observed, crucial for maintaining the system within safe operational thresholds.

Table 6.2 presents a detailed comparison of MTTF, average temperature, and average peak temperature for FLEA and FLEA+, with migration enabled.

Examining the MTTF data in the table, we observe marginal efficiency improvements in FLEA+ compared to the original FLEA heuristic. For instance, in the scenario labeled "8x8 Computation 50%", FLEA+ exhibits a 4.4% enhancement in MTTF relative to FLEA. Conversely, in the "14x14 Mixed 90%" scenario, FLEA+ underperforms by 1.4% compared to FLEA. On average, FLEA+ demonstrates a 1% higher MTTF across all evaluated scenarios than its predecessor, FLEA.

Concerning the average temperatures observed in the comparative study of FLEA and FLEA+, results reveal only minor variations. The most notable difference occurs in

Table 6.2: Comparison of MTTF between FLEA and FLEA+.

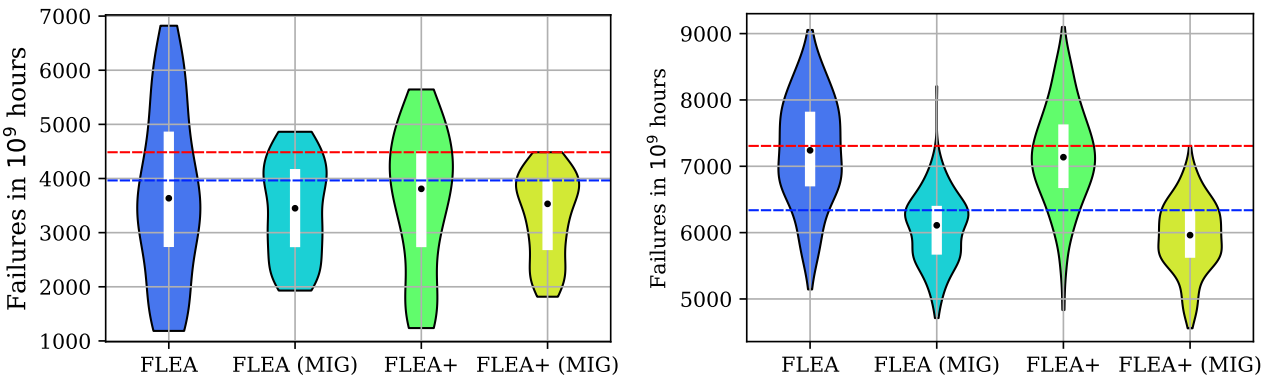
Workload	Occupancy (%)	Algorithm	MTTF (years)		Average Temp. (℃)		Avg. Peak Temp. (℃)	
			8x8	14x14	8x8	14x14	8x8	14x14
MIXED	50	FLEA	7.88	4.35	66.30	66.63	77.13	76.36
		FLEA+	7.92	4.33	66.31	66.91	75.96	76.03
	70	FLEA	5.24	3.20	73.69	72.06	84.31	85.09
		FLEA+	5.19	3.18	74.00	72.39	83.49	84.02
	90	FLEA	3.94	2.87	79.04	74.31	88.73	86.68
		FLEA+	3.93	2.83	79.06	74.47	88.60	85.72
COMPUTATION	50	FLEA	5.04	2.58	73.67	76.49	84.73	82.63
		FLEA+	5.26	2.59	72.89	76.37	83.72	82.44
	70	FLEA	4.30	1.67	77.42	84.18	86.51	95.04
		FLEA+	4.36	1.70	77.06	83.82	86.10	94.10
	90	FLEA	3.53	1.04	80.81	93.04	90.51	101.41
		FLEA+	3.40	1.07	81.67	92.51	90.68	101.07

the “8x8 Computation 50%” scenario, where FLEA+ maintains an average temperature that is 0.78℃ lower than that of the original FLEA heuristic. Conversely, in the “14x14 Mixed 70%” scenario, FLEA exhibits superior performance by keeping an average temperature of 0.33℃ cooler than FLEA+. Across all tested scenarios, the average temperature variation is minimal, generally within a narrow margin of less than 0.1℃ when comparing the two methodologies. Regarding peak temperatures, the variation stayed under 1℃, with an average difference of approximately 0.6℃.

Appendix B presents a violin-box plot for each scenario that compares the FIT of both FLEA and FLEA+ with and without migration. The graphical representation showcases the distribution of system FIT values, with each simulation depicted by a figure spanning from the minimum to the maximum FIT recorded. The width of the figure at various FIT levels indicates the density of PEs at those values, aiding in the identification of outliers. The white bar within each plot represents the interquartile range, stretching from the first to the third quartile. The median FIT value is marked by a black dot within this white bar, indicating that half of the PEs have FIT values below this point and half above it. Two scenarios from the appendix were chosen to present the differences between FLEA and FLEA+ and are depicted in Figure 6.2.

The violin-box plots contain blue and red dotted lines delimiting the region corresponding to the 25% of PEs with the highest FIT values in the FLEA+ (MIG) heuristic. Some observations can be drawn from these plots:

- The reduction in FIT values when migration (MIG) is employed highlights the relevance of this actuation mechanism. This demonstrates that enabling migration in the heuristics notably affects FIT values, thereby influencing the overall reliability of the system;
- In the “8x8 Computation 50%” (Figure 6.2(a)) scenario, it is observed that FLEA (MIG) has a greater number of PEs with FIT values exceeding the worst FIT in FLEA+ (MIG). This observation explains the 4.4% enhancement in MTTF observed with FLEA+ in



(a) 8x8 Computation 50%

(b) 14x14 Computation 70%

Figure 6.2: FIT comparison between FLEA and FLEA+. The blue dotted line is the FLEA+ with migration in the third quartile. The red dotted line is the FLEA+ with migration maximum FIT.

this scenario. Generally, FLEA+ shows a better performance in terms of FIT values compared to FLEA;

- In scenario “14x14 Computation 70%” (Figure 6.2(b)), we observe outliers in FLEA (MIG), which slightly impacts the MTTF.

6.2.2 FLEA+ Evaluation Conclusion

The modifications implemented in FLEA to enhance reliability management demonstrated that the heuristic, as initially conceived, manages MTTF accordingly. Regarding FLEA+, two significant costs are incurred due to the increased PBS size:

1. The increase in the Q-table is significant. The Q-table size has been increased from a 35-position, 8-bit vector to a configuration comprising 140 positions.
2. The computational cost associated with identifying the PBS for a PE has also increased. This increment considers 8 neighboring PEs instead of the previous 4. However, this increase in computational load occurs only once for each task allocation/migration in the system and solely for PEs within the task’s sphere of influence. Despite this increase, the computational cost remains low due to the evaluation involving 8 PEs instead of 4.

Results from simulations indicate minimal differences in (*i*) MTTF, (*ii*) average temperature, and (*iii*) average peak temperature between the original FLEA and FLEA+. However, a significant finding from the FIT plots reveals that the top 25% of PEs with the highest FITs in FLEA+ (MIG) consistently exhibit lower FITs than their counterparts in the original FLEA (MIG). Thus, we use FLEA+ in the next section, which compares our approach against other DTM and DRM heuristics.

6.3 Results Evaluation

This section presents the simulation results for various mapping strategies, including Grouped mapping, pattern mapping [Liu et al., 2018], PID mapping [Silva et al., 2020], and our proposed FLEA+ heuristic. We made simulations for all scenarios detailed in Section 6.1 and compiled a series of graphs and metrics for each. Throughout this section, we highlight specific metrics and compare them using graphical and tabular representations to support our analysis. For an extensive visual presentation of each simulation scenario, please refer to Appendix C. This appendix includes a thorough collection of graphical data, encompassing:

1. The temporal evolution of average and peak temperatures;
2. Heat map snapshots that illustrate the per-PE average temperature;
3. Violin-box plots offering a statistical view of the average temperatures;
4. Violin-box plots offering a statistical view of peak temperatures;
5. Violin-box plots offering a statistical view of FIT rates;
6. Comprehensive heat maps that plot per-PE FIT rates, attributable to various wear-out effects as well as the total FIT.

6.3.1 Temperature Evaluation

Figure 6.3 provides a heat map that depicts the thermal behavior of a 14x14 manycore system over time under various management algorithms. This Figure visually represents the average temperature of each PE within the manycore. The figure includes four snapshots, evenly distributed over the simulation period, to represent each evaluated management heuristic. In the manycore grid, each tile corresponds to a specific PE, with colors indicating the average temperature during the snapshot period, as denoted by the labels on the figure's left side. This graphical approach enables quick identification of thermal patterns and hotspots within the system.

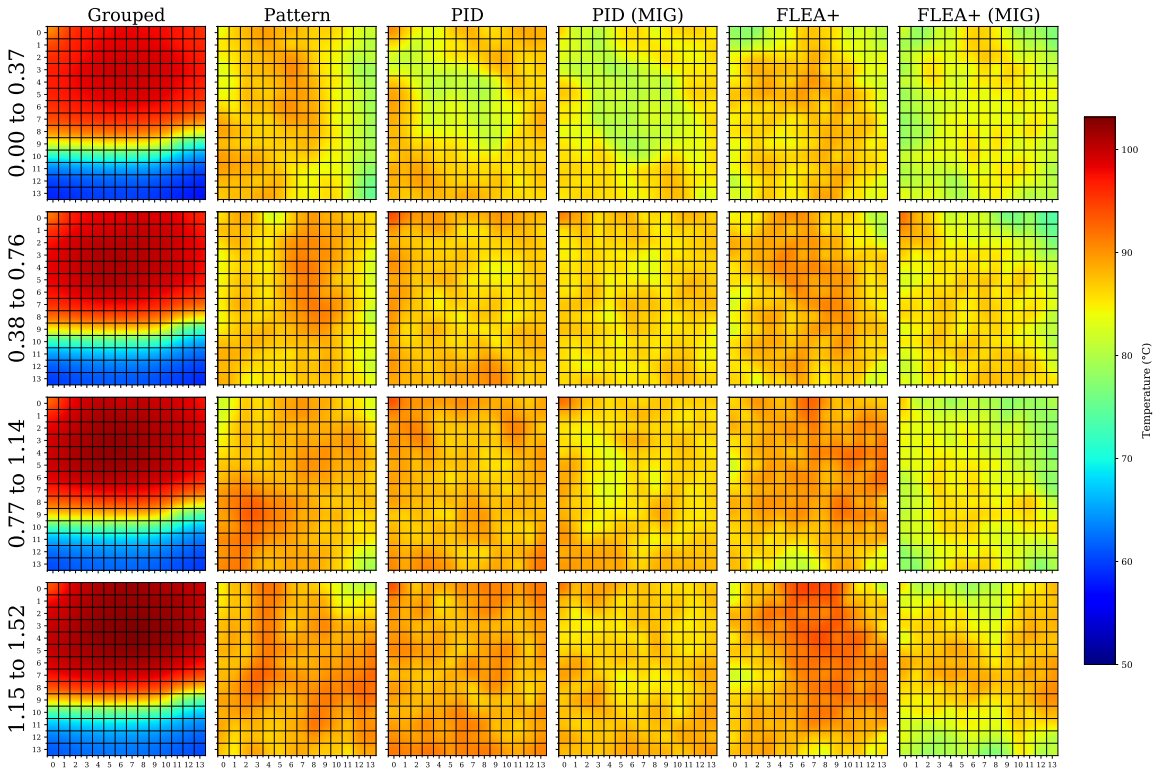


Figure 6.3: Heat maps comparison between different mapping and migration heuristics. Scenario 14x14 computation 70%.

Figure 6.3 shows that the Grouped heuristic has the worse thermal distribution. The Grouped heuristic has as its main goal reducing the hops between communicating tasks, thereby improving communication efficiency. Note that this goal, hop reduction, is the criterium adopted by most mapping heuristics. A significant drawback of this heuristic is the under-utilization of certain areas of the manycore, indicated by the blue (*coldest*) regions. While the Grouped strategy reduces hop count, it also creates thermal hotspots, as seen in the figure's red (*hottest*) regions. Thus, this heuristic demonstrates a trade-off between decreased hop count and thermal hotspots.

Contrasting with the Grouped heuristic, the Pattern mapping heuristic is designed to allocate tasks in an alternating pattern, as a chessboard, to distribute heat evenly across the manycore. However, in this scenario, with a 70% occupancy, achieving a perfect alternating allocation is unfeasible, resulting in hotspots where heat accumulation becomes more pronounced. This situation highlights the constraints of the Pattern heuristic in high task density scenarios, indicating its limited effectiveness in uniformly managing thermal load under such conditions. Figure C.25, which presents a 50% occupancy, shows that the Pattern can achieve a good thermal distribution, which corresponds to the limit of the Pattern mapping heuristic.

The PID heuristic computes a Proportional-Integral-Derivative (PID) score for each PE at every monitoring window. Task allocation is then directed toward PEs with the most favorable scores. In the initial phase of the simulation, ranging from 0 to 370 milliseconds,

there is a tendency for tasks to be predominantly allocated at the edges of the manycore. This initial pattern is due to the lack of thermal history required for computing accurate PID scores. However, as the simulation progresses and thermal data accumulates, task allocation under the PID heuristic becomes more balanced. This results in a uniform heat distribution across the many core surfaces in later snapshots, illustrating the heuristic's adaptability and effectiveness in managing thermal load over time.

We observe that PID without task migration capability reveals more thermal hotspots than seen with the Pattern heuristic. This phenomenon occurs because, in the PID approach, once a region is identified as favorable based on its PID score, several tasks may be allocated in this region, leading to increased heat generation in that area. However, the scenario changes significantly when observing the performance of PID with migration capability. Task migration in PID is activated when a PE is detected to be overheating. Tasks are relocated to areas with more advantageous PID scores. This dynamic and responsive strategy helps in mitigating the creation of hotspots. By continually adjusting task locations based on current thermal conditions and PID scores, the PID heuristic with migration capability achieves an effective thermal distribution.

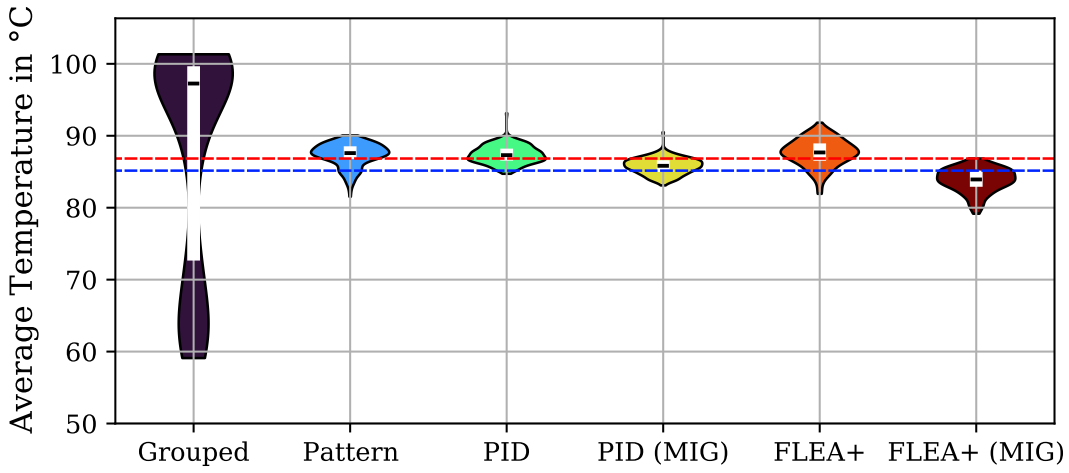
Observing the thermal behavior of FLEA+, it clearly produces more hotspot regions than those observed in the Pattern and PID strategies. In FLEA+, the temperature impacts the mapping process in the cluster selection, corresponding to the selection of the region for application mapping. Once the region is set, temperature no longer plays a role, and the task-to-core selection heuristic adopts a mapping strategy that chooses a PE consulting the Q-table using the available PBS within the cluster and the TPC that is being allocated. Since there is no task migration in the basic configuration, any suboptimal decisions taken during the initial mapping remain uncorrected. This lack of flexibility increases the hotspots, leading to the imbalanced thermal distribution observed in the manycore.

The migration capability enables FLEA+ to adapt by migrating tasks from overheating PEs to cooler locations, using real-time thermal data provided by TEA. Consequently, we observe that FLEA+ ensures that the average temperature of PEs remains cooler than that of other management strategies while sustaining the same workload.

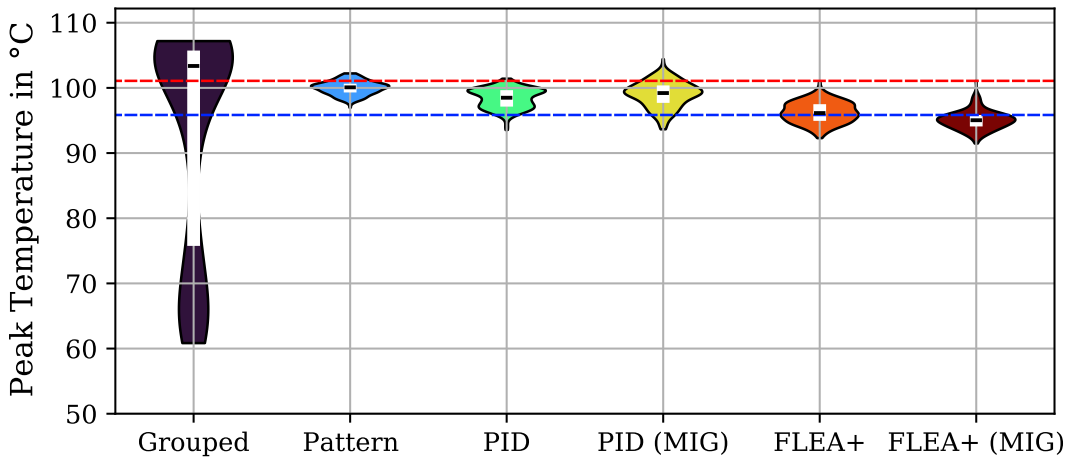
The adaptive behavior of FLEA+ with migration illustrates its effectiveness in handling thermal management. This is particularly noteworthy given that its training used a reward function related to reliability rather than temperature. By allowing for task migration response to violating thermal conditions, FLEA+ lowers the average PE temperature and mitigates the formation of thermal hotspots.

Figure 6.4 presents violin-box plots for the same scenario (14x14 computation 70%) that represent the thermal amplitude within the manycore, effectively illustrating the distribution of temperatures across its PEs. These plots provide a comprehensive view of the thermal distribution, with each violin-box plot spanning from the minimum to the maximum temperature. The width of each plot correlates with the density of PEs at specific tempera-

tures, aiding in identifying temperature outliers. Within each plot, a white bar represents the interquartile range, extending from the first quartile (the 25th percentile) to the third quartile (the 75th percentile). The median temperature is denoted by a black line within this white bar, indicating the temperature below which half of the PEs fall and above which the other half is situated. Additionally, the plots feature two dashed lines, red and blue, that facilitate a comparative analysis between different heuristics against FLEA+ with migration. The red line corresponds to the maximum temperature observed in FLEA+, while the blue line marks the temperature at the third quartile. These lines delineate the temperature spread of the hottest 25% of PEs, providing a clear reference for evaluating the effectiveness of various thermal management strategies in the manycore environment.



(a) Average Temperature



(b) Average Peak Temperature

Figure 6.4: Violin-box plot of PEs temperatures during the execution of scenario 14x14 computation 70%.

The violin plots in Figure 6.4 reveal specific details related to thermal management. For example, in Figure 6.4(a) - average temperature:

- The PID thermal management presents outliers, identified as a single PE with an average temperature of around 93℃ (PID) and 90℃ (PID MIG). In both cases, this outlier is the Manager PE (MPE), responsible for computing the PID scores. These PEs with higher temperatures (MPEs) can be observed in Figure 6.3 in the top-left corner, exhibiting a slightly redder hue than its surroundings, confirming the complementary nature of these visualizations.
- The hottest PEs under FLEA+ MIG are cooler than 75% of the PEs managed by the Pattern and PID. This indicates a more efficient thermal management in FLEA+ MIG. When compared to PID MIG, the difference becomes less pronounced. However, the temperature analysis still shows that most (75%) of PEs managed by PID MIG are warmer than the hottest PE in the top quartile managed by FLEA+ MIG. This comparison underscores the effectiveness of FLEA+ MIG in maintaining lower temperatures across the manycore. Furthermore, when migration is not incorporated, FLEA+ demonstrates a less effective thermal distribution than the Pattern and PID strategies, regardless of whether they include migration features. This observation highlights the significant role of task migration in enhancing the thermal performance of management heuristics like FLEA+.

The average peak temperature (Figure 6.4(b)) for this scenario shows that for the 25% of the PEs with the highest peak temperature, almost 100% of the PEs in the Pattern/PID/PID MIG heuristics are in this range. This is another result confirming the efficiency of FLEA+ MIG in thermal management.

These insights provide input for managing thermal profiles in manycore systems. They highlight the importance of task allocation and migration in sustaining performance and reliability over time, considering the harmful effects of excessive heat on electronic components. Thus, it is evident that while some heuristics may perform effectively under specific conditions, the versatility and adaptability of FLEA+ with task migration are essential for balancing workload and temperature distribution, ensuring the prolonged operational integrity of manycore systems.

The previous discussion focused on the temperature evaluation of one scenario among the 16 referenced in Table 6.1. Figures 6.5 and 6.6 present the average temperature and peak temperature compiled from each scenario. These scenarios are identified by their respective tags. Those starting with A, B, and C correspond to 8x8 manycore configurations, while the tags beginning with D, E, and F denote 14x14 manycore setups. Numbers after the letters, 5-7-9, corresponds to the system occupancy - 50%, 70%, and 90%.

Figure 6.5 shows the clear relationship between workload and temperature. The difference in average temperatures for different heuristics in the same scenario remains within a range of 5℃ at most. Important notes related to this graph:

- The average temperatures for different heuristics in the same scenario are close since the workload is the same;
- For scenarios in larger systems and higher computational load (E5-E7-E9) it is observed that FLEA+ MIG presents a reduction in the average temperature;
- The average temperature graph does not provide information related to thermal hotspots, justifying the plotting of thermal maps and the violin-box plots presented previously.

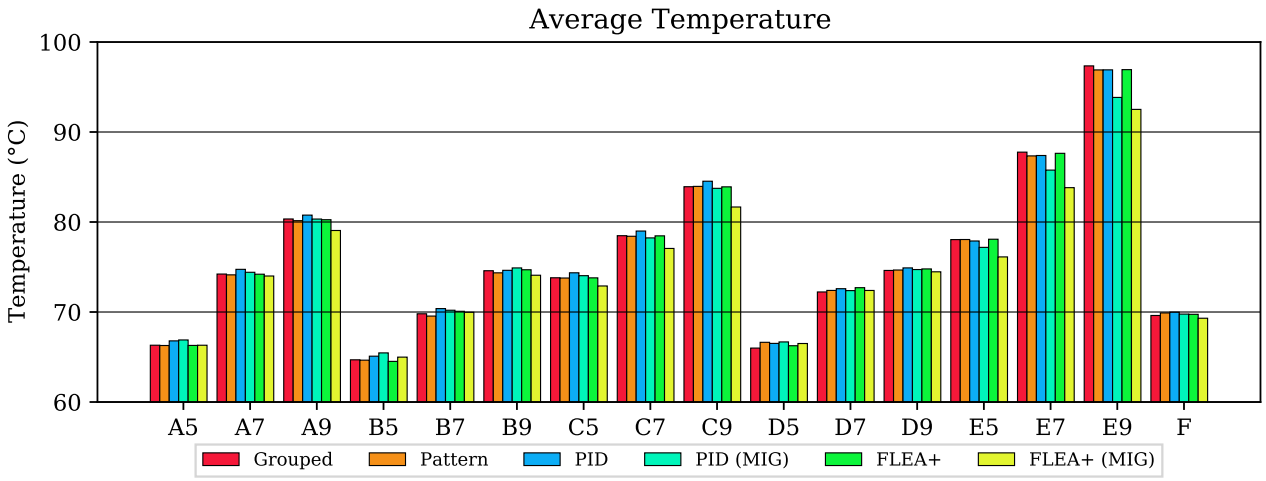


Figure 6.5: Average Temperature reached by the manycore when executing the proposed scenarios (Table 6.1) using different mapping and migration heuristics.

Figure 6.6 presents the average peak temperatures. Important notes related to this graph:

- As expected, the Grouped mapping presents the highest values;
- Scenarios with an occupancy of 50% (A5, B5, C5, D5, E5) have the Pattern mapping with the lowest average peak temperature, given that with this occupancy, the heuristic can evenly distribute the load on the PEs by interspersing used and unused PEs;
- FLEA+ (with and without migration) maintained the average peak temperature in all scenarios below the other management techniques (except in relation to Pattern in scenarios A5, B5, and C5);
- The PID could not minimize the average peak temperature in D5 scenario and PID and FLEA+ were unable to minimize it in E5.

Figures 6.5 and 6.6 present absolute temperature values. Table 6.3 helps to compare the (a) average temperature and (b) average peak temperature of FLEA+ MIG against other heuristics. The table employs a color-coding scheme with shades of green and red. The shades of green indicate a thermal advantage for FLEA+ MIG, while red tones a thermal disadvantage. From the Table 6.3(a), we may highlight that:

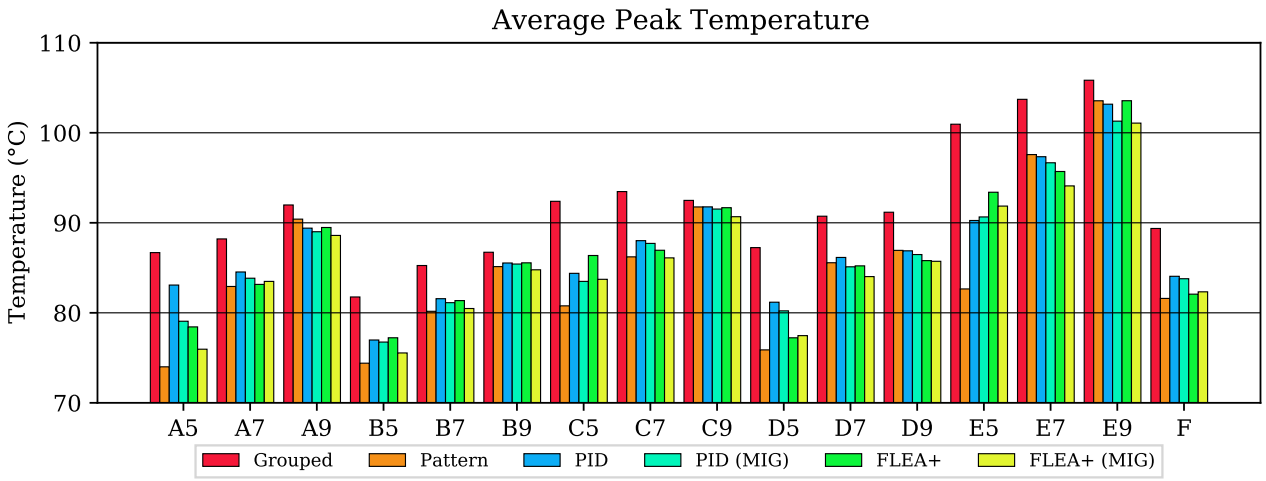


Figure 6.6: Average Peak Temperature reached by the manycore when executing the proposed scenarios (Table 6.1) using different mapping and migration heuristics.

- FLEA+ MIG consistently shows a thermal advantage in most scenarios, as indicated by the predominance of green shading across the scenarios. In particular, for high occupancy levels (A9, C9, E9);
- FLEA+ MIG provides larger average temperature reductions than heuristics without actuation (Grouped, Pattern), with differences reaching -4.84℃ for scenario E9;
- The thermal advantage of FLEA+ MIG is especially pronounced in CPU-bound workloads (scenarios C# and E#), with differences reaching -2.86℃ and -4.39℃ when compared to the PID and Pattern heuristics;
- It is also noticeable that FLEA+ becomes more efficient than other heuristics as the occupancy increases.
- Nevertheless, there are instances where FLEA+ MIG does not outperform other heuristics. The red shading in the table denotes these cases. However, the overall thermal profile of FLEA+ MIG is competitive, as illustrated by the average column.

Further considering Table 6.3(b), which compares the average peak temperature, we note that:

- The Grouped mapping heuristic significantly underperforms across all scenarios against every other heuristic, with the greatest thermal disadvantage reaching 10.73℃ over FLEA+ MIG.
- The Pattern mapping, on the other hand, demonstrates a mixed efficacy presenting a scenario where the average peak temperature was 9.21ºC smaller than FLEA+ MIG and other scenarios that underperform, reaching 3.48ºC over our heuristic. It successfully provides a thermal advantage by evenly distributing the workload in only half-occupied systems (A5, B5, C5, D5, E5). However, this advantage diminishes as the

Table 6.3: FLEA+ average temperature and average peak temperature comparison against other heuristics

(a) Average Temperature

Heuristic (avg temp)	Scenarios A5	A7	A9	B5	B7	B9	C5	C7	C9	D5	D7	D9	E5	E7	E9	F	Average
Grouped	0,00	-0,22	-1,28	0,30	0,17	-0,50	-0,91	-1,42	-2,25	0,51	0,16	-0,15	-1,93	-3,94	-4,84	-0,30	**-1,09**
Pattern	0,03	-0,13	-1,09	0,34	0,42	-0,26	-0,89	-1,36	-2,29	-0,13	-0,01	-0,20	-1,93	-3,52	-4,39	-0,59	**-1,03**
PID	-0,48	-0,74	-1,71	-0,10	-0,41	-0,54	-1,47	-1,94	-2,86	-0,01	-0,20	-0,43	-1,77	-3,57	-4,39	-0,68	**-1,37**
PID (MIG)	-0,58	-0,41	-1,27	-0,47	-0,23	-0,81	-1,15	-1,17	-2,07	-0,18	0,01	-0,25	-1,06	-1,95	-1,33	-0,46	**-0,86**
FLEA	0,02	-0,20	-1,20	0,48	-0,11	-0,60	-0,90	-1,40	-2,24	0,25	-0,31	-0,32	-1,97	-3,81	-4,42	-0,44	**-1,12**

(b) Average Peak Temperature

Heuristic (avg peak temp)	Scenarios A5	A7	A9	B5	B7	B9	C5	C7	C9	D5	D7	D9	E5	E7	E9	F	Average
Grouped	-10,73	-4,72	-3,38	-6,22	-4,77	-1,95	-8,67	-7,37	-1,81	-9,77	-6,71	-5,45	-9,09	-9,62	-4,77	-7,04	**-6,34**
Pattern	1,96	0,56	-1,81	1,13	0,31	-0,35	2,95	-0,12	-1,08	1,59	-1,54	-1,22	9,21	-3,48	-2,49	0,73	**0,37**
PID	-7,12	-1,05	-0,81	-1,44	-1,09	-0,76	-0,67	-1,92	-1,09	-3,71	-2,13	-1,17	1,60	-3,24	-2,11	-1,73	**-1,78**
PID (MIG)	-3,10	-0,35	-0,41	-1,21	-0,65	-0,64	0,23	-1,61	-0,85	-2,75	-1,09	-0,75	1,21	-2,57	-0,22	-1,46	**-0,98**
FLEA	-2,48	0,33	-0,88	-1,69	-0,88	-0,77	-2,65	-0,85	-1,00	0,25	-1,20	-0,08	-1,54	-1,60	-2,49	0,26	**-1,17**

occupancy increases. In every scenario, on average it was the only heuristic that presented better results for peak temperature than FLEA+ MIG (0.37ºC);

- Enabling migration reduced, on average, the peak temperature difference reached by PID heuristic of 0.8℃ (from 1.78℃ to 0.98℃). However, PID and PID MIG underperform against FLEA+ in almost every scenario, with peak temperature difference reaching up to 7.12℃;
- Even without migration, FLEA+ could produce an average peak temperature reduction compared to Grouped and PID, only losing to Pattern and PID MIG. Enabling FLEA+ migration produces an average reduction of 1.17℃ in the average peak temperature.

This temperature evaluation demonstrates the effectiveness of DTM methods with actuation mechanisms, specifically task migration, in managing thermal profiles. Although the absolute temperature differences across scenarios are minimal, thermal maps and violin-box plots reveal the importance of spatial temperature assessment. These graphs illustrate how heuristics like PID MIG and FLEA+ MIG successfully minimize thermal hotspots. We observed that heuristics requiring continuous metric calculations, as in the case of PID, tend to create a thermal hotspot in the PE performing these computations, adversely affecting the system MTTF. Additionally, the consistency of FLEA+ MIG in controlling the average peak temperature is a significant finding. Therefore, this initial set of results highlights the effectiveness of FLEA+ MIG in thermal management regarding absolute temperature values and the spatial distribution of temperature across the manycore system.

6.3.2 Hop Count Evaluation

As previously mentioned, the hop count is a critical metric for mapping because it represents the average distance between communicating tasks. A lower hop count reduces NoC traffic and congestion due to concurrent data flows. Although the OVP platform accurately models the behavior of the NoC, including its arbitration and routing algorithm, similar to an RTL model, it fails to replicate the effect of network traffic. Therefore, analyzing this parameter proves essential.

Figure 6.7 illustrates the average hop count for communicating tasks across all simulated scenarios (Table 6.1). The Grouped mapping, despite bringing communicating tasks closer together, presents high hop number values in several scenarios. The reason for this issue is the mapping fragmentation. At the end of applications and the insertion of new ones with a different number of tasks, the new tasks may be mapped to noncontinuous regions. The Grouped average hop number is 3.37, with a standard deviation of 1.24. For FLEA+ and FLEA+ MIG, these values are 3.02/3.42 (average values) and 0.72/0.66 (standard deviation). *These values show that FLEA+ can reduce the average distance between communicating tasks due to the clustering process, and the low standard deviation value demonstrates the scalability of the approach in the face of different system sizes and workloads.*

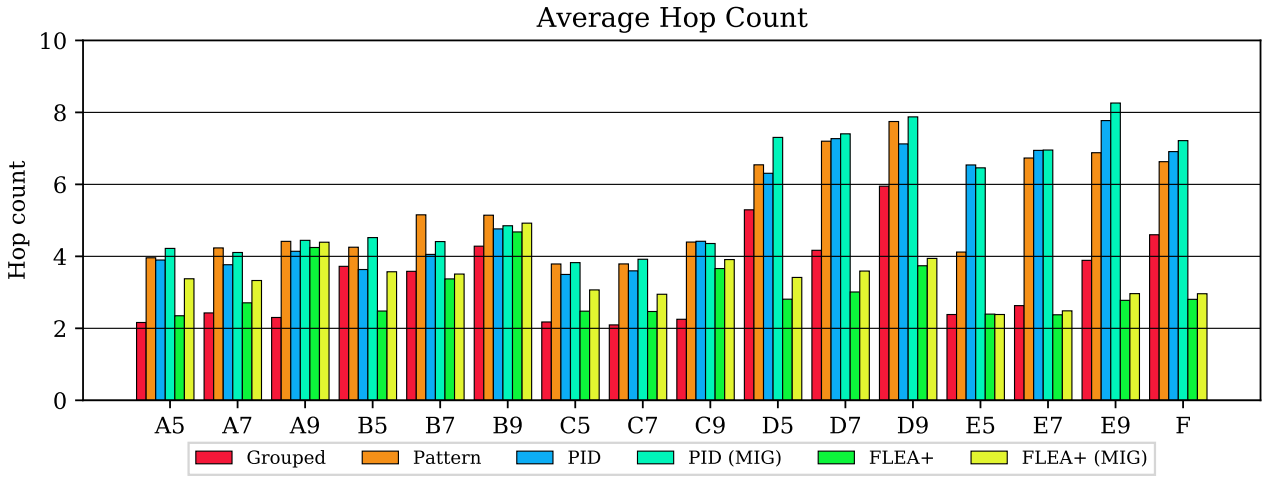


Figure 6.7: Average Hop Count between communicating tasks when executing the proposed scenarios (Table 6.1) using different mapping and migration heuristics

In 8x8 systems (A#, B#, and C#), the hop count increases with the system occupancy without showing high values (≈4). Note the patterning heuristic in cases B7 and B9, where the hop count is higher than other heuristics due to how this heuristic maps tasks (interleaving free and used PEs).

The behavior is distinct in 14x14 systems (D#, E#, and F). Given the larger search space, pattern, and PID heuristics excessively increase the hop count. The PID heuristic

searches for PEs with a lower PID score considering the entire system, which increases the hop count.

Our proposed approach, FLEA+, inherently minimizes the hop count due to adopting a clustering technique for mapping tasks. The hop count stays below 4 in most scenarios, except for scenarios A9 and B9. It is important to note that migration increases the hop count as tasks may migrate to the expanded cluster, outside of their initial clusters, with the goal of preventing hotspots and enhancing MTTF.

Table 6.4 provides a comparative analysis of hop count between FLEA+ MIG and other heuristics, highlighting the differences in percentages. The values shaded in green, which are negative, indicate the hop count reduction achieved by FLEA+ MIG in comparison to the corresponding heuristic. Conversely, the values shaded in red, which are positive, showcase the increases in hop count when FLEA+ MIG is compared with other heuristics.

Table 6.4: FLEA+ hop count comparison against other heuristics.

Heuristic (% hop)	Scenarios																Average
	A5	A7	A9	B5	B7	B9	C5	C7	C9	D5	D7	D9	E5	E7	E9	F	
Grouped	36,1	27,0	47,6	-4,2	-2,0	13,0	29,0	29,2	42,5	-55,1	-16,2	-51,0	0,0	-5,6	-31,4	-55,4	**3,9**
Pattern	-17,2	-27,0	-0,7	-19,0	-46,7	-4,5	-23,5	-28,5	-12,5	-91,8	-100,6	-96,7	-73,1	-170,3	-132,4	-124,0	**-56,3**
PID	-15,4	-13,2	5,7	-1,7	-15,4	3,3	-14,0	-22,0	-13,0	-85,0	-102,5	-80,7	-174,8	-178,7	-162,5	-133,4	**-58,0**
PID (MIG)	-24,9	-23,4	-1,1	-26,6	-25,6	1,4	-24,8	-32,9	-11,5	-114,1	-106,1	-100,0	-171,4	-179,1	-179,1	-143,9	**-67,9**
FLEA	30,5	18,6	3,2	30,5	4,0	4,9	19,2	16,3	6,4	17,6	16,2	5,1	-0,4	4,4	6,1	5,1	**12,2**

This table may be summarized as follows:

- Grouped, Pattern, and PID are not scalable in terms of hop number;
- The adoption of the clustering approach ensures a reduced average hop number, bringing scalability to FLEA+ (avg. hop number: 3,02; standard deviation: 0,72);
- The adoption of migration in FLEA+ to improve MTTF penalizes on average 12.2% the hop number (avg. hop number: 3,42; standard deviation: 0,66), but note that the largest penalties occur in systems with a small occupancy (A5, B5) where the heuristic has a larger number of PEs to explore.

6.3.3 Reliability Evaluation

The reliability evaluation begins by analyzing the impact of each wear-out effect on the FIT. Specifically, we focus on the FIT graph for a 14x14 manycore with 90% occupancy, detailed in Figure 6.8. For a comprehensive view of FIT data across other scenarios, please refer to Appendix C. It is important to note that the FIT values are relative to a baseline value established for each wear-out effect. During our experiments, we used this baseline value equal to 800 for all wear-out effects.

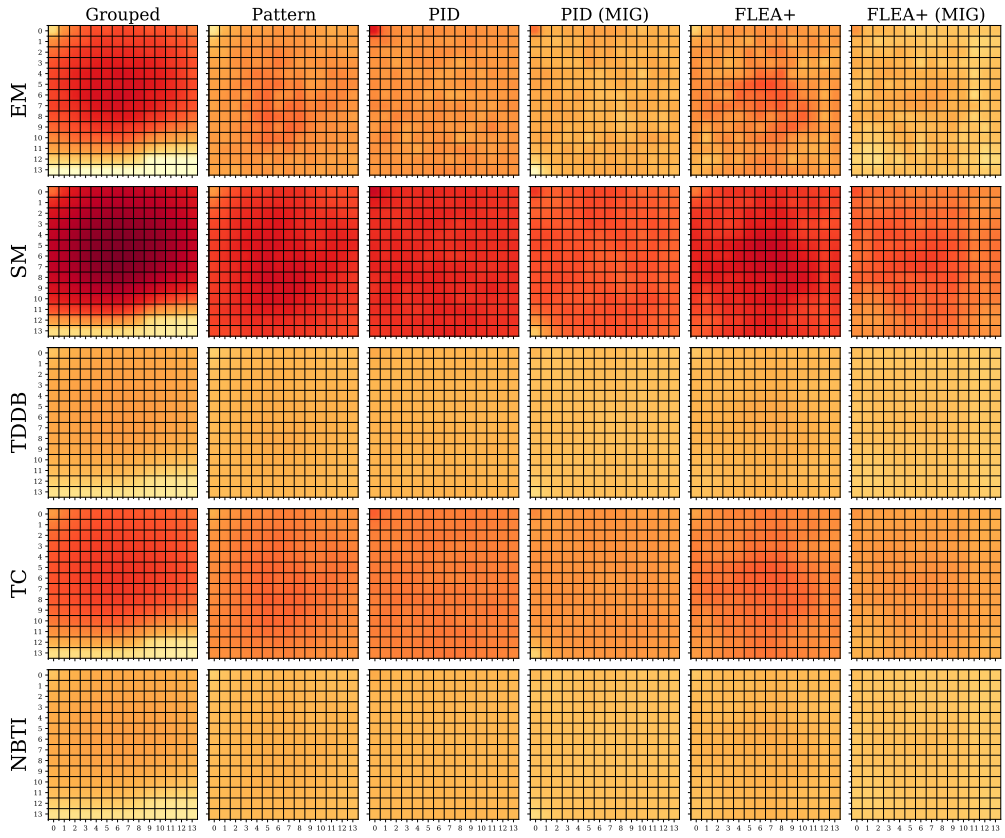


Figure 6.8: Manycore FIT intensity map per evaluated wear-out effect in the scenario 14x14 computation 90%.

The graph shows the intensity of various wear-out effects, including electromigration (EM), stress migration (SM), time-dependent dielectric breakdown (TDDB), thermal cycling (TC), and negative bias temperature instability (NBTI). Upon initial observation of the FIT graph, some trends become apparent. Notably, the FIT values associated with SM, TC, and EM are of the greatest magnitude.

A divergence in the EM effect is observed on the MPE, located in the top-left corner, under the PID heuristic. The FIT intensity for EM on this specific PE is significantly higher than the EM FIT values of other PEs within the manycore. This difference is due to the continuous computational load on the MPE, as it consistently executes the PID score calculations for each PE. This processing load on the MPE increases its vulnerability to EM. Consequently, the FIT rate for EM on the MPE is elevated, creating a reliability bottleneck within the system.

Figure 6.9, presents the FIT distribution14x14 computation at 50% occupancy. In this Figure, under the Pattern heuristic, we observe a chessboard pattern distribution across the PEs within the manycore. As the system's occupancy does not exceed 50%, it successfully interleaves the PEs, achieving an even distribution of FIT values throughout the system. This pattern efficiently mitigates concentrated wear-out as it evenly spreads the workload.

The chessboard distribution is intriguingly replicated under the FLEA+ heuristic, both with and without migration capabilities. This replication indicates that the reinforcement

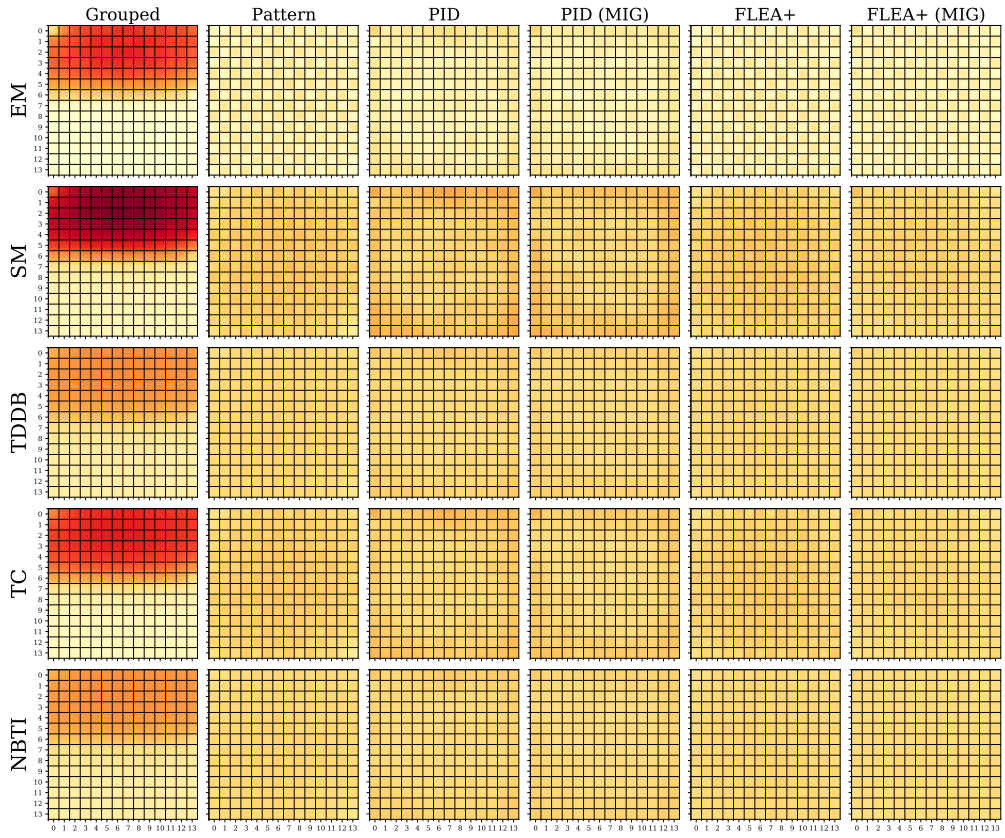


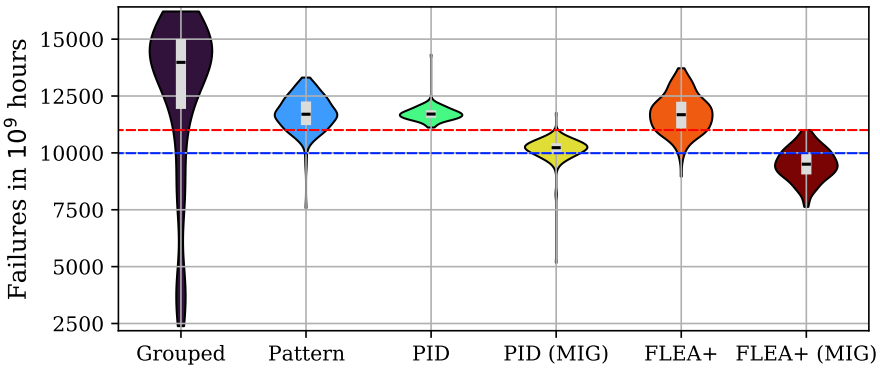
Figure 6.9: Manycore FIT intensity map per evaluated wear-out effect in the scenario 14x14 computation 50%.

learning algorithm in FLEA+ has learned a FIT minimization strategy akin to the Pattern. By rewarding reductions in FIT, the reinforcement learning process converged on a wear-leveling approach similar to the manually designed chessboard pattern. *This convergence validates the effectiveness of Pattern mapping and demonstrates the capability of reinforcement learning techniques to discover and apply complex strategies autonomously.*

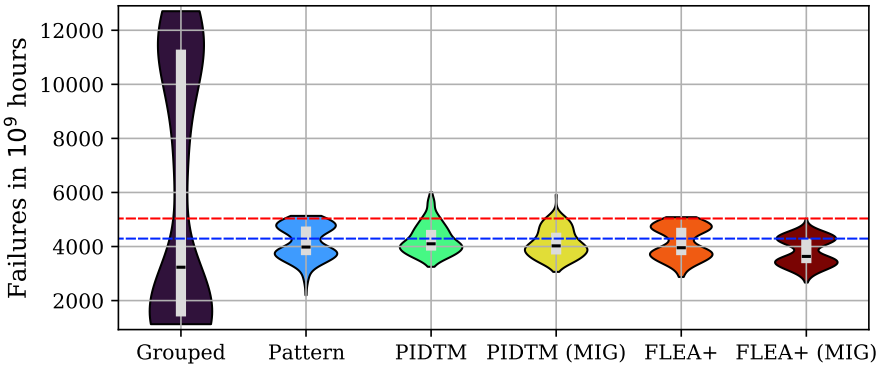
Violin-box plots have been used to enhance the visualization of the FIT distribution. Figure 6.10(a) and (b) display the FIT distribution graphs for the scenarios illustrated in Figures 6.8 and 6.9.

Referring to the 90% occupancy scenario in Figure 6.10(a), we corroborate the findings from the earlier figure. Generally, the top 25% PEs with the highest FIT values in FLEA+ MIG are lower than the most worn 75% of PEs under static mapping techniques (Grouped, Pattern, PID, and FLEA+ without migration). Even in comparison to PID MIG, it is observable that nearly all PEs fall within the range of the top 25% highest FIT PEs of FLEA+ MIG.

The graphical representation provided by the violin-box plots facilitates a more precise identification of outlier PEs. Figure 6.10(a), Pattern heuristic, outliers with relatively low FIT values are observable, corresponding to the MPE. The MPE, having a lower management load and spending considerable idle time, exhibits a scenario inverse to that observed



(a) 14x14 computation 90%



(b) 14x14 computation 50% (without clustering)

Figure 6.10: Violin-box plot showing FIT distribution for 14x14 computation (a) 90% and (b) 50% occupancy scenarios, respectively.

in the PID heuristic. In the case of the PID heuristic, an outlier is detected with a comparatively high FIT value due to PID score computation.

Figure 6.10(b) exhibits the FIT distribution from the experiment using the scenario 14x14 computation with 50% occupancy. The violin-box plot illustrates how the FLEA+ heuristic mirrors the chessboard organization inherent to the Pattern strategy.

We also observe the impact of task migration in Figure 6.10. By comparing FLEA+ and FLEA+ MIG, the task migration reduces the number of PEs with high FIT values. The median FIT value for PEs under FLEA+ MIG is lower, showing that migration reduces PE wearing.

Figure 6.11 displays visually the MTTF values of each simulated scenario (Table 6.1). The MTTF was estimated using a Monte Carlo simulation technique proposed by Srinivasan et al. [Srinivasan et al., 2005], as previously described in Section 3.3.2.

Analyzing the MTTF graph, Figure 6.11 we may note that:

- Systems with an 8x8 configuration (scenarios A#, B#, and C#) exhibit a higher MTTF compared to their 14x14 (scenarios D#, E#, and F), due to fewer cores at risk of failure. It should be remembered that our reliability analysis assumes a series model where

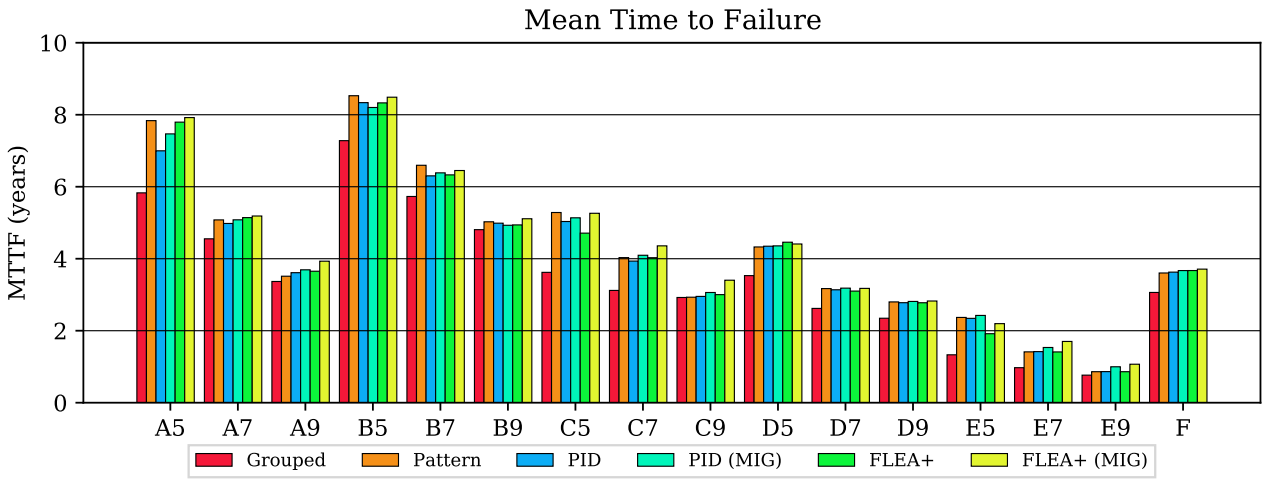


Figure 6.11: MTTF estimation when executing the proposed scenarios using different mapping and migration heuristics.

the system is considered failed if any PE fails, as there is no fault tolerance mechanism in place.

- An increase in system occupancy directly impacts reliability, leading to a reduced MTTF as occupancy levels rise. This trend is evident when comparing scenarios with the same workload, such as A5, A7, and A9.
- Grouped mapping consistently shows the poorest MTTF across all scenarios.
- Pattern and PID exhibit MTTFs close to those of FLEA+ MIG in several instances.

Table Table 6.5 consolidates the MTTF results. The table entries represent the percentage difference, with positive values indicating how much better FLEA+ MIG performs than other heuristics. Negative values, on the other hand, mean scenarios where FLEA+ MIG performs worse.

Table 6.5: FLEA+ with migration MTTF comparison against other heuristics.

Heuristic (MTTF)	Scenarios																Average
	A5	A7	A9	B5	B7	B9	C5	C7	C9	D5	D7	D9	E5	E7	E9	F	
Grouped	26,39	12,33	14,25	14,25	11,16	5,87	31,18	28,44	14,12	19,95	17,61	16,96	39,55	42,94	28,04	17,52	**21,54**
Pattern	1,01	2,12	10,69	-0,47	-2,33	1,57	-0,38	7,57	13,82	1,81	0,31	1,06	-7,73	17,06	19,63	2,96	**4,38**
PID	11,62	4,05	8,14	1,77	2,33	2,35	4,37	9,86	13,24	1,36	1,26	1,77	-6,36	16,47	19,63	2,16	**6,12**
PID (MIG)	5,68	2,12	6,11	3,42	0,93	3,52	2,28	5,96	10,00	1,13	0,00	0,71	-10,00	10,00	6,54	1,08	**3,23**
FLEA	1,64	0,96	7,12	1,88	1,86	3,33	10,46	7,57	11,76	-1,13	2,52	1,77	12,73	17,06	19,63	1,08	**6,61**

An analysis of Table 6.5 reveals that the FLEA+ MIG heuristic yields an average improvement of 21.54% in MTTF over the Grouped heuristic. Compared to the Pattern heuristic, the average enhancement is 4.38%, indicating that while FLEA+ MIG offers benefits, but the extent of improvement varies significantly depending on the specific scenario. This variation ranges from a decrease of 7.73% in scenario E5 to an increase of 19.63% in

E9. Compared to the PID heuristic, FLEA+ MIG shows an average improvement of 6.12%, and a 3.23% improvement against PID MIG.

These results demonstrate the efficiency of FLEA+ MIG to maximize MTTF in scenarios with higher workloads and larger system dimensions.

6.3.4 Scalability Evaluation

In the evaluation of the scalability aspects of different heuristics, we consider both time complexity which deals with how processing time increases with input size, and space complexity which considers the amount of memory used.

The *Grouped* and *Pattern* heuristics exhibit time complexity of $O(1)$. This behavior is due to their operation mechanism that involves querying a priority address table indexed by an incremented pointer with each allocation. It also indicates that their spatial complexity is $O(n_{PES})$, where n_{PES} represents the number of PEs since the space required is proportional to the number of entries which is dependent on the number of PEs.

Regarding the *PID* heuristic, it can be stated that the time complexity to perform a task allocation is $O(n_{PES})$ because it requires searching through an unordered list to determine which PE has the optimal PID score. However, the calculation of the PID score also has a complexity of $O(n_{PES})$ and must be executed within each monitoring window. Thus, there is an upper bound on the scalability of the PID heuristic's execution imposed by the size of the monitoring window. If the PID score for each PE cannot be computed within the monitoring window's duration, the heuristic's functionality is compromised. Additionally, the spatial resources required by the PID approach exceed those of the previous heuristics since it involves maintaining a thermal history of 10 monitoring windows for each PE. Nonetheless, its asymptotical spatial complexity remains $O(n_{PES})$.

The scalability of the *FLEA* algorithm must be assessed in two stages. The first stage involves the cluster selection process that occurs once for the allocation of each application. As this involves a sliding window search iterated once over the entire dimension of the manycore, its complexity is $O(n_{PES})$. Following the cluster selection, the task allocation starts, where the search is restricted within the chosen cluster, leading to a complexity of $O(len(cluster))$, with $len(cluster)$ representing the size of the cluster. In terms of spatial demands, the heuristic requires only the trained policy table, which has a constant size, leading to a space complexity of $O(1)$.

To test their scalability, we simulated a 20x20 system with 400 PEs executing computation-intensive workloads with 70% system utilization. Figure 6.12 displays temperature snapshots for all tested management heuristics. It reveals that the thermal distribution

created by the Pattern heuristic generates several hotspots; as previously discussed, its ability to distribute heat drops significantly when system utilization exceeds 50%.

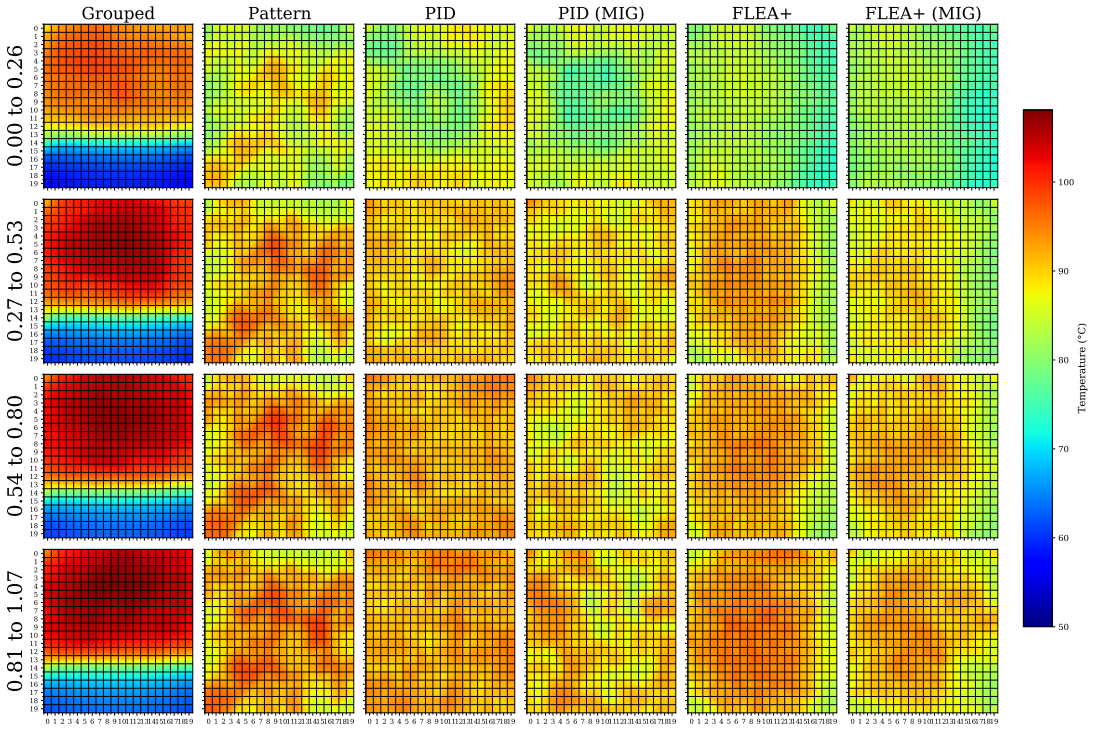
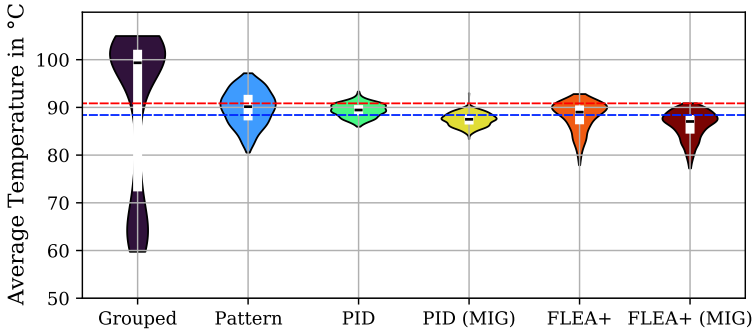


Figure 6.12: Average temperature snapshot of a 20X20 system executing a compute-intensive workload with 70% of system occupation.

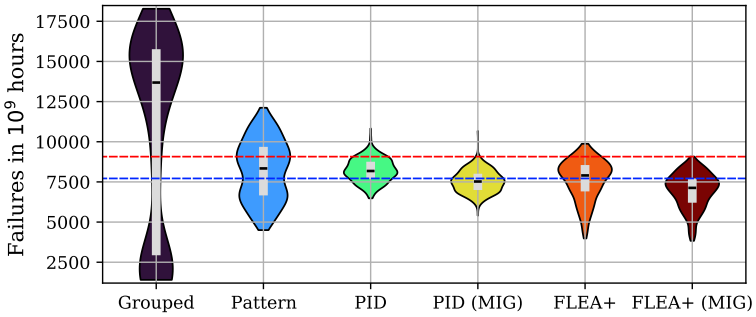
Furthermore, we observe that migration assists in reducing the heat concentration for both PID and FLEA+ heuristics. Figures 6.13(a) and 6.13(b) present the average temperature and FIT distributions, respectively, through violin-box plots. These figures allow us to compare the thermal distribution more effectively. Comparing Pattern to the FLEA+, almost 50% of the PEs using Pattern present average temperatures higher than those seen in the hottest PE managed by FLEA+.

Additionally, about 75% of PEs managed by PID and 50% of those with PID with migration reach temperatures within the hottest 25% range for FLEA+. Outliers in both PID management techniques result from intense and periodic computing in the MPE to determine the system's PID score. This simulation demonstrates that even with a doubling of the number of PEs, 196 to 400, we keep the thermal and FIT management capacity.

Overall, the evaluation reveals that the Grouped and Pattern heuristics maintain a constant time complexity, while the scalability of the PID heuristic is limited by the size of the monitoring window. The FLEA algorithm demonstrates an intermediate time complexity, which depends on the cluster size following the initial selection process, while maintaining constant spatial demands. It is clear that while the Grouped and Pattern heuristics offer excellent scalability with constant time complexity and predictable resource requirements, the PID heuristic has potential limitations due to its reliance on the monitoring window size. The FLEA algorithm emerges as a compromise solution, enabling scalability while efficiently



(a) PE average temperature violin-box plot



(b) PE failure in time violin-box plot.

Figure 6.13: Violin-box plots for of a 20X20 system executing a compute-intensive workload with 70% of system occupation.

allocating resources within clusters. This analysis helps in understanding the advantages and disadvantages of each heuristic as systems expand, encompassing both processing capabilities and memory resources.

6.4 Final Remarks

This Chapter evaluated DTM and DRM heuristics, using the reference manycore, with a particular focus on the FLEA+ heuristic. The experimental setup, encompassing sixteen scenarios, provided robust thermal and reliability evaluations. The results offer insights into the efficacy of the FLEA+ heuristic alongside other mapping strategies like Grouped, Pattern, and PID mappings.

A significant finding is the enhanced wear-leveling capability of the FLEA+ heuristic, especially when migration is enabled. The lower FIT values observed in the top quartile of PEs under FLEA+ MIG, compared to the original FLEA, show its superiority in mitigating wear-out. This aspect is crucial in prolonging the lifetime of manycores, ensuring their reliability. The violin-box plots further reinforced these findings, providing a clear and comprehensive view of PE FIT distribution across the various scenarios.

Regarding thermal management, FLEA+ MIG demonstrated a consistent thermal advantage across most scenarios, particularly at higher occupancy levels. This proves its efficient thermal management capabilities, which are essential in keeping the operational integrity and performance of manycores.

Results also highlighted the scalability challenges associated with other heuristics like Grouped, Pattern, and PID mappings, particularly in terms of hop count. FLEA+ MIG reduced hop count compared to these heuristics, indicating its better scalability. This aspect is critical in larger manycores, where efficient data transfer and communication are pivotal.

Despite its strengths, FLEA+ is not without its weaknesses. Compared to Pattern and Grouped, a potential drawback is the increased computational cost of identifying the PBS for a PE. However, it only occurs once for each task allocation/migration. Furthermore, the effectiveness of the Pattern mapping in half-occupied systems suggests that FLEA+ might benefit from further optimization in scenarios with varying occupancy levels.

In conclusion, the FLEA+ heuristic emerges as an effective approach for managing thermal and reliability aspects in manycore. Its ability to enhance wear-leveling, manage thermal behavior efficiently, and scale effectively in hop count makes it a valuable tool in manycore system management. The findings contribute to the existing body of knowledge and pave the way for more resilient and efficient manycore systems in the future.

7. CONCLUSIONS AND FUTURE WORK

This Chapter presents the conclusions drawn from the research presented in this Thesis and offers recommendations for future work.

7.1 Conclusions

The Thesis statement, presented in the Introduction, declares that developing and applying scalable Dynamic Thermal Management (DTM) and Dynamic Reliability Management (DRM) strategies for manycore systems is feasible using a Reinforcement Learning (RL) technique. The research, development, and evaluation conducted in this Thesis have significantly contributed to this domain. A novel technique named FLEA was introduced, integrating reliability-aware RL into system-level task management. This integration aims to enhance the lifetime reliability of manycore systems.

We developed an abstract manycore platform, Chronos-V, to enable the deployment of the management heuristics. Chronos-V was a fundamental tool for executing, developing, and testing the RL-based management strategy. The platform was created with customizable hardware, integrating an Instruction Set Simulator (ISS) with high-level models of routers, memory, peripherals, and the Temperature Estimator Accelerator (TEA), initially described at the RTL level.

On the software side, we expanded the FreeRTOS with a message-passing interface and three management modules – monitoring, decision, and actuating. A periodic process monitors the temperature, with each PE estimating its energy consumption and transmitting this data to the TEA peripheral. TEA then computes the temperature of each PE and sends this information to the manager PE (MPE). The MPE executes the decision-making heuristics, including application mapping, task migrations, and dynamic voltage and frequency scaling. The actuation occurs at the PE level.

The proposed management technique, FLEA, proved effective through a policy lookup table trained with Q-learning, an RL technique. The main advantage of using Q-learning is its model-free ability to create a trial-and-error task allocation strategy during the design phase based on reliability feedback, leading to significant improvements in reliability and efficiency.

Comparative analysis and extensive simulations demonstrated that the RL-based task management strategy outperforms existing methods in thermal and reliability performance in the proposed scenarios. The results indicated that the heuristic based on a lookup table effectively controlled the mean and peak temperatures, achieving a better spatial temperature distribution across the manycore. FLEA also significantly mitigated temperature-

related wear-out effects, showing an average MTTF improvement of 4.38% over Pattern, 6.12% over PID, and 3.23% over PID with migration, thus extending the expected manycore lifetime.

The effectiveness of this RL-based approach not only corroborates the Thesis statement but also lays the foundation for future research into the reliability management of manycore systems. In summary, this Thesis demonstrates the effectiveness and practicality of RL-based strategies in improving the lifetime reliability of manycore systems. Through simulations and systematic comparisons with existing techniques, this research shows that an RL-based task management approach represents a significant advance in managing manycore' thermal and reliability constraints, thus validating the hypothesis proposed in the thesis statement.

7.2 Future Research Directions

Moving forward from the research presented in this Thesis, the Chronos-V platform and the FLEA heuristic offer different paths to advance the research. The following points outline potential future works that could significantly extend this research:

- **Optimization of Chronos-V Platform:** Continued platform development may address the identified bottlenecks in the Chronos-V model. The performance profiling has indicated that our manycore model currently achieves an average of $\sim$50% utilization on each host machine core. Optimizing the simulator's code and refactoring critical sections, such as the "iterator", could improve the parallelization efficiency, leading to faster and more scalable parallel simulations.
- **Graphical User Interface (GUI) for Chronos-V:** Designing and implementing a GUI would assist in real-time analysis and monitoring of simulations. As highlighted in [Ruaro and J. M. Martin, 2022], a GUI can significantly accelerate manycore development.
- **FLEA Evaluation with Multitasking:** Adapting the FLEA heuristic to handle multitasking scenarios presents a research challenge. Currently, FLEA queries the Q-table by indexing the Task Power Category (TPC) and PE bin state (PBS). The challenge lies in determining the PBS for a PE with multitasking PEs neighboring. One solution could involve using the PE's capability to estimate its power consumption, using an average value to infer an equivalent TPC for that PE.
- **Dynamic Voltage and Frequency Scaling (DVFS) using FLEA:** Incorporating DVFS strategies into the FLEA heuristic stands as another actuation knob to further enhance thermal behavior and reliability. This topic requires research into modeling a scalable

actuation space in which reinforcement learning can relate task deadlines to the PEs' speed.

- **Development of a Reliability Estimator Accelerator (REA):** Similarly to the development of the peripheral Thermal Estimation Accelerator (TEA) [Silva et al., 2019], a REA could enable online learning capabilities. The system can continuously maintain the reinforcement learning training by calculating the Failure in Time (FIT) in real-time. This poses an engineering challenge, requiring the implementation of aging models in hardware at the RTL level and assessing their area, power impact, and scalability for larger systems.

- **Re-Evaluation of *Q*-learning Learning Rate (Alpha) for Online Training:** With the shift to online training, leveraged by an REA module, the learning rate decay applied in the *Q*-learning training requires reconsideration. Rather than tending towards zero as during design-time training presented in this Thesis, another decay rate, or possibly a dynamic adjustment strategy, will need to be determined. Besides that, another important piece of information is establishing how long the system must train before achieving adequate performance with an uninitialized policy table.

- **Validation of FLEA on RTL-Level Platform Simulation:** To assess the practical applicability and accuracy of the FLEA heuristic in detail, it is essential to apply the *Q*-table for mapping on a platform modeled at the RTL-level and evaluate temperature and reliability results. The simulation at this lower abstraction level—with clock cycle accuracy—will validate the FLEA heuristic under more realistic and granular conditions.

- **Real-System Evaluation with Thermal Imaging:** Testing the heuristics on actual hardware platforms and performing thermal evaluations via thermal imaging, as detailed in the literature [Zhang et al., 2023], would provide invaluable data. This step would ground the theoretical models and identify unforeseen practical constraints and opportunities for further refinement.

In summary, this set of future works promises to advance the current state-of-the-art dynamic thermal and reliability management systems. Such advancements would have far-reaching implications for many applications, from high-performance computing to sustainable and resilient electronic device operation.

REFERENCES

[Abadeer et al., 1999] Abadeer, W. W. B., Bagramian, A., Conkle, D. W., Griffin, C. W., Langlois, E., Lloyd, B. F., Mallette, R. P., Massucco, J. E., McKenna, J. M., Mittl, S. W., and Noel, P. H. (1999). Key measurements of ultrathin gate dielectric reliability and in-line monitoring. *IBM J. Res. Dev*, 43(3):407–416. https://doi.org/10.1147/rd.433.0407.

[Amid et al., 2020] Amid, A., Biancolin, D., Gonzalez, A., Grubb, D., Karandikar, S., Liew, H., Magyar, A., Mao, H., Ou, A. J., Pemberton, N., Rigge, P., Schmidt, C., Wright, J. C., Zhao, J., Bachrach, J., Shao, Y. S., Nikolic, B., and Asanovic, K. (2020). Invited: Chipyard - An Integrated SoC Research and Implementation Environment. In *ACM/IEEE Design Automation Conference (DAC)*, pages 1–6. https://doi.org/10.1109/DAC18072.2020.9218756.

[Arrhenius, 1889] Arrhenius, S. (1889). Uber die Reaktionsgeschwindigkeit bei der Inversion von Rohrzucker durch Säuren. *Zeitschrift für physikalische Chemie*, 4(1):226–248. https://doi.org/10.1515%2Fzpch-1889-0416.

[Benini et al., 2012] Benini, L., Flamand, E., Fuin, D., and Melpignano, D. (2012). P2012: Building an ecosystem for a scalable, modular and high-efficiency embedded computing accelerator. In *IEEE Design, Automation Test in Europe Conference (DATE)*, pages 983–987. https://doi.org/10.1109/DATE.2012.6176639.

[Bergman and Lavine, 2017] Bergman, T. L. and Lavine, A. S. (2017). *Fundamentals of Heat and Mass Transfer*. John Wiley & Sons, 8th edition. 992p.

[Bertozzi et al., 2005] Bertozzi, D., Jalabert, A., Murali, S., Tamhankar, R., Stergiou, S., Benini, L., and De Micheli, G. (2005). NoC synthesis flow for customized domain specific multiprocessor systems-on-chip. *IEEE Transactions on Parallel and Distributed Systems*, 16(2):113–129. https://doi.org/10.1109/TPDS.2005.22.

[Bienia et al., 2008] Bienia, C., Kumar, S., Singh, J. P., and Li, K. (2008). The PARSEC benchmark suite: Characterization and architectural implications. In *International Conference on Parallel Architectures and Compilation Techniques (PACT)*, pages 72–81. https://doi.org/10.1145/1454115.1454128.

[Binkert et al., 2011a] Binkert, N. et al. (2011a). The Gem5 Simulator. *SIGARCH Comput. Archit. News*, 39(2):1–7. https://doi.org/10.1145/2024716.2024718.

[Binkert et al., 2011b] Binkert, N. L., Beckmann, B. M., Black, G., Reinhardt, S. K., Saidi, A. G., Basu, A., Hestness, J., Hower, D., Krishna, T., Sardashti, S., Sen, R., Sewell, K., Altaf, M. S. B., Vaish, N., Hill, M. D., and Wood, D. A. (2011b). The gem5 simulator. *SIGARCH Comput. Archit. News*, 39(2):1–7. https://doi.org/10.1145/2024716.2024718.

[Black, 1969] Black, J. (1969). Electromigration—A brief survey and some recent results. *IEEE Transactions on Electron Devices*, 16(4):338–347. https://doi.org/10.1109/T-ED.1969.16754.

[Bohr, 2007] Bohr, M. (2007). A 30 Year Retrospective on Dennard's MOSFET Scaling Paper. *IEEE Solid-State Circuits Society Newsletter*, 12(1):11–13. https://doi.org/10.1109/N-SSC.2007.4785534.

[Bolchini et al., 2014] Bolchini, C., Carminati, M., Gribaudo, M., and Miele, A. (2014). A lightweight and open-source framework for the lifetime estimation of multicore systems. In *IEEE International Conference on Computer Design (ICCD)*, pages 166–172. https://doi.org/10.1109/ICCD.2014.6974677.

[Borkar, 2007] Borkar, S. (2007). Thousand Core Chips: A Technology Perspective. In *ACM/IEEE Design Automation Conference (DAC)*, pages 746–749. https://doi.org/10.1145/1278480.1278667.

[Calazans et al., 2003] Calazans, N. L. V., Moreno, E. I., Hessel, F., da Rosa, V. M., Moraes, F., and Carara, E. (2003). From VHDL Register Transfer Level to SystemC Transaction Level Modeling: A Comparative Case Study. In *IEEE Symposium on Integrated Circuits and Systems Design (SBCCI)*, page 355. https://doi.org/10.1109/SBCCI.2003.1232853.

[Carara et al., 2009] Carara, E. A., Oliveira, R. P. d., Calazans, N. L. V., and Moraes, F. G. (2009). HeMPS - a Framework for NoC-based MPSoC Generation. In *IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1345–1348. https://doi.org/10.1109/ISCAS.2009.5118013.

[Carlson et al., 2011] Carlson, T. E., Heirman, W., and Eeckhout, L. (2011). Sniper: exploring the level of abstraction for scalable and accurate parallel multi-core simulation. In *ACM Conference on High Performance Computing Networking, Storage and Analysis (SC)*, pages 52:1–52:12. https://doi.org/10.1145/2063384.2063454.

[Castilhos et al., 2016] Castilhos, G., Moraes, F. G., and Ost, L. (2016). A lightweight software-based runtime temperature monitoring model for multiprocessor embedded systems. In *IEEE Symposium on Integrated Circuits and Systems Design (SBCCI)*, pages 1–6. https://doi.org/10.1109/SBCCI.2016.7724040.

[Cataldo et al., 2018] Cataldo, R., Fernandes, R., Martin, K. J. M., Sepúlveda, J., Susin, A. A., Marcon, C. A. M., and Diguet, J. (2018). Subutai: distributed synchronization primitives in NoC interfaces for legacy parallel-applications. In *ACM/IEEE Design Automation Conference (DAC)*, pages 83:1–83:6. https://doi.org/10.1145/3195970.3196124.

[Chandrakasan et al., 2001] Chandrakasan, A. P., Bowhill, W. J., and Fox, F. (2001). *Design of high-performance microprocessor circuits*. Wiley-IEEE press, 1st edition. 584p.

[Chaturvedi et al., 2012] Chaturvedi, V., Huang, H., Ren, S., and Quan, G. (2012). On the fundamentals of leakage aware real-time DVS scheduling for peak temperature minimization. *Journal of Systems Architecture*, 58(10):387–397. https://doi.org/10.1016/j.sysarc.2012.08.002.

[Chen et al., 2023] Chen, K., Liao, Y., Chen, C., and Wang, L. (2023). Adaptive Machine Learning-Based Proactive Thermal Management for NoC Systems. *IEEE Trans. Very Large Scale Integr. Syst*, 31(8):1114–1127. https://doi.org/10.1109/TVLSI.2023.3282969.

[Cheng et al., 1998] Cheng, Y., Raha, P., Teng, C., Rosenbaum, E., and Kang, S. (1998). ILLIADS-T: an electrothermal timing simulator for temperature-sensitive reliability diagnosis of CMOS VLSI chips. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 17(8):668–681. https://doi.org/10.1109/43.712099.

[da Silva, 2021] da Silva, A. H. L. (2021). *Dynamic Thermal Management For Noc-based Many-core Systems*. PhD thesis, Pontifical Catholic University of Rio Grande do Sul, PPGCC. 115p.

[Das et al., 2016] Das, A., Al-Hashimi, B. M., and Merrett, G. V. (2016). Adaptive and Hierarchical Runtime Manager for Energy-Aware Thermal Management of Embedded Systems. *ACM Trans. Embed. Comput. Syst*, 15(2):24:1–24:25. https://doi.org/10.1145/2834120.

[Das et al., 2014] Das, A., Shafik, R. A., Merrett, G. V., Al-Hashimi, B. M., Kumar, A., and Veeravalli, B. (2014). Reinforcement learning-based inter-and intra-application thermal optimization for lifetime improvement of multicore systems. In *ACM/IEEE Design Automation Conference (DAC)*, pages 1–6. https://doi.org/10.1145/2593069.2593199.

[Dennard et al., 1999] Dennard, R. H., Gaensslen, F. H., Yu, H., Rideout, V. L., Bassous, E., and Leblanc, A. R. (1999). Design Of Ion-implanted MOSFET's with Very Small Physical Dimensions. *Proc. IEEE*, 87(4):668–678. https://doi.org/10.1109/jproc.1999.752522.

[Dennard et al., 1974] Dennard, R. H., Gaensslen, F. H., Yu, H.-N., Rideout, V. L., Bassous, E., and LeBlanc, A. R. (1974). Design of ion-implanted MOSFET's with very small physical dimensions. *IEEE Journal of Solid-State Circuits*, 9(5):256–268. https://doi.org/10.1109/JSSC.1974.1050511.

[Ditzel et al., 2021] Ditzel, D. R. et al. (2021). Accelerating ML Recommendation with over a Thousand RISC-V/Tensor Processors on Esperanto's ET-SoC-1 Chip. In *IEEE Hot Chips Symposium (HCS)*, pages 1–23. https://doi.org/10.1109/HCS52781.2021.9566904.

[Dueck and Scheuer, 1990] Dueck, G. and Scheuer, T. (1990). Threshold accepting: a general purpose optimization algorithm appearing superior to simulated annealing. *Journal of computational physics*, 90(1):161–175. https://doi.org/10.1016/0021-9991(90)90201-B.

[Elmohr et al., 2018] Elmohr, M. A., Eissa, A. S., Ibrahim, M., Khamis, M., El-Ashry, S., Shalaby, A., Abdelsalam, M., and El-Kharashi, M. W. (2018). RVNoC: A Framework for Generating RISC-V NoC-Based MPSoC. In *Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP)*, pages 617–621. https://doi.org/10.1109/PDP2018.2018.00103.

[Esmaeilzadeh et al., 2011] Esmaeilzadeh, H., Blem, E. R., Amant, R. S., Sankaralingam, K., and Burger, D. (2011). Dark Silicon and the End of Multicore Scaling. In *ACM International Symposium on Computer Architecture (ISCA)*, pages 365–376. https://doi.org/10.1145/2000064.2000108.

[Esmaeilzadeh et al., 2012] Esmaeilzadeh, H., Blem, E. R., Amant, R. S., Sankaralingam, K., and Burger, D. (2012). Dark Silicon and the End of Multicore Scaling. *IEEE Micro*, 32(3):122–134. https://doi.org/10.1109/MM.2012.17.

[Even-Dar and Mansour, 2001] Even-Dar, E. and Mansour, Y. (2001). Learning Rates for Q-Learning. In *European Conference on Computational Learning Theory (EuroCOLT)*, page 589–604. https://doi.org/10.1007/3-540-44581-1_39.

[Even-Dar and Mansour, 2003] Even-Dar, E. and Mansour, Y. (2003). Learning Rates for Q-learning. *J. Mach. Learn. Res*, 5:1–25. http://jmlr.org/papers/v5/evendar03a.html.

[Fang, 2021] Fang, G. Y. L. (2021). Instruction-Level Power Consumption Simulator for Modeling Simple Timing and Power Side Channels in a 32-bit RISC-V Micro-Processor. Master's thesis, Massachusetts Institute of Technology (MIT). 140p.

[FreeRTOS, 2022] FreeRTOS (2022). Real-time Operating System for Microcontrollers. http://www.freertos.org/.

[Gnad et al., 2015] Gnad, D., Shafique, M., Kriebel, F., Rehman, S., Sun, D., and Henkel, J. (2015). Hayat: harnessing dark silicon and variability for aging deceleration and balancing. In *ACM/IEEE Design Automation Conference (DAC)*, pages 180:1–180:6. https://doi.org/10.1145/2744769.2744849.

[Gou et al., 2018] Gou, C., Benoit, A., Chen, M., Marchal, L., and Wei, T. (2018). Reliability-Aware Energy Optimization for Throughput-Constrained Applications on MPSoC. In *IEEE International Conference on Parallel and Distributed Systems (ICPADS)*, pages 1–10. https://doi.org/10.1109/PADSW.2018.8644620.

[Haghbayan et al., 2015] Haghbayan, M.-H., Kanduri, A., Rahmani, A.-M., Liljeberg, P., Jantsch, A., and Tenhunen, H. (2015). Mappro: Proactive runtime mapping for dynamic workloads by quantifying ripple effect of applications on networks-on-chip. In *IEEE/ACM International Symposium on Networks-on-Chip (NOCS)*, pages 26:1–26:8. https://doi.org/10.1145/2786572.2786589.

[Haghbayan et al., 2023] Haghbayan, M. H., Miele, A., Mutlu, O., and Plosila, J. (2023). Run-Time Resource Management in CMPs Handling Multiple Aging Mechanisms. *IEEE Trans. Computers*, 72(10):2872–2887. https://doi.org/10.1109/TC.2023.3272800.

[Haghbayan et al., 2016] Haghbayan, M.-H., Miele, A., Rahmani, A. M., Liljeberg, P., and Tenhunen, H. (2016). A lifetime-aware runtime mapping approach for many-core systems in the dark silicon era. In *IEEE Design, Automation Test in Europe Conference (DATE)*, pages 854–857. https://ieeexplore.ieee.org/document/7459428/.

[Haghbayan et al., 2020a] Haghbayan, M. H., Miele, A., Zou, Z., Tenhunen, H., and Plosila, J. (2020a). Thermal-Cycling-aware Dynamic Reliability Management in Many-Core System-on-Chip. In *IEEE Design, Automation Test in Europe Conference (DATE)*, pages 1229–1234. https://doi.org/10.23919/DATE48585.2020.9116325.

[Haghbayan et al., 2020b] Haghbayan, M.-H., Miele, A., Zou, Z., Tenhunen, H., and Plosila, J. (2020b). Thermal-cycling-aware dynamic reliability management in many-core system-on-chip. In *IEEE Design, Automation Test in Europe Conference (DATE)*, pages 1229–1234. https://doi.org/10.23919/DATE48585.2020.9116325.

[Haghbayan et al., 2014] Haghbayan, M.-H., Rahmani, A.-M., Weldezion, A. Y., Liljeberg, P., Plosila, J., Jantsch, A., and Tenhunen, H. (2014). Dark silicon aware power management for manycore systems under dynamic workloads. In *IEEE International Conference on Computer Design (ICCD)*, pages 509–512. https://doi.org/10.1109/ICCD.2014.6974729.

[Hartigan and Wong, 1979] Hartigan, J. A. and Wong, M. A. (1979). Algorithm AS 136: A k-means clustering algorithm. *Journal of the royal statistical society. series c (applied statistics)*, 28(1):100–108. https://doi.org/10.2307/2346830.

[Heirman et al., 2012] Heirman, W., Carlson, T., and Eeckhout, L. (2012). Sniper: Scalable and accurate parallel multi-core simulation. In *ACM Conference on High Performance Computing Networking, Storage and Analysis (SC)*, pages 91–94. https://doi.org/10.1145/2063384.2063454.

[Hill and Marty, 2008] Hill, M. D. and Marty, M. R. (2008). Amdahl's Law in the Multicore Era. *Computer*, 41(7):33–38. https://doi.org/10.1109/MC.2008.209.

[Hoffmann et al., 2013] Hoffmann, H., Maggio, M., Santambrogio, M. D., Leva, A., and Agarwal, A. (2013). A generalized software framework for accurate and efficient management of performance goals. In *International Conference on Embedded Software (EMSOFT)*, pages 19:1–19:10. https://doi.org/10.1109/EMSOFT.2013.6658597.

[Hotfilter et al., 2021] Hotfilter, T., Höfer, J., Kreß, F., Kempf, F., and Becker, J. (2021). FLECSim-SoC: A Flexible End-to-End Co-Design Simulation Framework for System on

Chips. In *International System-on-Chip Conference (SOCC)*, pages 83–88. https://doi.org/10.1109/SOCC52499.2021.9739212.

[Huang et al., 2006] Huang, W., Ghosh, S., Velusamy, S., Sankaranarayanan, K., Skadron, K., and Stan, M. R. (2006). HotSpot: A Compact Thermal Modeling Methodology for Early-Stage VLSI Design. *IEEE Trans. Very Large Scale Integr. Syst*, 14(5):501–513. https://doi.org/10.1109/TVLSI.2006.876103.

[Huang et al., 2014] Huang, X., Yu, T., Sukharev, V., and Tan, S. X.-D. (2014). Physics-based electromigration assessment for power grid networks. In *ACM/IEEE Design Automation Conference (DAC)*, pages 80:1–80:6. https://doi.org/10.1145/2593069.2593180.

[Imperas, 2021] Imperas (2021). Open Virtual Platforms - the source of Fast Processor Models & Platforms. http://www.ovpworld.org/.

[Jantsch et al., 2017] Jantsch, A., Dutt, N., and Rahmani, A. M. (2017). Self-awareness in systems on chip—a survey. *IEEE Design & Test*, 34(6):8–26. https://doi.org/10.1109/MDAT.2017.2757143.

[JEDEC, 2016] JEDEC (2016). Failure mechanisms and models for semiconductor devices. Technical report, Committee for Wafer-level Reliability.

[Jiang et al., 2013] Jiang, N., Becker, D. U., Michelogiannakis, G., Balfour, J. D., Towles, B., Shaw, D. E., Kim, J., and Dally, W. J. (2013). A detailed and flexible cycle-accurate Network-on-Chip simulator. In *IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 86–96. https://doi.org/10.1109/ISPASS.2013.6557149.

[Kamaleldin and Göhringer, 2021] Kamaleldin, A. and Göhringer, D. (2021). Design For Agility: A Modular Reconfigurable Platform for Heterogeneous Many-Core Architectures. In *International Conference on Field-Programmable Logic and Applications (FPL)*, pages 265–266. https://doi.org/10.1109/FPL53798.2021.00050.

[Karkar et al., 2022] Karkar, A., Dahir, N., Mak, T., and Tong, K.-F. (2022). Thermal and performance efficient on-chip surface-wave communication for many-core systems in dark silicon era. *ACM Journal on Emerging Technologies in Computing Systems*, 18(3):49:1–49:18. https://doi.org/10.1145/3501771.

[Khamis et al., 2022] Khamis, M., El-Ashry, S., Abdelsalam, M., El-Kharashi, M. W., and Shalaby, A. (2022). Emulation and verification framework for MPSoC based on NoC and RISC-V. *Des. Autom. Embed. Syst*, 26(3):133–159. https://doi.org/10.1007/s10617-022-09265-1.

[Kim et al., 2017] Kim, T., Sun, Z., Chen, H.-B., Wang, H., and Tan, S. X.-D. (2017). Energy and lifetime optimizations for dark silicon manycore microprocessor considering both hard and soft errors. *IEEE Transactions on Very Large Scale Integration Systems*, 25(9):2561–2574. https://doi.org/10.1109/TVLSI.2017.2707401.

[Kim et al., 2020] Kim, Y. G., Kim, M., Kong, J., and Chung, S. W. (2020). An Adaptive Thermal Management Framework for Heterogeneous Multi-Core Processors. *IEEE Trans. Computers*, 69(6):894–906. https://doi.org/10.1109/TC.2020.2970062.

[Kong et al., 2012] Kong, J., Chung, S. W., and Skadron, K. (2012). Recent thermal management techniques for microprocessors. *ACM Computing Surveys*, 44(3):13:1–13:42. https://doi.org/10.1145/2187671.2187675.

[Krishnan et al., 2022] Krishnan, S., Lam, M., Chitlangia, S., Wan, Z., Barth-Maron, G., Faust, A., and Reddi, V. J. (2022). QuaRL: Quantization for Fast and Environmentally Sustainable Reinforcement Learning. *n Transactions on Machine Learning Research*, 2022:1–23. https://openreview.net/forum?id=xwWsiFmUEs.

[Kurth et al., 2018] Kurth, A., Capotondi, A., Vogel, P., Benini, L., and Marongiu, A. (2018). HERO: an open-source research platform for HW/SW exploration of heterogeneous manycore systems. In *Workshop on AutotuniNg and aDaptivity AppRoaches for Energy Efficient Systems (ANDARE@PACT)*, pages 5:1–5:6. https://doi.org/10.1145/3295816.3295821.

[Lala, 1996] Lala, P. K. (1996). *Practical digital logic design and testing*. Prentice-Hall, Inc, 1st edition. 420p.

[Li et al., 2019] Li, B., Wang, X., Singh, A. K., and Mak, T. (2019). On Runtime Communication and Thermal-Aware Application Mapping and Defragmentation in 3D NoC Systems. *IEEE Transactions on Parallel and Distributed Systems*, 30(12):2775–2789. https://doi.org/10.1109/TPDS.2019.2921542.

[Li et al., 2018] Li, M., Liu, W., Yang, L., Chen, P., and Chen, C. (2018). Chip temperature optimization for dark silicon many-core systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 37(5):941–953. https://doi.org/10.1109/TCAD.2017.2740306.

[Li et al., 2015] Li, M., Yi, J., Liu, W., Zhang, W., Yang, L., and Sha, E. H.-M. (2015). An efficient technique for chip temperature optimization of multiprocessor systems in the dark silicon era. In *IEEE International Conference on High Performance Computing and Communications (HPCC)*, pages 688–693. https://doi.org/10.1109/HPCC-CSS-ICESS.2015.59.

[Li et al., 2004] Li, P., Pileggi, L. T., Asheghi, M., and Chandra, R. (2004). Efficient full-chip thermal modeling and analysis. In *IEEE International Conference on Computer-Aided Design (ICCAD)*, pages 319–326. https://doi.org/10.1109/ICCAD.2004.1382594.

[Li et al., 2009] Li, S., Ahn, J. H., Strong, R. D., Brockman, J. B., Tullsen, D. M., and Jouppi, N. P. (2009). McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures. In *IEEE International Microarchitecture Symposium (MICRO)*, pages 469–480. https://doi.org/10.1145/1669112.1669172.

[Li et al., 2011] Li, S., Chen, K., Ahn, J. H., Brockman, J. B., and Jouppi, N. P. (2011). CACTI-P: Architecture-level modeling for SRAM-based structures with advanced leakage reduction techniques. In *IEEE International Conference on Computer-Aided Design (ICCAD)*, pages 694–701. https://doi.org/10.1109/ICCAD.2011.6105405.

[Li, 2023] Li, S. E. (2023). *Reinforcement learning for sequential decision and optimal control*. Springer, 1st edition. 492p.

[Lienhard, 2006] Lienhard, John H. IV; Lienhard, J. H. V. (2006). *A heat transfer textbook*. Phlogistron, 3th edition. 760p.

[Lin et al., 2014] Lin, X., Wang, Y., Bogdan, P., Chang, N., and Pedram, M. (2014). Reinforcement learning based power management for hybrid electric vehicles. In *IEEE International Conference on Computer-Aided Design (ICCAD)*, pages 32–38. https://doi.org/10.1109/ICCAD.2014.7001326.

[Liu et al., 2018] Liu, W., Yang, L., Jiang, W., Feng, L., Guan, N., Zhang, W., and Dutt, N. D. (2018). Thermal-aware Task Mapping on Dynamically Reconfigurable Network-on-Chip based Multiprocessor System-on-Chip. *IEEE Transactions on Computers*, 67(12):1818 – 1834. https://doi.org/10.1109/TC.2018.2844365.

[Liu et al., 2019] Liu, W., Yi, J., Li, M., Chen, P., and Yang, L. (2019). Energy-Efficient Application Mapping and Scheduling for Lifetime Guaranteed MPSoCs. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst*, 38(1):1–14. https://doi.org/10.1109/TCAD.2018.2801242.

[Lopes et al., 2021] Lopes, G., Weber, I. I., Marcon, C. A. M., and Moraes, F. G. (2021). Chronos: An Abstract NoC-based Manycore with Preserved Temporal and Spatial Traffic Distribution. In *IEEE Latin America Symposium on Circuits and System (LASCAS)*, pages 1–4. https://doi.org/10.1109/LASCAS51355.2021.9459124.

[Mack et al., 2020] Mack, J., Kumbhare, N., Krishnakumar, A., Ogras, Ü. Y., and Akoglu, A. (2020). User-Space Emulation Framework for Domain-Specific SoC Design. In *International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 44–53. https://doi.org/10.1109/IPDPSW50202.2020.00016.

[Martin et al., 2005] Martin, M. M. K., Sorin, D. J., Beckmann, B. M., Marty, M. R., Xu, M., Alameldeen, A. R., Moore, K. E., Hill, M. D., and Wood, D. A. (2005). Multifacet's general

execution-driven multiprocessor simulator (GEMS) toolset. *SIGARCH Comput. Archit. News*, 33(4):92–99. https://doi.org/10.1145/1105734.1105747.

[Martins et al., 2019] Martins, A. L., da Silva, A. H. L., Rahmani, A. M., Dutt, N., and Moraes, F. G. (2019). Hierarchical adaptive Multi-objective resource management for many-core systems. *Journal of Systems Architecture*, 97:416–427. https://doi.org/10.1016/j.sysarc.2019.01.006.

[Martins, 2018] Martins, A. L. d. M. (2018). *Multi-Objective Resource Management for Many-core Systems*. PhD thesis, Pontifical Catholic University of Rio Grande do Sul, PPGCC. 147p.

[Martins et al., 2014a] Martins, A. L. M., Silva, D. R. G., Castilhos, G. M., Monteiro, T. M., and Moraes, F. G. (2014a). A method for NoC-based MPSoC energy consumption estimation. In *IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, pages 427–430. https://doi.org/10.1109/ICECS.2014.7050013.

[Martins et al., 2014b] Martins, A. L. M., Silva, D. R. G., Castilhos, G. M., Monteiro, T. M., and Moraes, F. G. (2014b). A method for NoC-based MPSoC energy consumption estimation. In *IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, pages 427–430. https://doi.org/10.1109/ICECS.2014.7050013.

[Merchant et al., 2021] Merchant, F., Sisejkovic, D., Reimann, L. M., Yasotharan, K., Grass, T., and Leupers, R. (2021). ANDROMEDA: An FPGA Based RISC-V MPSoC Exploration Framework. In *International Conference on VLSI Design (VLSID)*, pages 270–275. https://doi.org/10.1109/VLSID51830.2021.00051.

[Miller et al., 2010] Miller, J. E., Kasture, H., Kurian, G., III, C. G., Beckmann, N., Celio, C., Eastep, J., and Agarwal, A. (2010). Graphite: A distributed parallel simulator for multicores. In *IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 1–12. https://doi.org/10.1109/HPCA.2010.5416635.

[Mohammed et al., 2020] Mohammed, M. S., Al-Kubati, A. A., Paraman, N., Ab Rahman, A. A.-H., and Marsono, M. (2020). DTaPO: Dynamic thermal-aware performance optimization for dark silicon many-core systems. *Electronics*, 9(11):1–18. https://doi.org/10.3390/electronics9111980.

[Monemi et al., 2017] Monemi, A., Tang, J. W., Palesi, M., and Marsono, M. N. (2017). ProNoC: A low latency network-on-chip based many-core system-on-chip prototyping platform. *Microprocess. Microsystems*, 54:60–74. https://doi.org/10.1016/j.micpro.2017.08.007.

[Moore, 1965] Moore, G. (1965). Cramming more Components onto Integrated Circuits. *Electronics*, 38(8):1–6.

[Moore, 1998] Moore, G. E. (1998). Cramming More Components Onto Integrated Circuits. *Proc. IEEE*, 86(1):82–85. https://doi.org/10.1109/jproc.1998.658762.

[Namazi et al., 2019] Namazi, A., Safari, S., Mohammadi, S., and Abdollahi, M. (2019). SORT: Semi Online Reliable Task Mapping for Embedded Multi-Core Systems. *ACM Trans. Model. Perform. Evaluation Comput. Syst*, 4(2):11:1–11:25. https://doi.org/10.1145/3322899.

[Ost et al., 2009] Ost, L., Guindani, G. M., Indrusiak, L. S., Reinbrecht, C., da Rosa, T. R., and Moraes, F. (2009). A high abstraction, high accuracy power estimation model for networks-on-chip. In *IEEE Symposium on Integrated Circuits and Systems Design (SBCCI)*, pages 1–6. https://doi.org/10.1145/1601896.1601936.

[Pagani et al., 2015a] Pagani, S., Chen, J., Shafique, M., and Henkel, J. (2015a). MatEx: efficient transient and peak temperature computation for compact thermal models. In *IEEE Design, Automation Test in Europe Conference (DATE)*, pages 1515–1520. https://doi.org/10.7873/DATE.2015.0328.

[Pagani et al., 2017] Pagani, S., Khdr, H., Chen, J., Shafique, M., Li, M., and Henkel, J. (2017). Thermal Safe Power (TSP): Efficient Power Budgeting for Heterogeneous Many-core Systems in Dark Silicon. *IEEE Trans. Computers*, 66(1):147–162. https://doi.org/10.1109/TC.2016.2564969.

[Pagani et al., 2014] Pagani, S., Khdr, H., Munawar, W., Chen, J.-J., Shafique, M., Li, M., and Henkel, J. (2014). TSP: Thermal Safe Power - Efficient power budgeting for many-core systems in dark silicon. In *IEEE International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, pages 1–10. https://doi.org/10.1145/2656075.2656103.

[Pagani et al., 2015b] Pagani, S., Shafique, M., Khdr, H., Chen, J.-J., and Henkel, J. (2015b). seBoost: selective boosting for heterogeneous manycores. In *IEEE International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, pages 104–113. https://doi.org/10.1109/CODESISSS.2015.7331373.

[Parry et al., 1998] Parry, J., Rosten, H., and Kromann, G. (1998). The development of component-level thermal compact models of a C4/CBGA interconnect technology: the Motorola PowerPC 603 and PowerPC 604 RISC microprocessors. *IEEE Transactions on Components, Packaging, and Manufacturing Technology*, 21(1). https://doi.org/10.1109/95.679039.

[Passage et al., 2019] Passage, J. M., Azhari, N., and Lloyd, J. R. (2019). Stress Migration Followed by Electromigration Reliability Testing. In *IEEE International Reliability Physics Symposium IRPS*, pages 1–5. https://doi.org/10.1109/IRPS.2019.8720473.

[Pecht et al., 2017] Pecht, M. G., Radojcic, R., and Rao, G. (2017). *Guidebook for managing silicon chip reliability*. CRC press, 1st edition. 224p.

[Pourmohseni et al., 2022] Pourmohseni, B., Wildermann, S., Smirnov, F., Meyer, P. E., and Teich, J. (2022). Task Migration Policy for Thermal-Aware Dynamic Performance Optimization in Many-Core Systems. *IEEE Access*, 10:33787–33802. https://doi.org/10.1109/ACCESS.2022.3162617.

[Powell, 2022] Powell, W. B. (2022). *Reinforcement Learning and Stochastic Optimization: A Unified Framework for Sequential Decisions*. Wiley, 1st edition. 1136p.

[Rahimipour et al., 2020] Rahimipour, S., Flayyih, W. N., Kamsani, N. A., Hashim, S. J., Stan, M. R., and Rokhani, F. Z. (2020). Low-Power, Highly Reliable Dynamic Thermal Management by Exploiting Approximate Computing. *IEEE Trans. Very Large Scale Integr. Syst*, 28(10):2210–2222. https://doi.org/10.1109/TVLSI.2020.3012626.

[Rahmani et al., 2016] Rahmani, A. M., Liljeberg, P., Hemani, A., Jantsch, A., and Tenhunen, H. (2016). *The Dark Side of Silicon*. Springer, 1st edition.

[Ramachandran et al., 2008] Ramachandran, P., Adve, S. V., Bose, P., and Rivers, J. A. (2008). Metrics for Architecture-Level Lifetime Reliability Analysis. In *IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 202–212. https://doi.org/10.1109/ISPASS.2008.4510752.

[Ranieri et al., 2015] Ranieri, J., Vincenzi, A., Chebira, A., Atienza, D., and Vetterli, M. (2015). Near-Optimal Thermal Monitoring Framework for Many-Core Systems-on-Chip. *IEEE Trans. Computers*, 64(11):3197–3209. https://doi.org/10.1109/TC.2015.2395423.

[Ranjbar et al., 2019] Ranjbar, B., Nguyen, T. D. A., Ejlali, A., and Kumar, A. (2019). Online Peak Power and Maximum Temperature Management in Multi-core Mixed-Criticality Embedded Systems. In *Euromicro Conference on Digital System Design (DSD)*, pages 546–553. https://doi.org/10.1109/DSD.2019.00084.

[Rathore, 2020] Rathore, V. (2020). *Scalable Techniques for Extending Lifetime Reliability of Manycore Systems*. PhD thesis, School of Computer Science & Engineering, Nanyang Technological University. 254p.

[Rathore et al., 2018] Rathore, V., Chaturvedi, V., Singh, A. K., Srikanthan, T., Rohith, R., Lam, S.-K., and Shaflque, M. (2018). HiMap: A hierarchical mapping approach for enhancing lifetime reliability of dark silicon manycore systems. In *IEEE Design, Automation Test in Europe Conference (DATE)*, pages 991–996. https://doi.org/10.23919/DATE.2018.8342153.

[Rathore et al., 2019a] Rathore, V., Chaturvedi, V., Singh, A. K., Srikanthan, T., and Shafique, M. (2019a). LifeGuard: A reinforcement learning-based task mapping strategy for performance-centric aging management. In *ACM/IEEE Design Automation Conference (DAC)*, pages 1–6. https://doi.org/10.1145/3316781.3317849.

[Rathore et al., 2019b] Rathore, V., Chaturvedi, V., Singh, A. K., Srikanthan, T., and Shafique, M. (2019b). Towards Scalable Lifetime Reliability Management for Dark Silicon Manycore Systems. In *IEEE International Symposium on On-Line Testing and Robust System Design (IOLTS)*, pages 204–207. https://doi.org/10.1109/IOLTS.2019.8854454.

[Rathore et al., 2021] Rathore, V., Chaturvedi, V., Singh, A. K., Srikanthan, T., and Shafique, M. (2021). Longevity Framework: Leveraging Online Integrated Aging-Aware Hierarchical Mapping and VF-Selection for Lifetime Reliability Optimization in Manycore Processors. *IEEE Trans. Computers*, 70(7):1106–1119. https://doi.org/10.1109/TC.2020.3006571.

[Rathore et al., 2016a] Rathore, V., Chaturvedi, V., and Srikanthan, T. (2016a). Performance constraint-aware task mapping to optimize lifetime reliability of manycore systems. In *ACM Great Lakes Symposium on VLSI (GLVLSI)*, pages 377–380. https://doi.org/10.1145/2902961.2902996.

[Rathore et al., 2016b] Rathore, V., Chaturvedi, V., and Srikanthan, T. (2016b). Performance constraint-aware task mapping to optimize lifetime reliability of manycore systems. In *ACM Great Lakes Symposium on VLSI (GLVLSI)*, pages 377–380. https://doi.org/10.1145/2902961.2902996.

[Real et al., 2016] Real, M. M., Wehner, P., Rettkowski, J., Migliore, V., Lapotre, V., Göhringer, D., and Gogniat, G. (2016). MPSoCSim extension: An OVP simulator for the evaluation of cluster-based multi and many-core architectures. In *International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation (SAMOS)*, pages 342–347. https://doi.org/10.1109/SAMOS.2016.7818370.

[Ris-Ala, 2023] Ris-Ala, R. (2023). *Fundamentals of Reinforcement Learning*. Springer, 1st edition. 88p.

[Rohith et al., 2018] Rohith, R., Rathore, V., Chaturvedi, V., Singh, A. K., Thambipillai, S., and Lam, S.-K. (2018). LifeSim: A lifetime reliability simulator for manycore systems. In *IEEE Computing and Communication Workshop and Conference (CCWC)*, pages 375–381. https://doi.org/10.1109/CCWC.2018.8301711.

[Rovinski et al., 2019] Rovinski, A., Zhao, C., Al-Hawaj, K., Gao, P., Xie, S., Torng, C., Davidson, S., Amarnath, A., Vega, L., Veluri, B., Rao, A., Ajayi, T., Puscar, J., Dai, S., Zhao, R., Richmond, D., Zhang, Z., Galton, I., Batten, C., Taylor, M. B., and Dreslinski, R. G. (2019). Evaluating Celerity: A 16-nm 695 Giga-RISC-V Instructions/s Manycore

Processor With Synthesizable PLL. *IEEE Solid-State Circuits Letters*, 2(12):289–292. https://doi.org/10.1109/LSSC.2019.2953847.

[Ruaro et al., 2019] Ruaro, M., Caimi, L. L., Fochi, V., and Moraes, F. G. (2019). Memphis: a framework for heterogeneous many-core SoCs generation and validation. *Des. Autom. Embed. Syst*, 23(3-4):103–122. https://doi.org/10.1007/s10617-019-09223-4.

[Ruaro and J. M. Martin, 2022] Ruaro, M. and J. M. Martin, K. (2022). ManyGUI: A Graphical Tool to Accelerate Many-Core Debugging Through Communication, Memory, and Energy Profiling. In *System Engineering for Constrained Embedded Systems (DroneSE and RAPIDO)*, page 39–46. https://doi.org/10.1145/3522784.3522791.

[Sabry, 2003] Sabry, M.-N. (2003). Compact thermal models for electronic systems. *IEEE Transactions on Components and Packaging Technologies*, 26(1):179–185. https://doi.org/10.1109/TCAPT.2002.808009.

[Sahoo et al., 2021] Sahoo, S. S., Ranjbar, B., and Kumar, A. (2021). Reliability-aware resource management in multi-/many-core systems: A perspective paper. *Journal of Low Power Electronics and Applications*, 11(1):7. https://doi.org/10.3390/jlpea11010007.

[Sahoo et al., 2019] Sahoo, S. S., Veeravalli, B., and Kumar, A. (2019). A Hybrid Agent-based Design Methodology for Dynamic Cross-layer Reliability in Heterogeneous Embedded Systems. In *ACM/IEEE Design Automation Conference (DAC)*, page 38. https://doi.org/10.1145/3316781.3317746.

[Savas et al., 2020] Savas, S., Ul-Abdin, Z., and Nordström, T. (2020). A framework to generate domain-specific manycore architectures from dataflow programs. *Microprocess. Microsystems*, 72. https://doi.org/10.1016/j.micpro.2019.102908.

[Sha et al., 2018a] Sha, S., Wen, W., Ren, S., and Quan, G. (2018a). M-Oscillating: Performance Maximization on Temperature-Constrained Multi-Core Processors. *IEEE Transactions on Parallel and Distributed Systems*, 29(11):2528–2539. https://doi.org/10.1109/TPDS.2018.2835474.

[Sha et al., 2018b] Sha, S., Wen, W., Ren, S., and Quan, G. (2018b). M-Oscillating: Performance Maximization on Temperature-Constrained Multi-Core Processors. *IEEE Transactions on Parallel and Distributed Systems*, 29(11):2528–2539. https://doi.org/10.1109/TPDS.2018.2835474.

[Shivakumar et al., 2002] Shivakumar, P., Kistler, M., Keckler, S. W., Burger, D., and Alvisi, L. (2002). Modeling the Effect of Technology Trends on the Soft Error Rate of Combinational Logic. In *IEEE International Conference on Dependable Systems and Networks (DSN)*, pages 389–398. https://doi.org/10.1109/DSN.2002.1028924.

[Silva et al., 2020] Silva, A. L. d., del Mestre Martins, A. L., and Moraes, F. G. (2020). Mapping and Migration Strategies for Thermal Management in Many-Core Systems. In *IEEE Symposium on Integrated Circuits and Systems Design (SBCCI)*, pages 1–6. https://doi.org/10.1109/SBCCI50935.2020.9189933.

[Silva et al., 2019] Silva, A. L. d., Martins, A. L. d. M., and Moraes, F. G. (2019). Fine-grain Temperature Monitoring for many-core Systems. In *IEEE Symposium on Integrated Circuits and Systems Design (SBCCI)*, pages 1–6. https://doi.org/10.1145/3338852.3339841.

[Silva et al., 2021] Silva, A. L. d., Weber, I. I., Martins, A. L. d. M., and Moraes, F. G. (2021). Hardware Accelerator for Runtime Temperature Estimation in Many-Cores. *IEEE Design & Test*, 38(4):62–69. https://doi.org/10.1109/MDAT.2021.3068914.

[Singh et al., 2011] Singh, P., Karl, E., Blaauw, D., and Sylvester, D. (2011). Compact degradation sensors for monitoring NBTI and oxide degradation. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 20(9):1645–1655. https://doi.org/10.1109/TVLSI.2011.2161784.

[Spiliopoulos et al., 2011] Spiliopoulos, V., Kaxiras, S., and Keramidas, G. (2011). Green governors: A framework for Continuously Adaptive DVFS. In *IEEE International Green Computing Conference and Workshops (IGCC)*, pages 1–8. https://doi.org/10.1109/IGCC.2011.6008552.

[Srinivasan, 2006] Srinivasan, J. (2006). *Lifetime Reliability Aware Microprocessors*. PhD thesis, University of Illinois. 92p.

[Srinivasan et al., 2003] Srinivasan, J., Adve, S. V., Bose, P., Rivers, J., and Hu, C.-K. (2003). RAMP: A model for reliability aware microprocessor design. Technical report, IBM.

[Srinivasan et al., 2005] Srinivasan, J., Adve, S. V., Bose, P., and Rivers, J. A. (2005). Exploiting Structural Duplication for Lifetime Reliability Enhancement. In *ACM International Symposium on Computer Architecture (ISCA)*, pages 520–531. https://doi.org/10.1109/ISCA.2005.28.

[Streetman et al., 2016] Streetman, B. G., Banerjee, S., et al. (2016). *Solid State Electronic Devices*. Pearson, 7th edition. 596p.

[Strong et al., 2009] Strong, A. W., Wu, E. Y., Vollertsen, R.-P., Sune, J., La Rosa, G., Sullivan, T. D., and Rauch III, S. E. (2009). *Reliability wear-out mechanisms in advanced CMOS technologies*. Wiley-Blackwell, 1st edition. 624p.

[Su et al., 2003] Su, H., Liu, F., Devgan, A., Acar, E., and Nassif, S. R. (2003). Full chip leakage estimation considering power supply and temperature variations. In *IEEE Symposium on Low Power Electronics and Design (ISLPED)*, pages 78–83. https://doi.org/10.1145/871506.871529.

[Suter, 2013] Suter, F. (2013). Daggen: A synthetic task graph generator. https://github.com/frs69wq/daggen.

[Sutter, 2005] Sutter, H. (2005). The free lunch is over: A fundamental turn toward concurrency in software. *Dr. Dobb's journal*, 30(3):202–210. https://www.cs.utexas.edu/~lin/cs380p/Free_Lunch.pdf.

[Trivedi, 2016] Trivedi, K. S. (2016). *Probability and Statistics with Reliability, Queuing and Computer Science Applications*. John Wiley & Sons, 2nd edition. 880p.

[Tsitsiklis, 1994] Tsitsiklis, J. N. (1994). Asynchronous Stochastic Approximation and Q-Learning. *Mach. Learn*, 16(3):185–202. https://doi.org/10.1007/BF00993306.

[Uddin, 2017] Uddin, M. I. (2017). One-IPC high-level simulation of microthreaded many-core architectures. *Int. J. High Perform. Comput. Appl*, 31(2):152–162. https://doi.org/10.1177/1094342015584495.

[Uhlendorf et al., 2021] Uhlendorf, R., Silva, E., Viel, F., and Zeferino, C. (2021). An MPI-based MPSoC Platform in FPGA. *IEEE Latin America Transactions*, 19(4):697–705. https://doi.org/10.1007/s10617-022-09265-1.

[Wan et al., 2020] Wan, B., Wang, Y., Su, Y., and Fu, G. (2020). Reliability evaluation of multi-mechanism failure for semiconductor devices using physics-of-failure technique and maximum entropy principle. *IEEE Access*, 8:188154–188170. https://doi.org/10.1109/ACCESS.2020.3031022.

[Wang et al., 2019] Wang, L., Lv, P., Liu, L., Han, J., Leung, H., Wang, X., Yin, S., Wei, S., and Mak, T. S. T. (2019). A Lifetime Reliability-Constrained Runtime Mapping for Throughput Optimization in Many-Core Systems. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst*, 38(9):1771–1784. https://doi.org/10.1109/TCAD.2018.2855168.

[Wang et al., 2018] Wang, L., Lv, P., Liu, L., Han, J., Leung, H.-F., Wang, X., Yin, S., Wei, S., and Mak, T. (2018). A Lifetime Reliability-Constrained Runtime Mapping for Throughput Optimization in Many-Core Systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 38(9):1771–1784. https://doi.org/10.1109/TCAD.2018.2855168.

[Wang and Chen, 2002] Wang, T. and Chen, C. C. (2002). 3-D Thermal-ADI: a linear-time chip level transient thermal simulator. *IEEE Transactions on Computer-Aided De-*

sign of Integrated Circuits and Systems, 21(12):1434–1445. https://doi.org/10.1109/TCAD.2002.804385.

[Wang et al., 2011] Wang, Y., Xie, Q., Ammari, A. C., and Pedram, M. (2011). Deriving a near-optimal power management policy using model-free reinforcement learning and Bayesian classification. In *ACM/IEEE Design Automation Conference (DAC)*, pages 41–46. https://doi.org/10.1145/2024724.2024735.

[Watkins, 1989] Watkins, C. J. C. H. (1989). *Learning from Delayed Rewards*. PhD thesis, King's College, Cambridge United Kingdom. 234p.

[Wen et al., 2020] Wen, S., Wang, X., Singh, A. K., Jiang, Y., and Yang, M. (2020). Performance optimization of many-core systems by exploiting task migration and dark core allocation. *IEEE Transactions on Computers*, 71(1):92–106. https://doi.org/10.1109/TC.2020.3042663.

[Woo et al., 1995] Woo, S. C., Ohara, M., Torrie, E., Singh, J. P., and Gupta, A. (1995). The SPLASH-2 programs: Characterization and methodological considerations. *ACM SIGARCH computer architecture news*, 23(2):24–36. https://doi.org/10.1145/223982.223990.

[Wu et al., 2002a] Wu, E., Sune, J., Lai, W., Nowak, E., McKenna, J., Vayshenker, A., and Harmon, D. (2002a). Interplay of voltage and temperature acceleration of oxide breakdown for ultra-thin gate oxides. *Solid-State Electronics*, 46(11):1787–1798. https://doi.org/10.1016/S0038-1101(02)00151-X.

[Wu et al., 2002b] Wu, E. Y., Nowak, E. J., Vayshenker, A., Lai, W. L., and Harmon, D. L. (2002b). CMOS scaling beyond the 100-nm node with silicon-dioxide-based gate dielectrics. *IBM Journal of Research and Development*, 46(2.3):287–298. https://doi.org/10.1147/rd.462.0287.

[Xi et al., 2015] Xi, S. L., Jacobson, H., Bose, P., Wei, G.-Y., and Brooks, D. (2015). Quantifying sources of error in McPAT and potential impacts on architectural studies. In *IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 577–589. https://doi.org/10.1109/HPCA.2015.7056064.

[Yang et al., 2017a] Yang, L., Liu, W., Guan, N., Li, M., Chen, P., and Edwin, H. (2017a). Dark silicon-aware hardware-software collaborated design for heterogeneous many-core systems. In *IEEE Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 494–499. https://doi.org/10.1109/ASPDAC.2017.7858371.

[Yang et al., 2017b] Yang, L., Liu, W., Jiang, W., Li, M., Chen, P., and Sha, E. H.-M. (2017b). Fotonoc: A folded torus-like network-on-chip based many-core systems-on-chip in the

dark silicon era. *IEEE Transactions on Parallel and Distributed Systems*, 28(7):1905–1918. https://doi.org/10.1109/TPDS.2016.2643669.

[Yoo et al., 2022] Yoo, J., An, T., Oh, C., Cho, Y., Lee, H., Im, Y., Kim, M., and Kim, M. (2022). Thermal-Aware Optimization of SoC Floorplan with Heterogenous Multi-Cores. In *IEEE Intersociety Conference on Thermal and Thermomechanical Phenomena in Electronic Systems (iTherm)*, pages 1–6. https://doi.org/10.1109/iTherm54085.2022.9899498.

[Yu et al., 2015] Yu, J., Zhou, W., Yang, Y., Zhang, X., and Yu, Z. (2015). Many-Core Processors Granularity Evaluation by Considering Performance, Yield, and Lifetime Reliability. *IEEE Trans. Very Large Scale Integr. Syst*, 23(10):2043–2053. https://doi.org/10.1109/TVLSI.2014.2359076.

[Zafar, 2007] Zafar, S. (2007). A Model for Negative Bias Temperature Instability in Oxide and High κ pFETs. In *IEEE Symposium on VLSI Circuits (VLSI)*, pages 1–5. https://doi.org/10.1109/ICICDT.2007.4299550.

[Zaruba and Benini, 2019] Zaruba, F. and Benini, L. (2019). The Cost of Application-Class Processing: Energy and Performance Analysis of a Linux-Ready 1.7-GHz 64-Bit RISC-V Core in 22-nm FDSOI Technology. *IEEE Trans. Very Large Scale Integr. Syst*, 27(11):2629–2640. https://doi.org/10.1109/TVLSI.2019.2926114.

[Zhang et al., 2023] Zhang, J., Sadiqbatcha, S., and Tan, S. X. (2023). Hot-Trim: Thermal and Reliability Management for Commercial Multicore Processors Considering Workload Dependent Hot Spots. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst*, 42(7):2290–2302. https://doi.org/10.1109/TCAD.2022.3216552.

[Zhang et al., 2018] Zhang, K., Guliani, A., Memik, S. O., Memik, G., Yoshii, K., Sankaran, R., and Beckman, P. H. (2018). Machine Learning-Based Temperature Prediction for Runtime Thermal Management Across System Components. *IEEE Trans. Parallel Distributed Syst*, 29(2):405–419. https://doi.org/10.1109/TPDS.2017.2732951.

[Zhang et al., 2015] Zhang, R., Stan, M. R., and Skadron, K. (2015). Hotspot 6.0: Validation, Acceleration and Extension. Technical report, University of Virginia, Tech. Rep.

[Zhang et al., 2017] Zhang, W., Ji, X., Lu, Y., Wang, H., Chen, H., and Yew, P. (2017). Prophet: A Parallel Instruction-Oriented Many-Core Simulator. *IEEE Trans. Parallel Distributed Syst*, 28(10):2939–2952. https://doi.org/10.1109/TPDS.2017.2700307.

APPENDIX A – LIST OF PUBLICATIONS

Journal Publications

WEBER, IAÇANÃ IANISKI; DAL ZOTTO, ANGELO ELIAS; MORAES, FERNANDO G.
Chronos-V: a Many-core High-level Model with Support for Management Techniques.
ANALOG INTEGRATED CIRCUITS AND SIGNAL PROCESSING, Volume 117, pages 57–71, 2023.
https://link.springer.com/article/10.1007/s10470-023-02190-8

DA SILVA, ALZEMIRO LUCAS; WEBER, IACANA IANISKI; MARTINS, ANDRE LUIS DEL MESTRE; MORAES, FERNANDO G.
Hardware Accelerator for Runtime Temperature Estimation in Many-cores.
IEEE Design & Test, v. 38(4), p. 62-69, 2021.
https://ieeexplore.ieee.org/document/9386103

Conference Publications

WEBER, IAÇANÃ IANISKI; ZANINI, VITOR BALBINOT; MORAES, FERNANDO G.
FLEA - FIT-Aware Heuristic for Application Allocation in Many-Cores based on Q-Learning.
In: SBESC, 2023
https://ieeexplore.ieee.org/document/10324296

WEBER, IACANA IANISKI; DALZOTTO, ANGELO ELIAS; MORAES, FERNANDO G.
A High-level Model to Leverage NoC-based Many-core Research.
In: SBCCI, 2022
https://ieeexplore.ieee.org/document/9893235

SILVA, ALZEMIRO; WEBER, IACANA; MARTINS, ANDRE LUIS DEL MESTRE; MORAES, FERNANDO G.
Reliability Assessment of Many-Core Dynamic Thermal Management.
In: ISCAS, 2022
https://ieeexplore.ieee.org/document/9937286

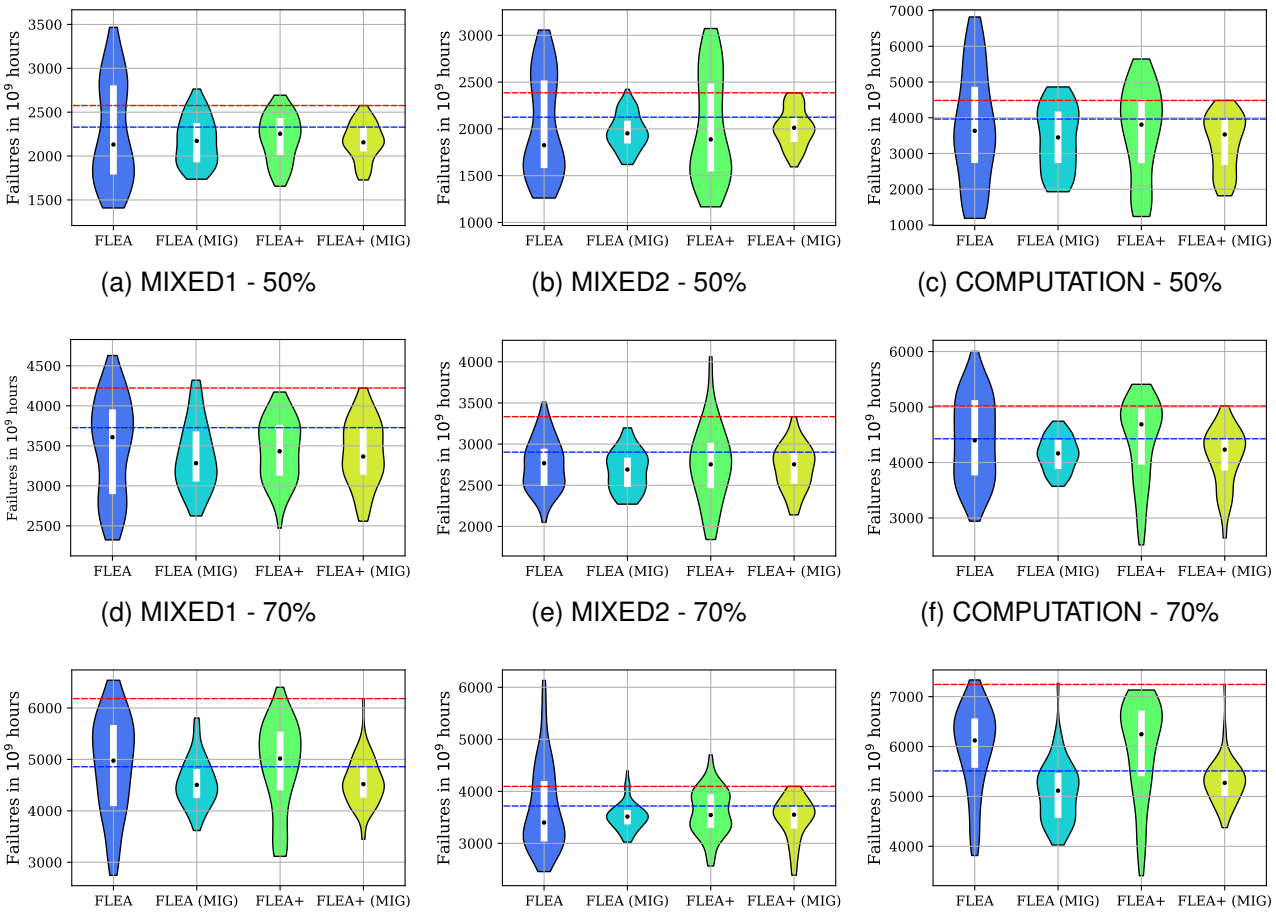
SILVA, ALZEMIRO; WEBER, IACANA; DEL MESTRE MARTINS, ANDRE LUIS; MORAES, FERNANDO G.
Dynamic Thermal Management in Many-Core Systems Leveraged by Abstract Modeling.
In: ISCAS, 2021
https://ieeexplore.ieee.org/document/9401414

LOPES, G.; WEBER, IACANA; MARCON, C.; MORAES, FERNANDO G.
Chronos: an Abstract NoC-based Manycore with Preserved Temporal and Spatial Traffic Distribution.
In: LASCAS, 2021
http://dx.doi.org/10.1109/LASCAS51355.2021.9459124

WEBER, IACANA; DE OLIVEIRA, LEONARDO; CARARA, EVERTON; MORAES, FERNANDO G.
Reducing NoC Energy Consumption Exploring Asynchronous End-to-end GALS Communication.
In: SBCCI, 2020
https://ieeexplore.ieee.org/document/8533228
not related to the Thesis subject

WEBER, IACANA; MARCHEZAN, GEANINNE; CAIMI, LUCIANO; MARCON, CESAR; MORAES, FERNANDO G.
Open-Source NoC-Based Many-Core for Evaluating Hardware Trojan Detection Methods.
In: ISCAS, 2020
https://ieeexplore.ieee.org/document/9180578
not related to the Thesis subject

APPENDIX B – FLEA AND FLEA+ FIT COMPARISON



(a) MIXED1 - 50% (b) MIXED2 - 50% (c) COMPUTATION - 50%

(d) MIXED1 - 70% (e) MIXED2 - 70% (f) COMPUTATION - 70%

(g) MIXED1 - 90% (h) MIXED2 - 90% (i) COMPUTATION - 90%

Figure B.1: FIT comparison between FLEA and FLEA+ in the 8x8 manycore. The blue dotted line is the FLEA+ with migration third quartile. The red dotted line is the FLEA+ with migration maximum FIT.

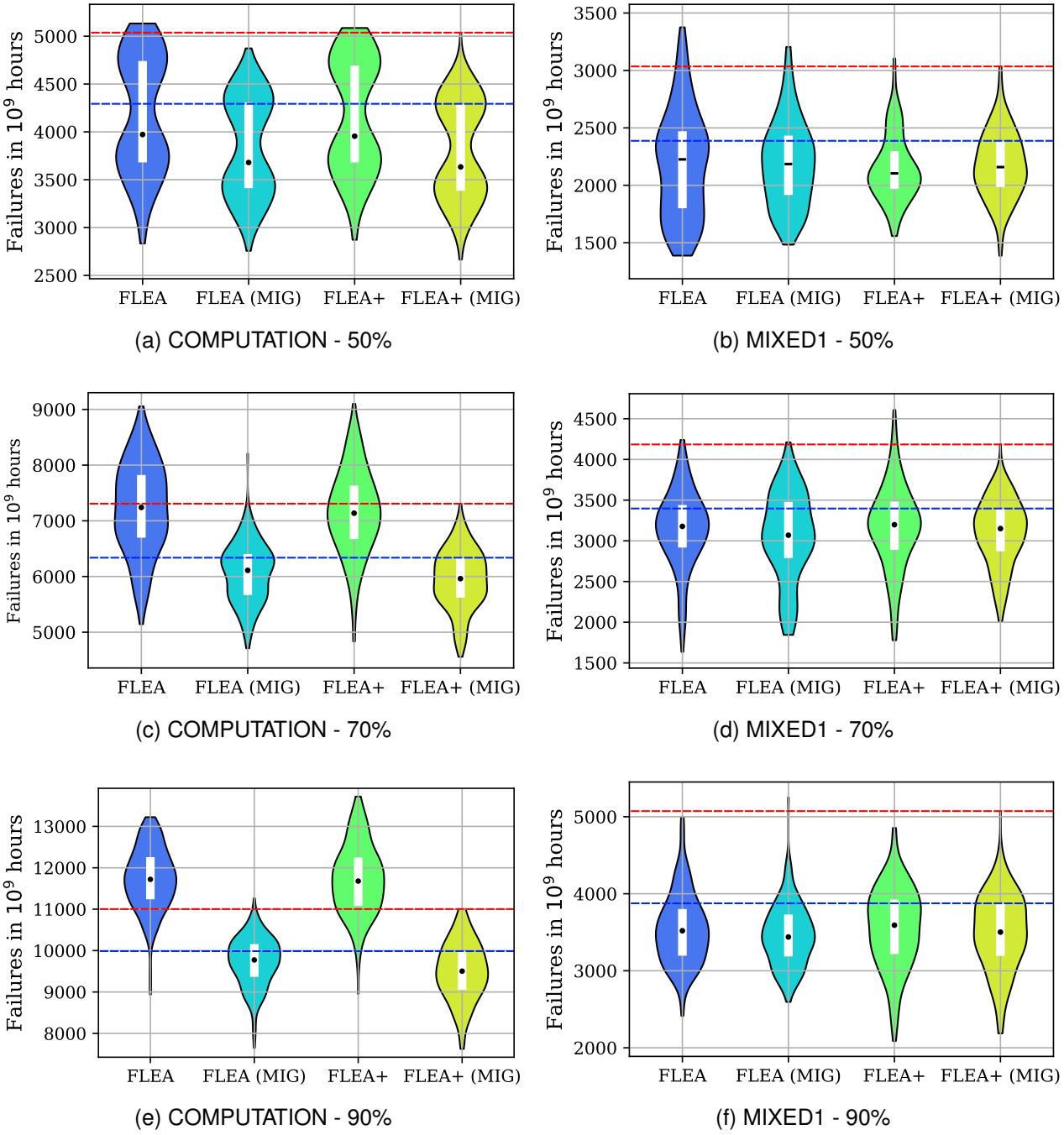


Figure B.2: FIT comparison between FLEA and FLEA+ in the 14x14 manycore. The blue dotted line is the FLEA+ with migration third quartile. The red dotted line is the FLEA+ with migration maximum FIT.

APPENDIX C – RESULTS

C.1 8x8 MIXED 1 50%

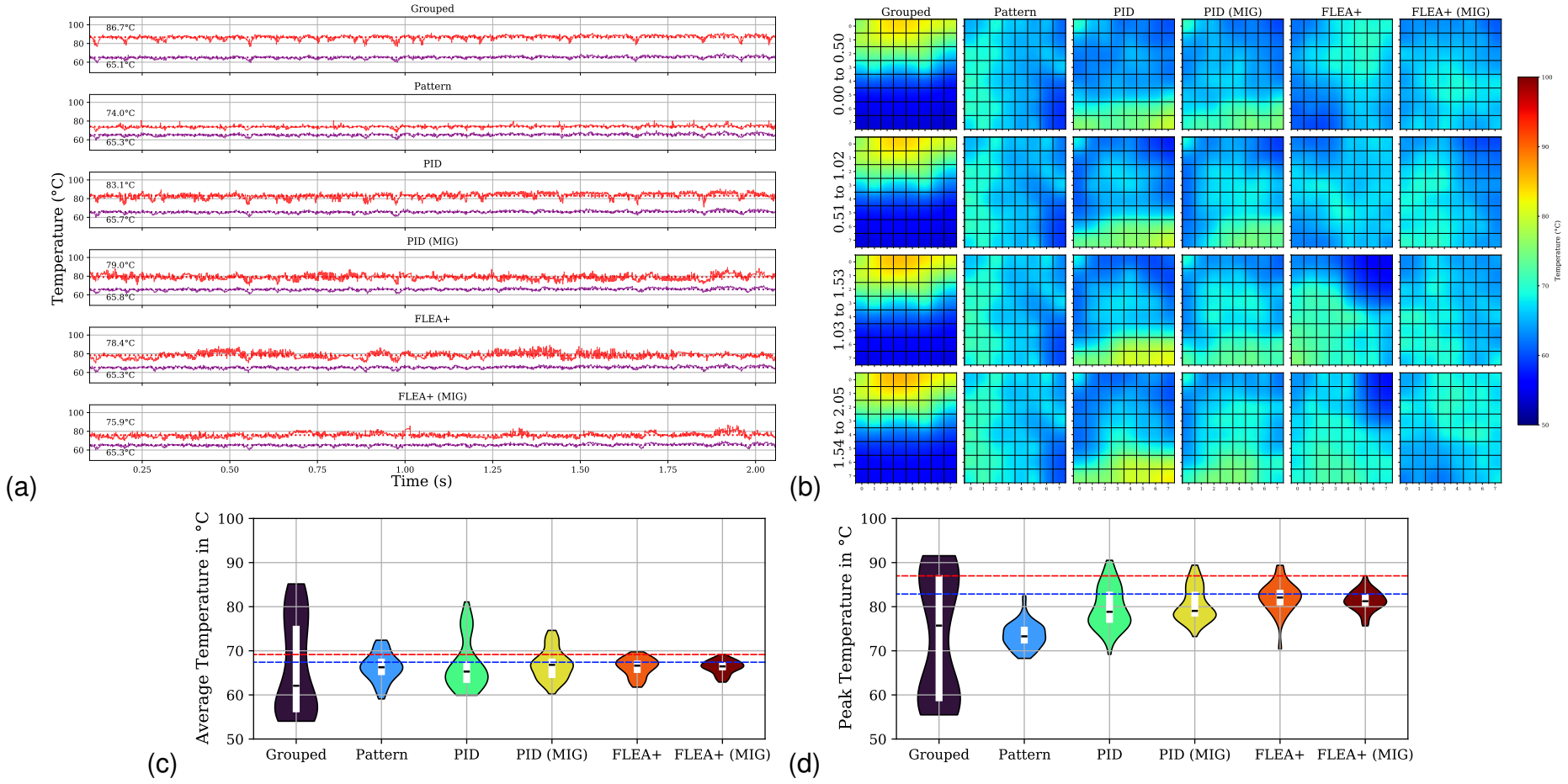


Figure C.1: 8x8 MIXED1 50% - (a) average and peak temperatures; (b) average temperature snapshot; (c) PE average temperature violin-box plot; (d) PE average peak temperature violin-box plot.

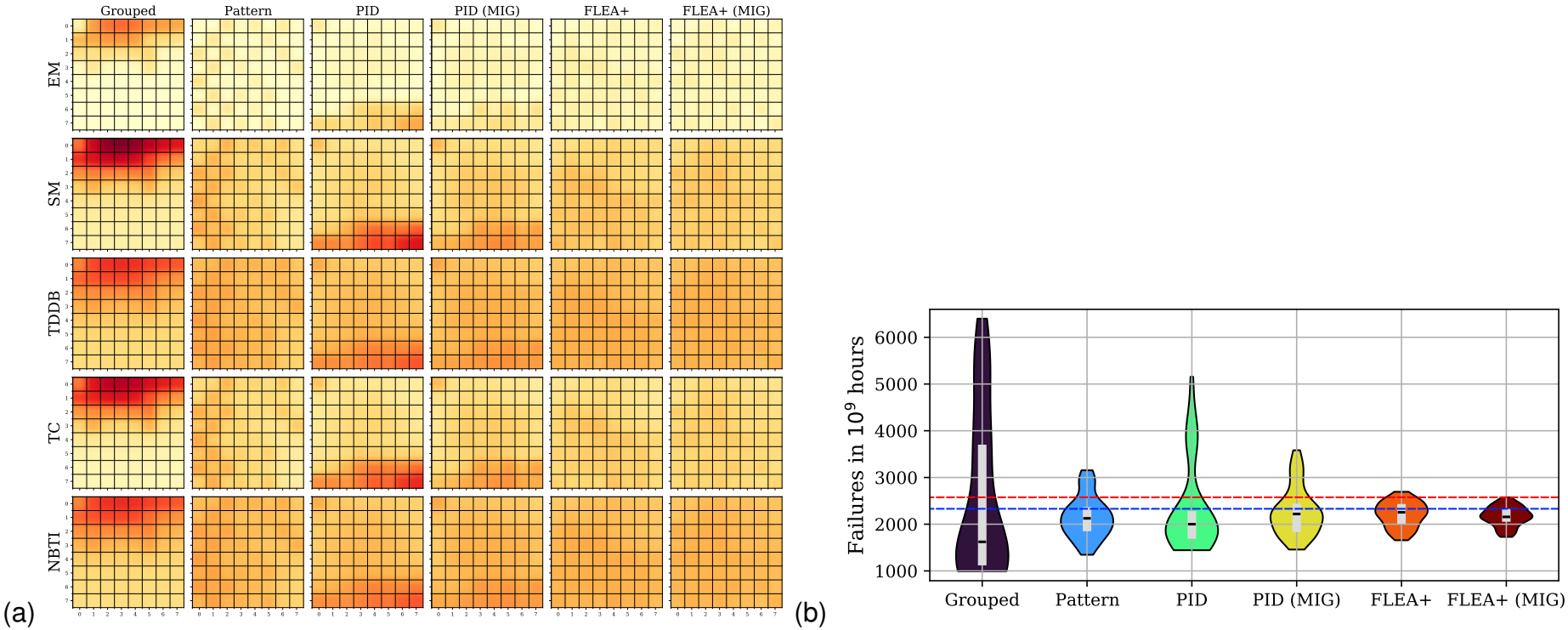


Figure C.2: 8x8 MIXED1 50% - (a) Manycore FIT intensity map per evaluated effect; (b) PE failure in time violin-box plot.

C.2 8x8 MIXED 1 70%

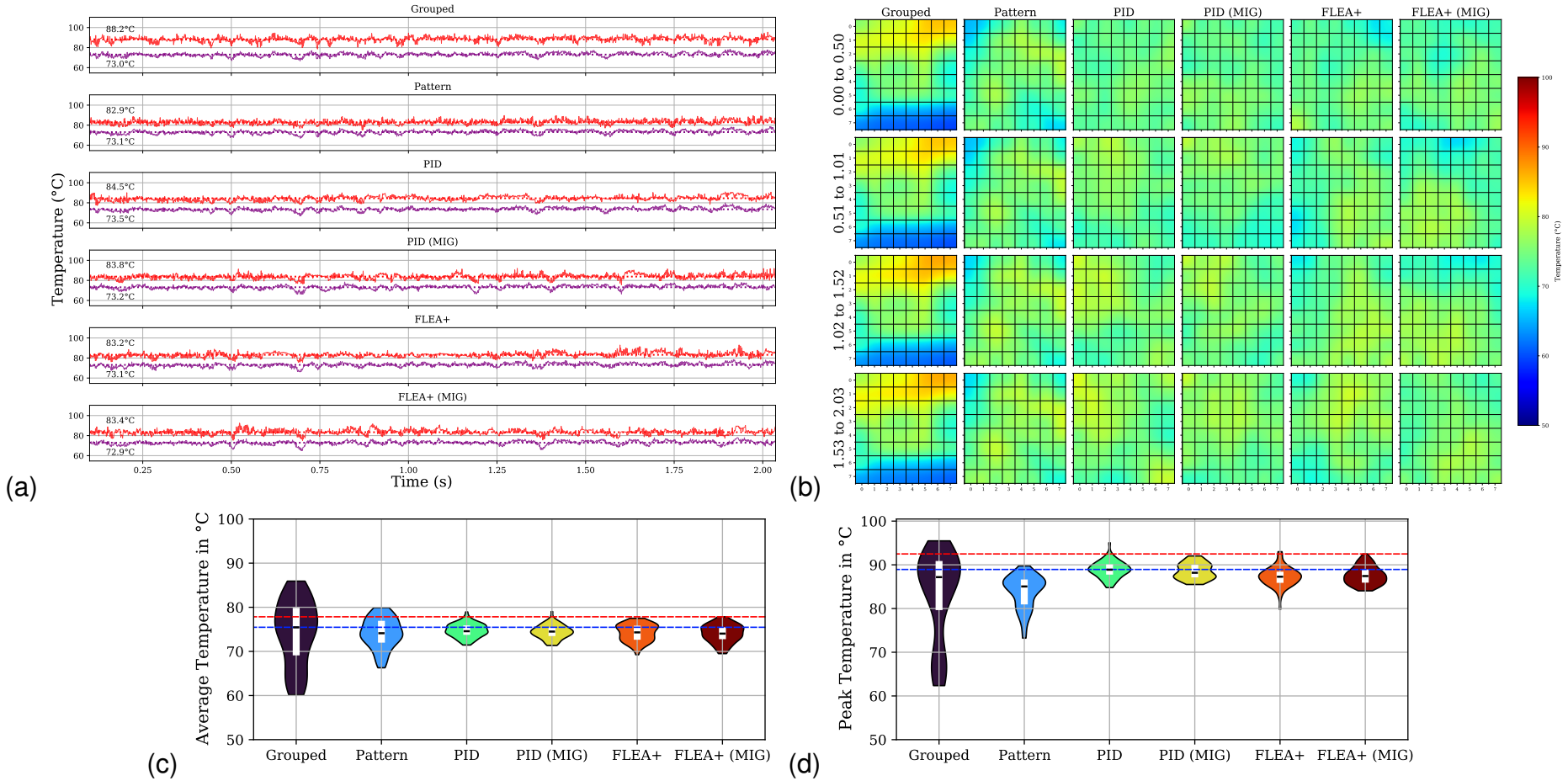


Figure C.3: 8x8 MIXED1 70% - (a) average and peak temperatures; (b) average temperature snapshot; (c) PE average temperature violin-box plot; (d) PE average peak temperature violin-box plot.

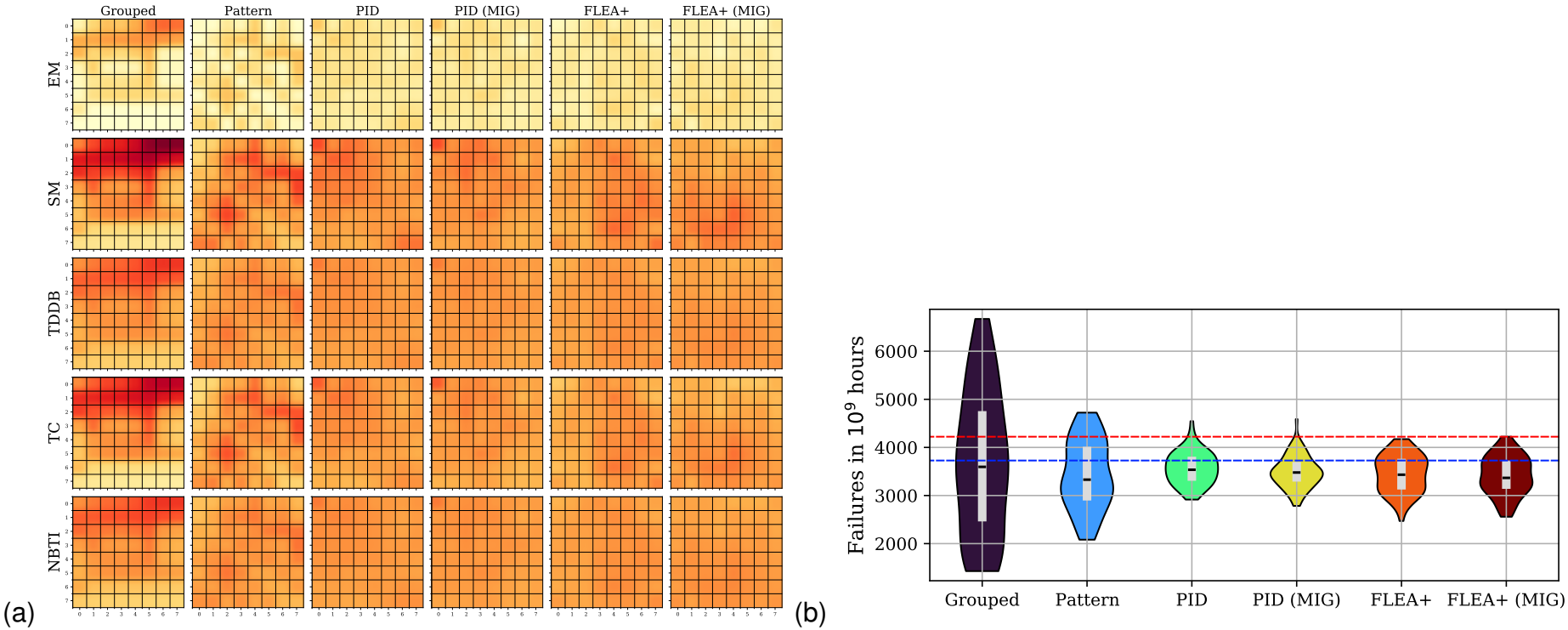


Figure C.4: 8x8 MIXED1 70% - (a) Manycore FIT intensity map per evaluated effect; (b) PE failure in time violin-box plot.

C.3 8x8 MIXED 1 90%

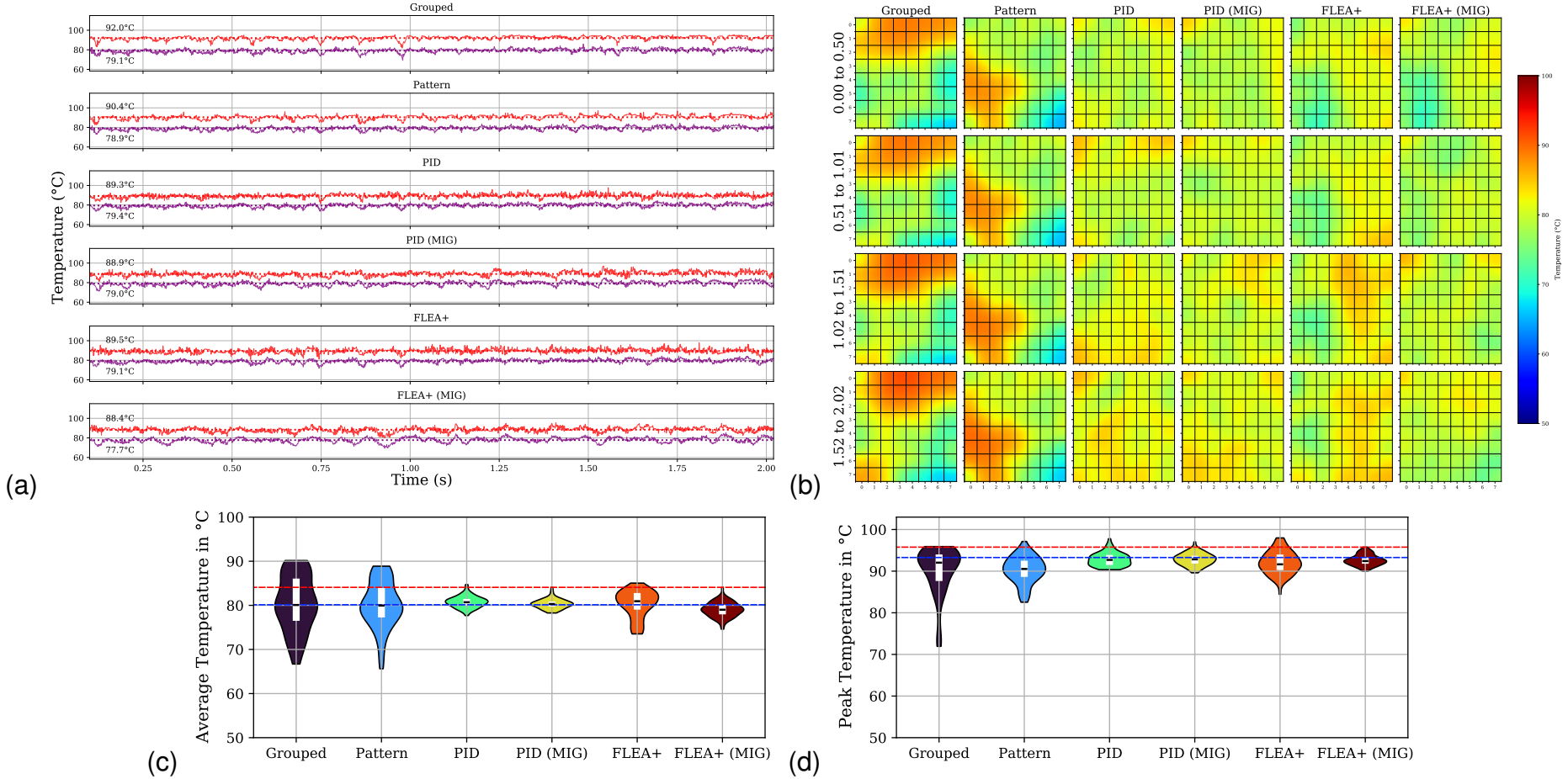


Figure C.5: 8x8 MIXED1 90% - (a) average and peak temperatures; (b) average temperature snapshot; (c) PE average temperature violin-box plot; (d) PE average peak temperature violin-box plot.

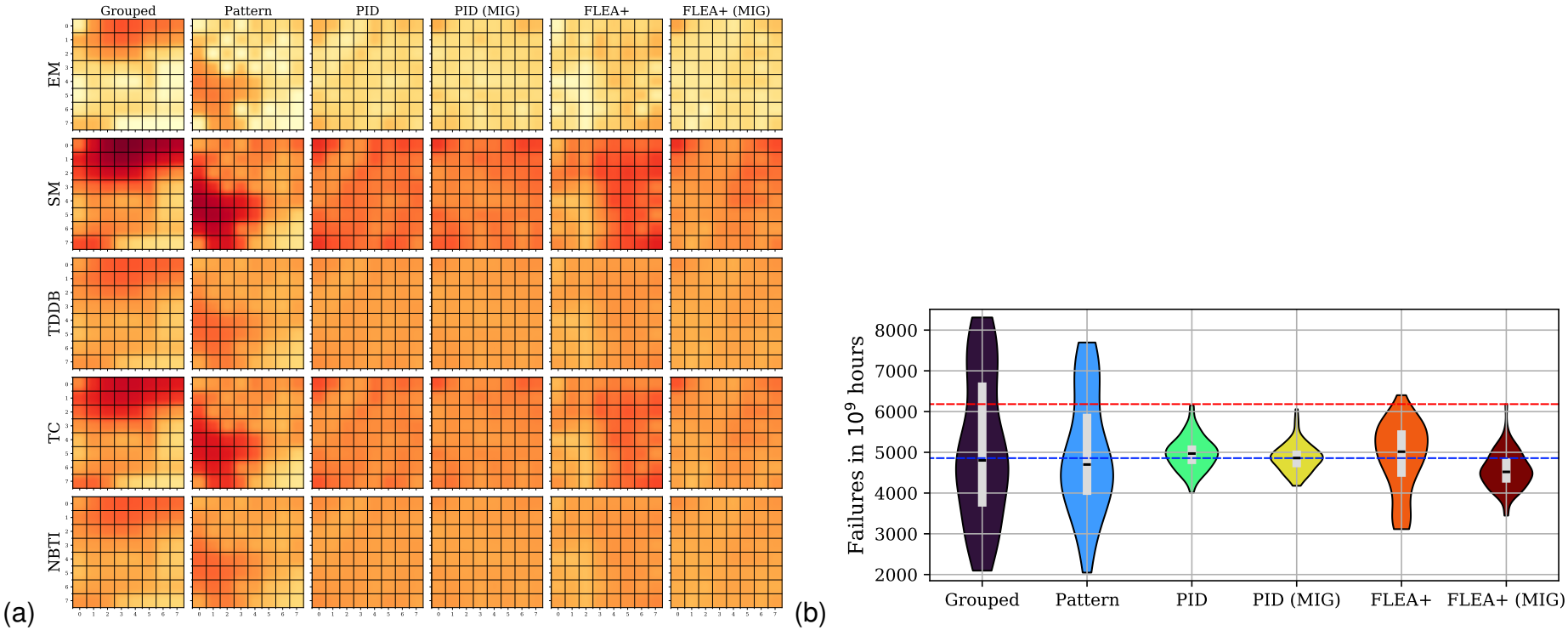


Figure C.6: 8x8 MIXED1 90% - (a) Manycore FIT intensity map per evaluated effect; (b) PE failure in time violin-box plot.

C.4 8x8 MIXED 2 50%

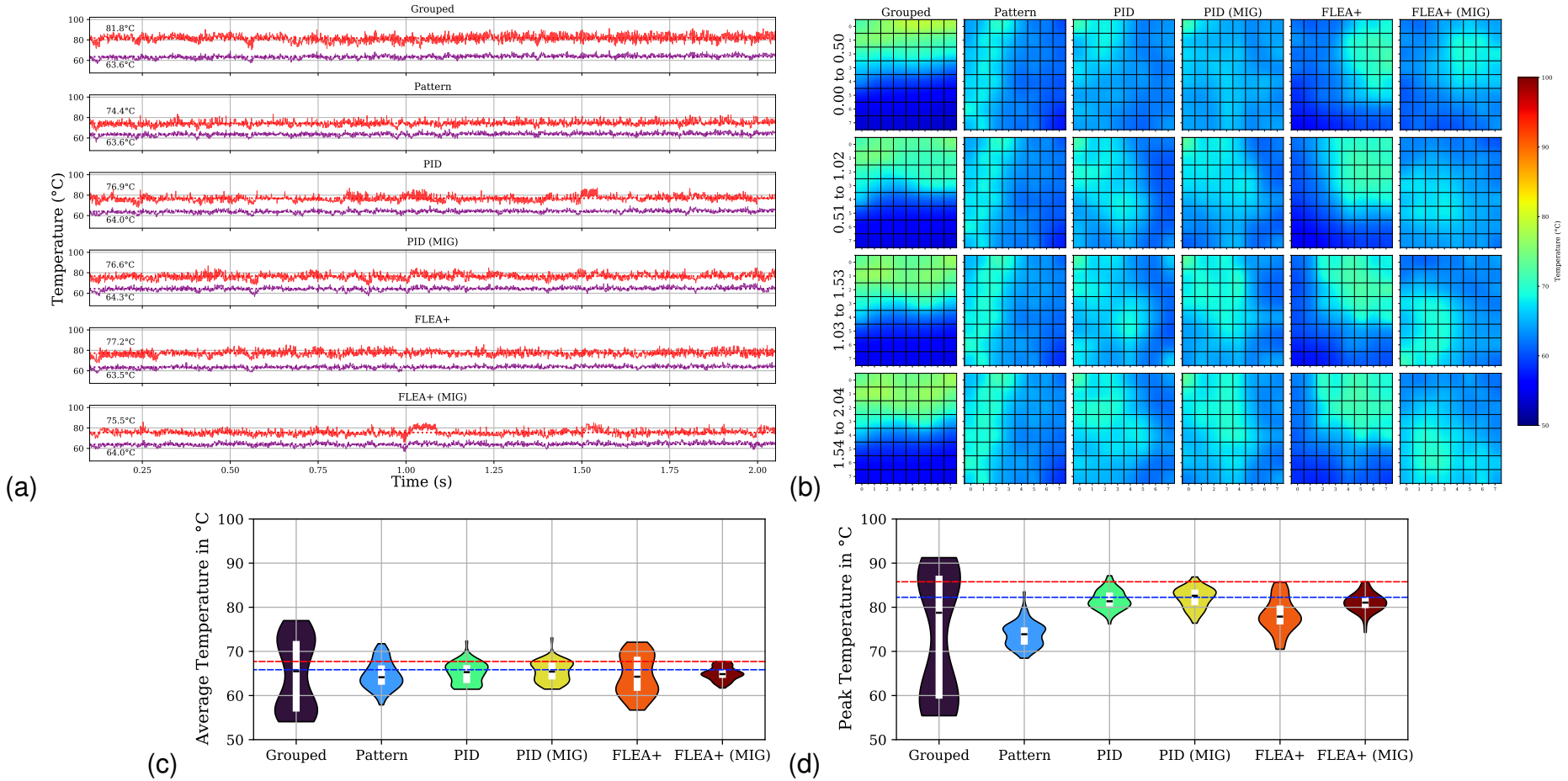


Figure C.7: 8x8 MIXED2 50% - (a) average and peak temperatures; (b) average temperature snapshot; (c) PE average temperature violin-box plot; (d) PE average peak temperature violin-box plot.

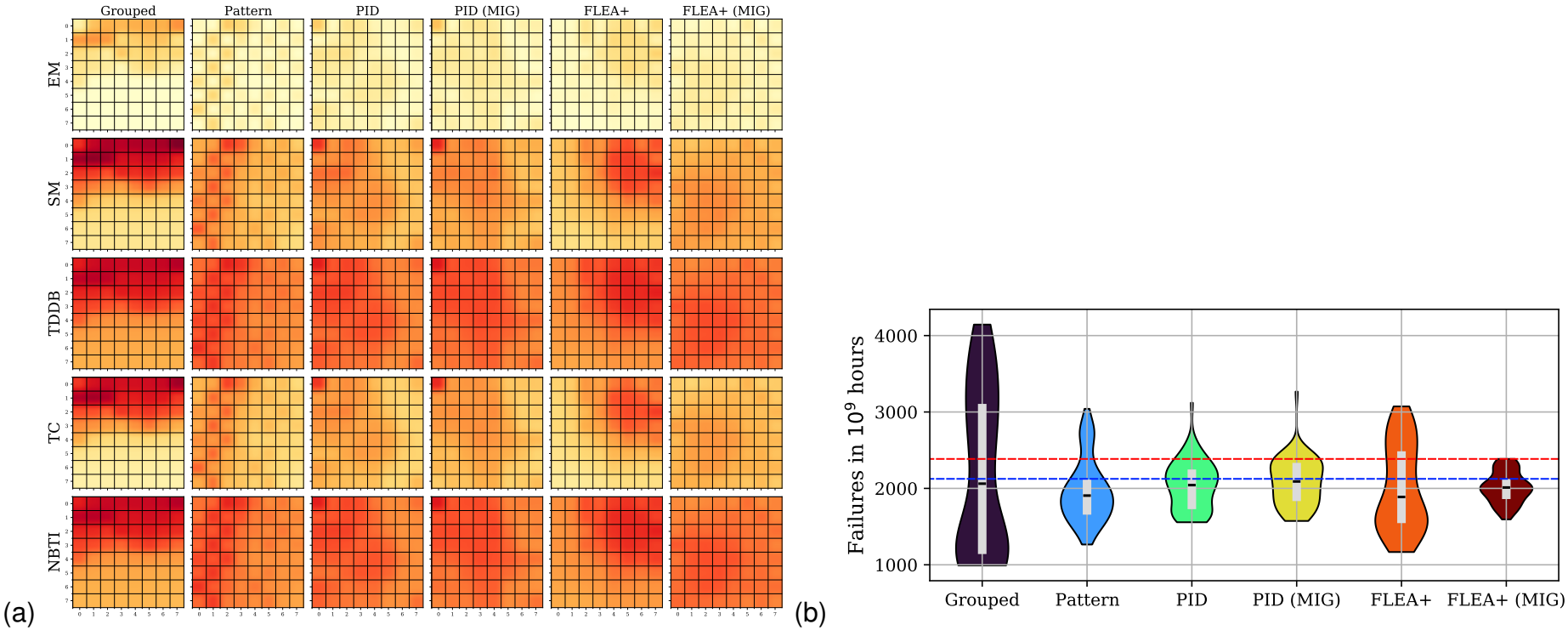


Figure C.8: 8x8 MIXED2 50% - (a) Manycore FIT intensity map per evaluated effect; (b) PE failure in time violin-box plot.

C.5 8x8 MIXED 2 70%

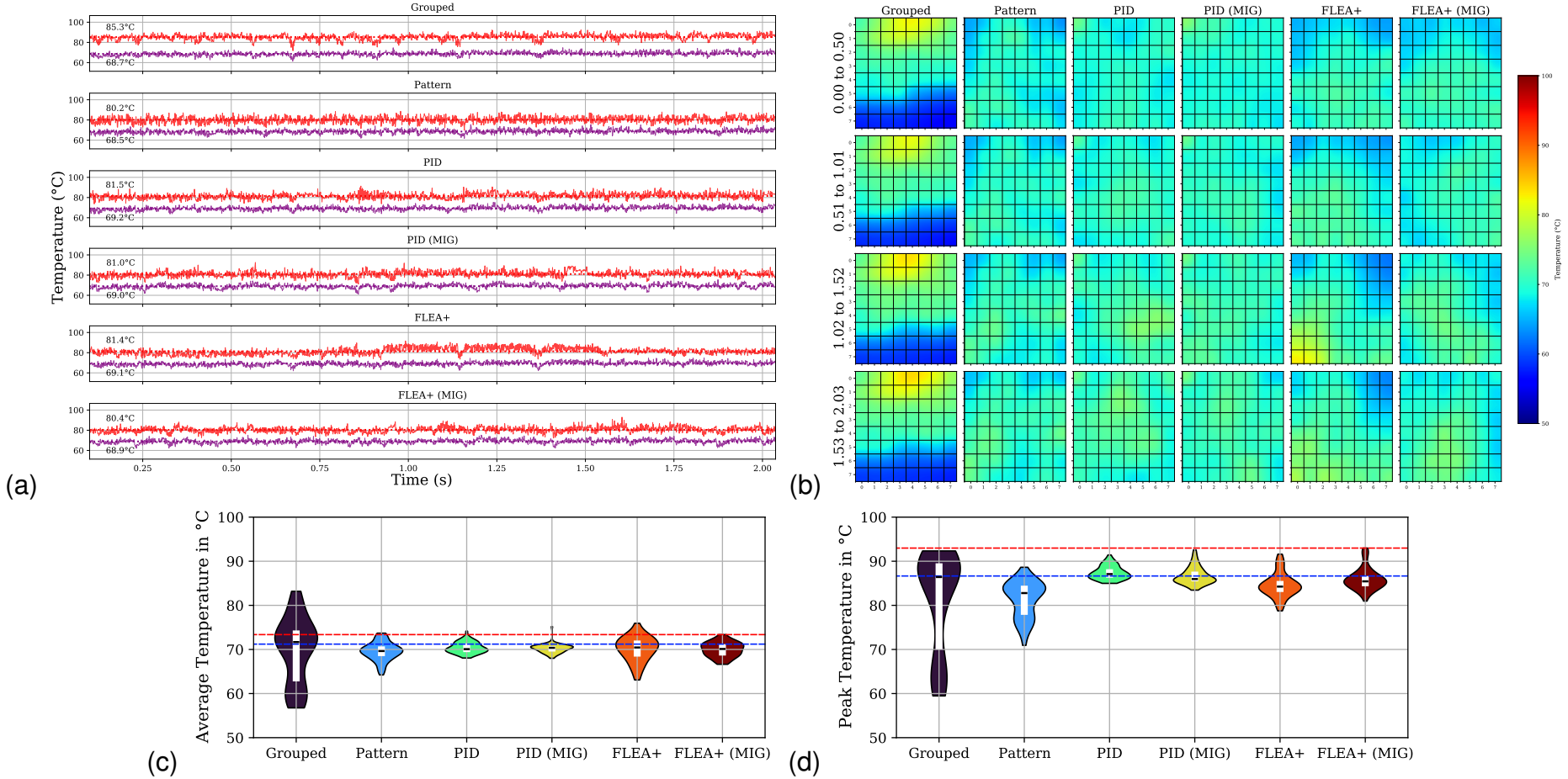


Figure C.9: 8x8 MIXED2 70% - (a) average and peak temperatures; (b) average temperature snapshot; (c) PE average temperature violin-box plot; (d) PE average peak temperature violin-box plot.

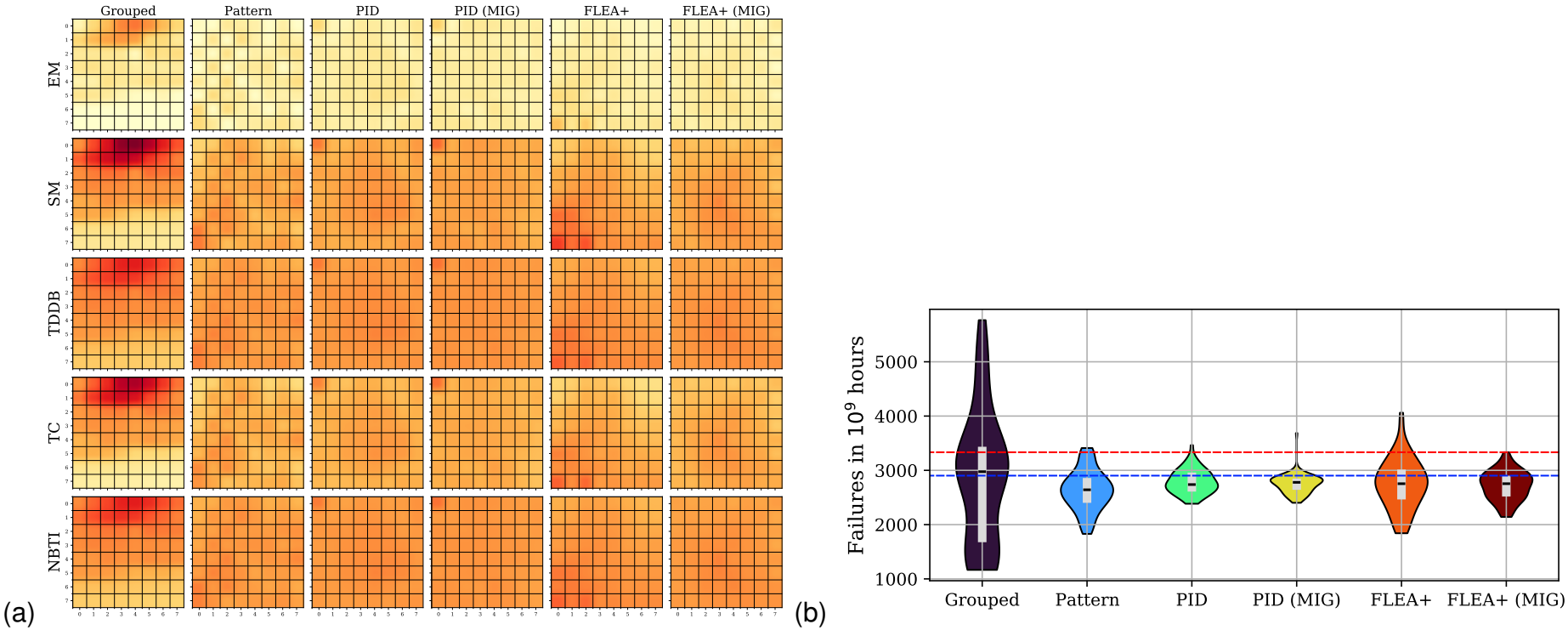


Figure C.10: 8x8 MIXED2 70% - (a) Manycore FIT intensity map per evaluated effect; (b) PE failure in time violin-box plot.

C.6 8x8 MIXED 2 90%

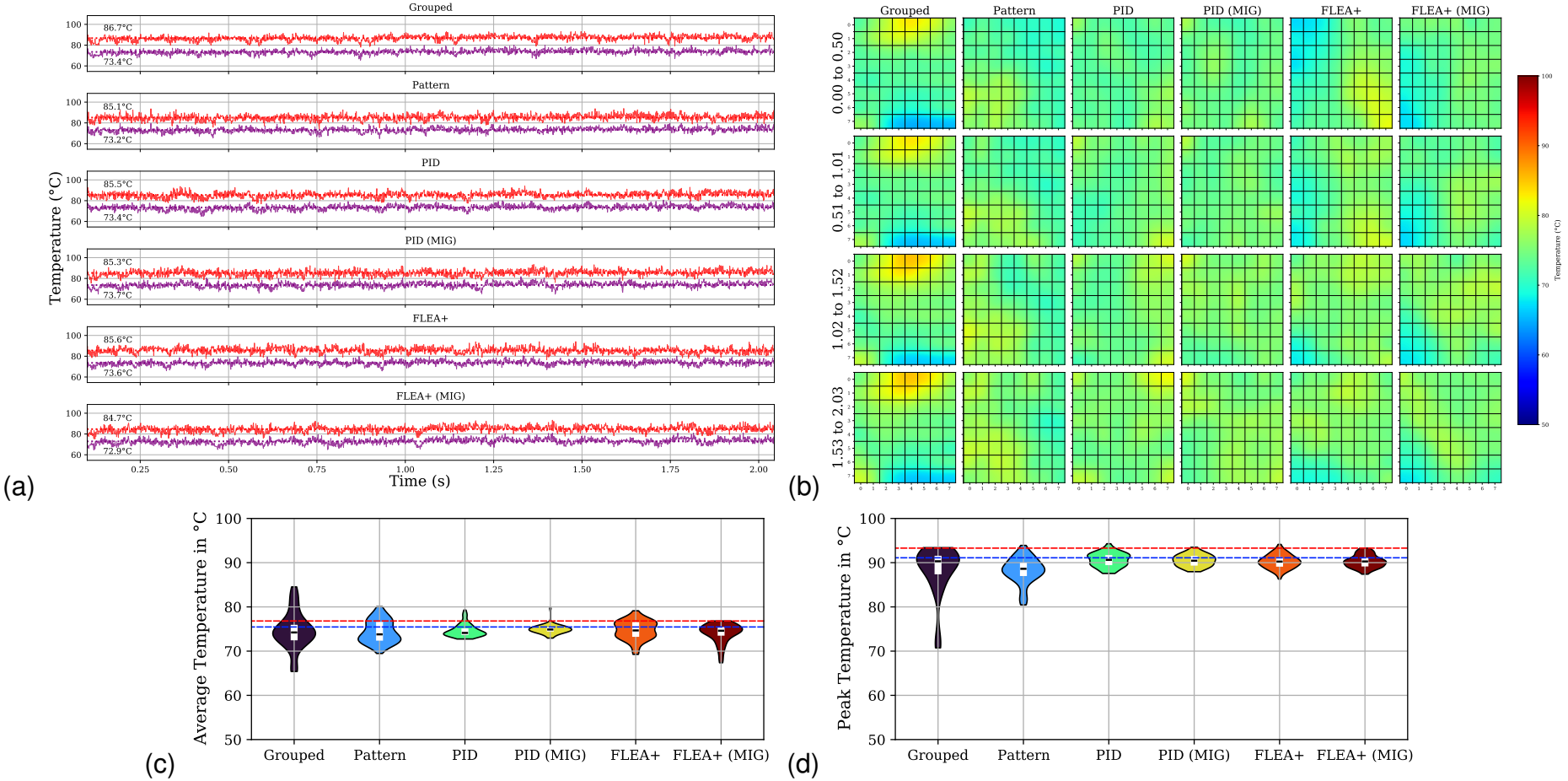


Figure C.11: 8x8 MIXED2 90% - (a) average and peak temperatures; (b) average temperature snapshot; (c) PE average temperature violin-box plot; (d) PE average peak temperature violin-box plot.

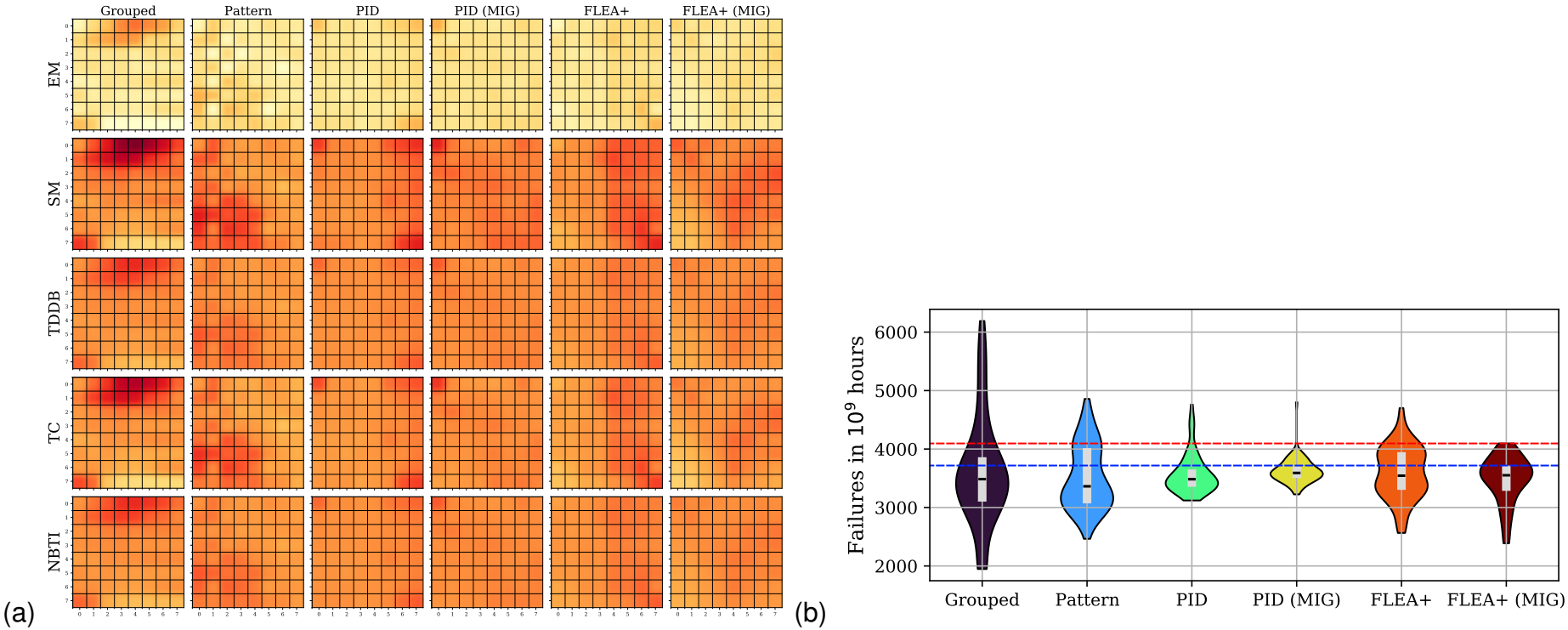


Figure C.12: 8x8 MIXED2 90% - (a) Manycore FIT intensity map per evaluated effect; (b) PE failure in time violin-box plot.

C.7 8x8 COMPUTATION 50%

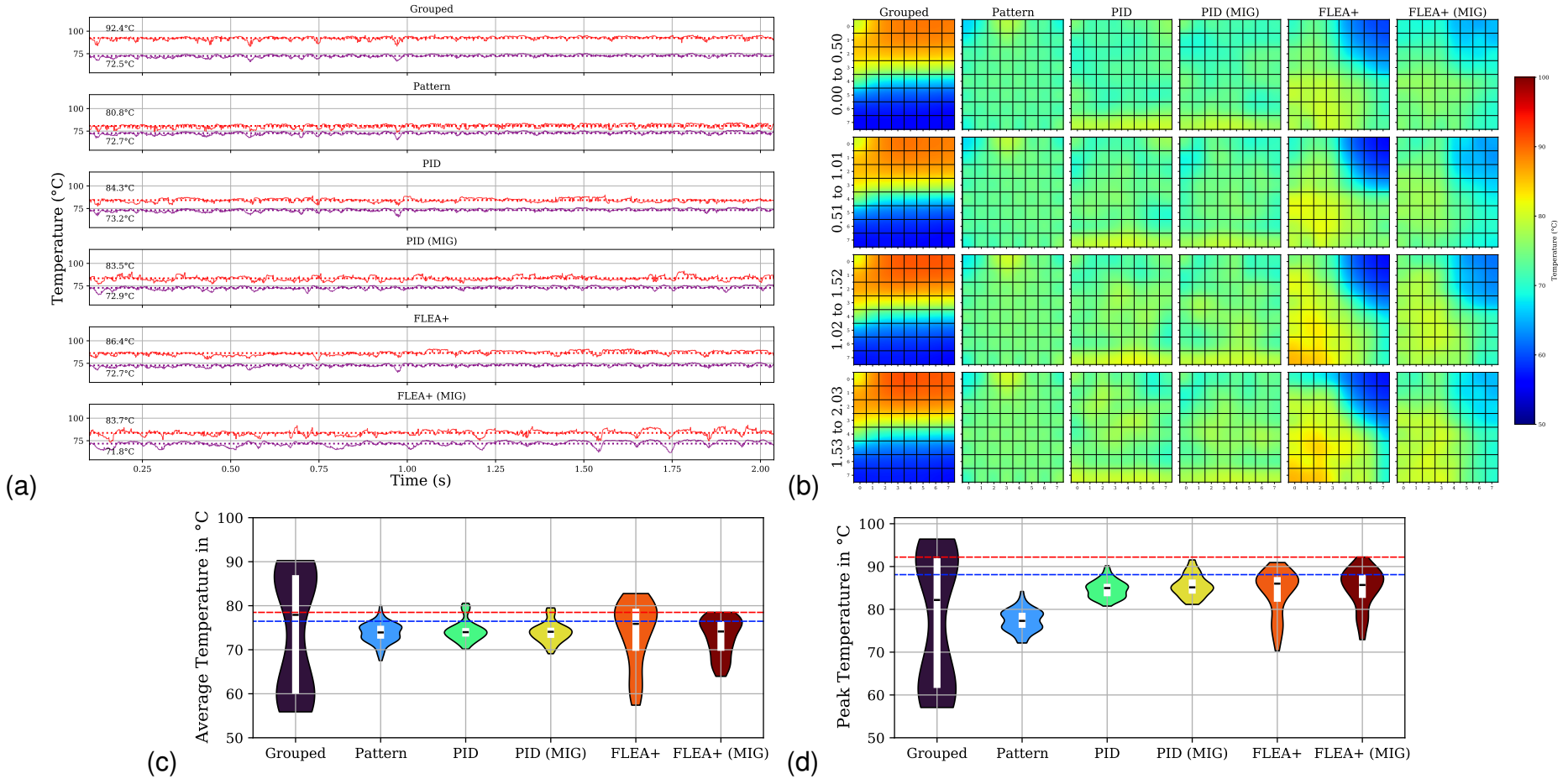


Figure C.13: 8x8 COMPUTATION 50% - (a) average and peak temperatures; (b) average temperature snapshot; (c) PE average temperature violin-box plot; (d) PE average peak temperature violin-box plot.

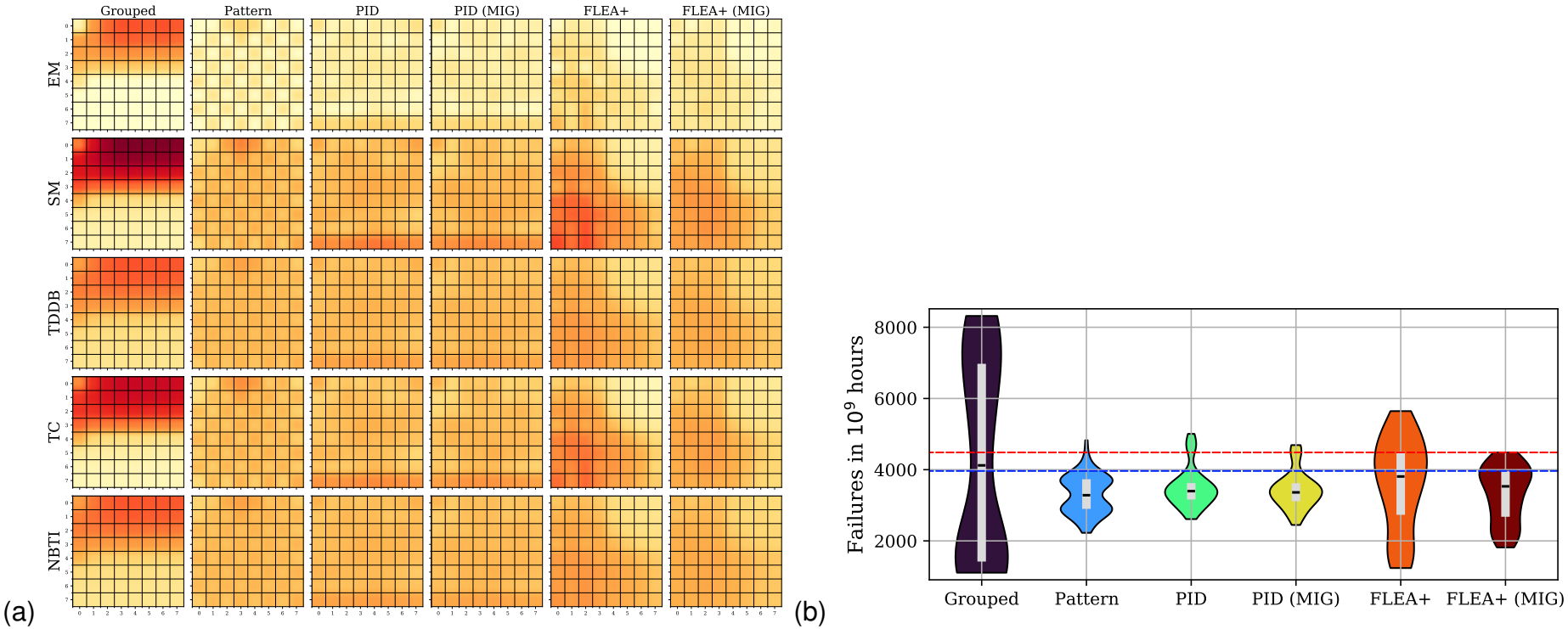


Figure C.14: 8x8 COMPUTATION 50% - (a) Manycore FIT intensity map per evaluated effect; (b) PE failure in time violin-box plot.

C.8 8x8 COMPUTATION 70%

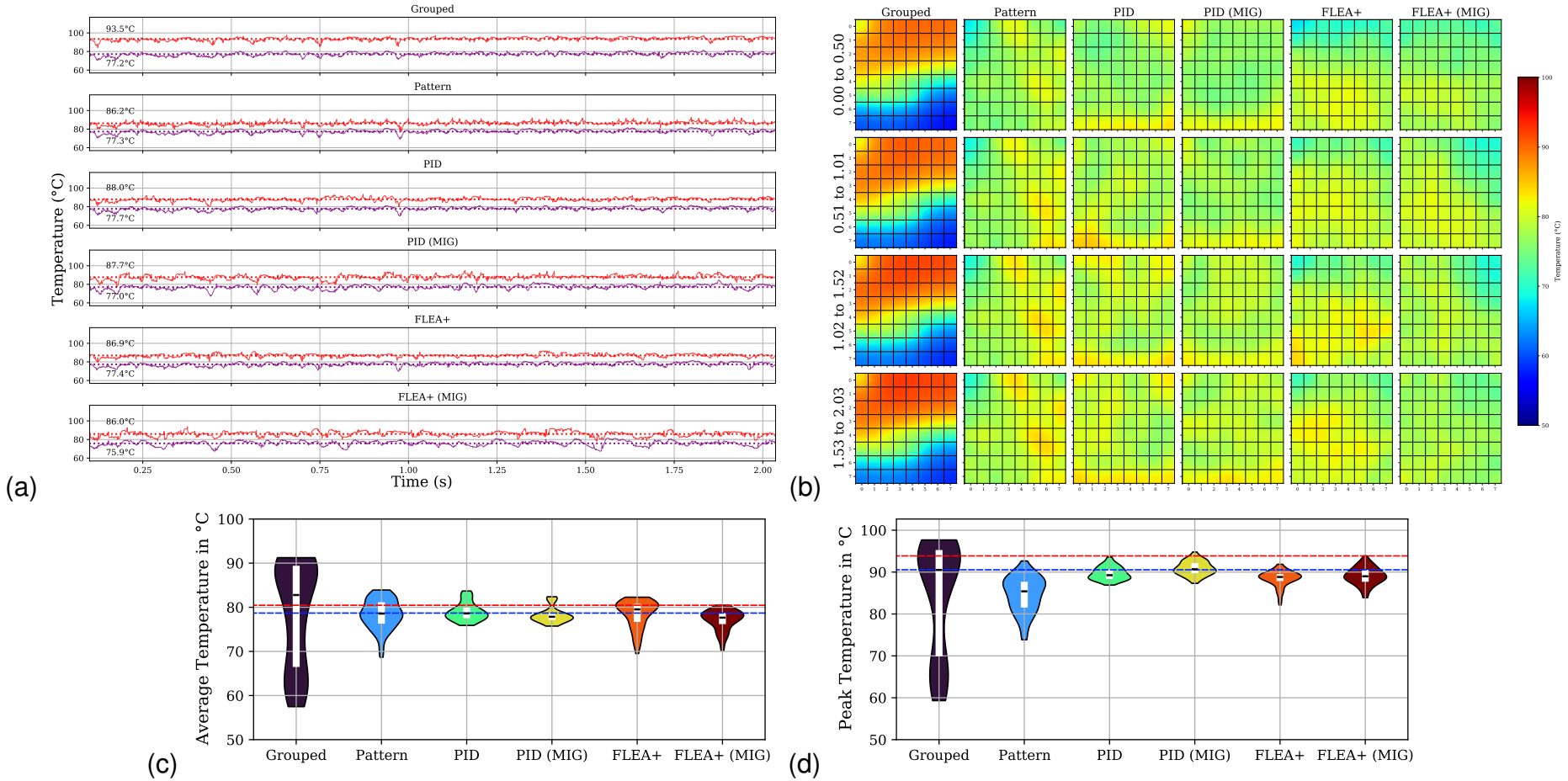


Figure C.15: 8x8 COMPUTATION 70% - (a) average and peak temperatures; (b) average temperature snapshot; (c) PE average temperature violin-box plot; (d) PE average peak temperature violin-box plot.

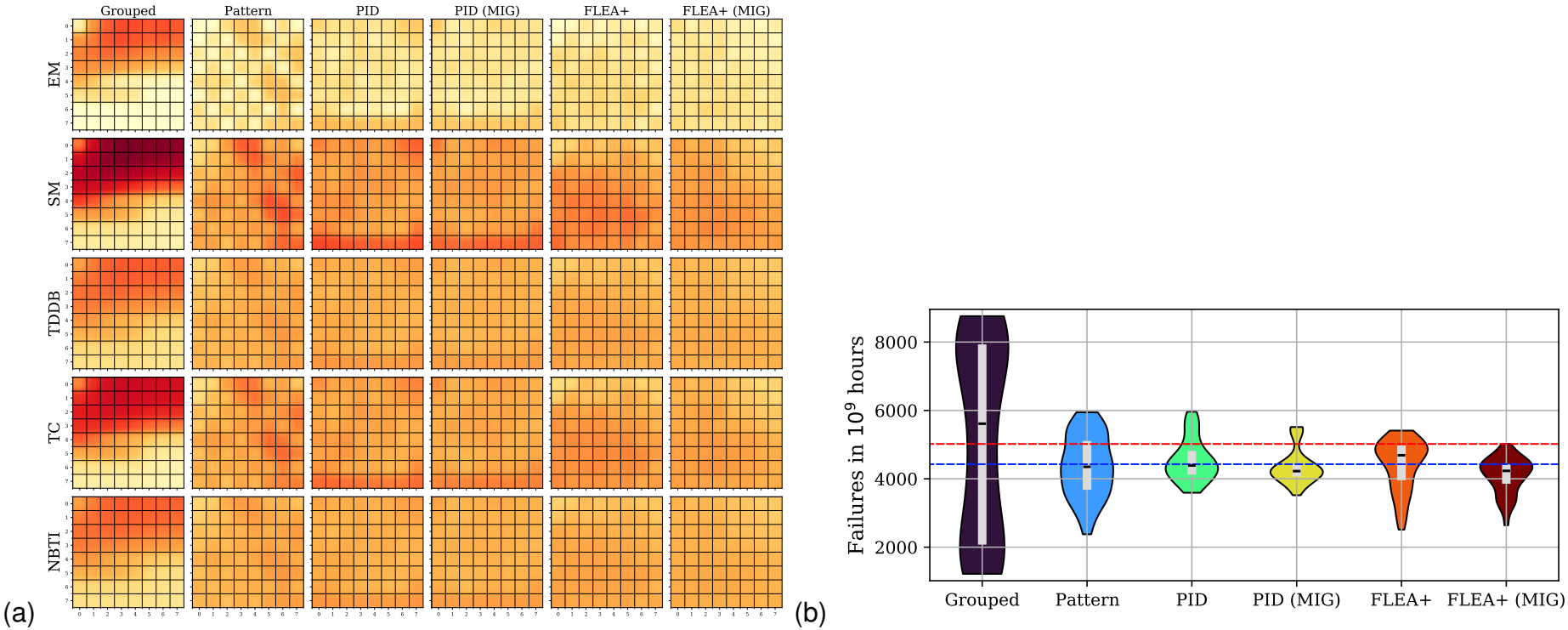


Figure C.16: 8x8 COMPUTATION 70% - (a) Manycore FIT intensity map per evaluated effect; (b) PE failure in time violin-box plot.

C.9 8x8 COMPUTATION 90%

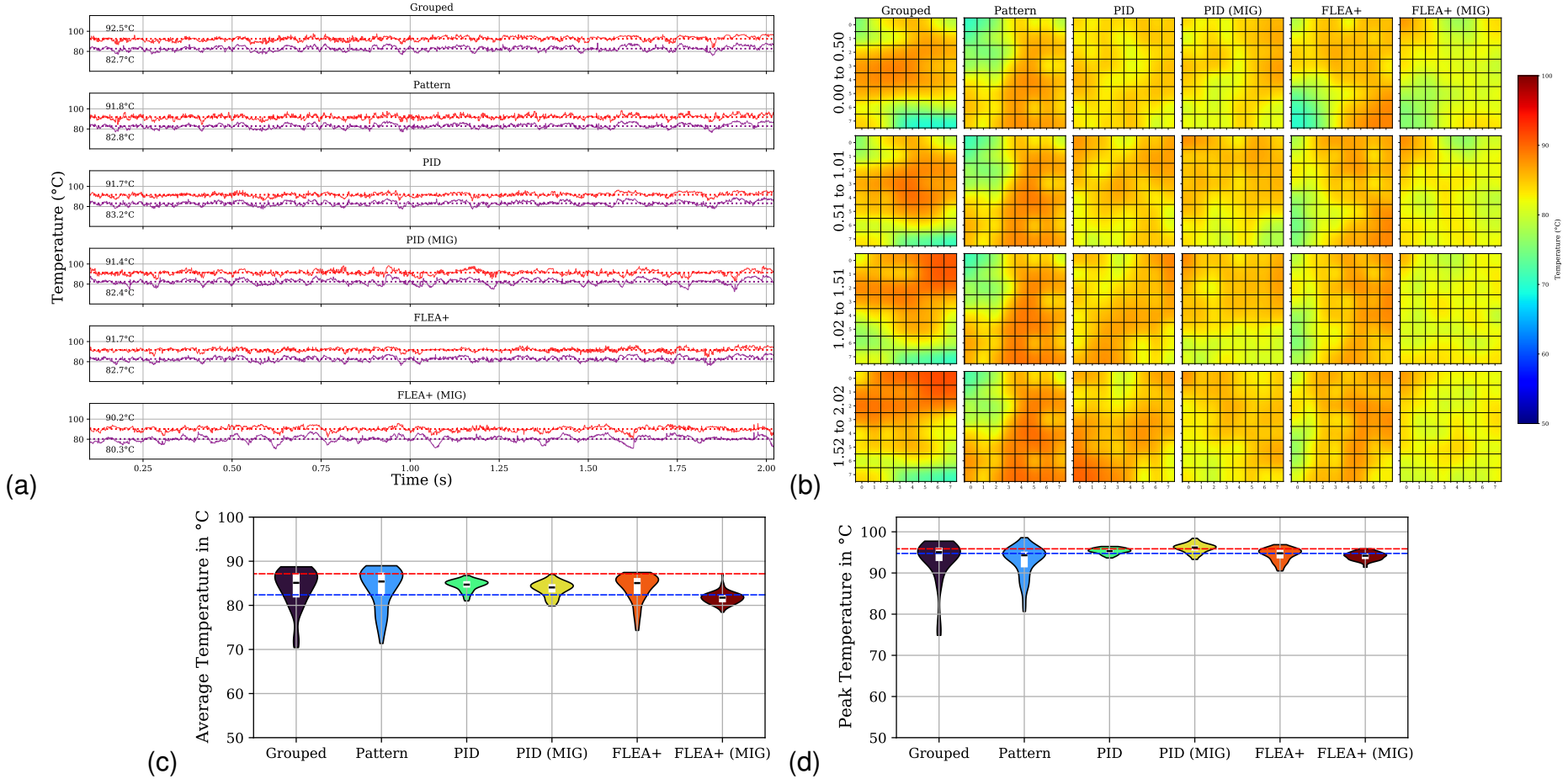


Figure C.17: 8x8 COMPUTATION 90% - (a) average and peak temperatures; (b) average temperature snapshot; (c) PE average temperature violin-box plot; (d) PE average peak temperature violin-box plot.

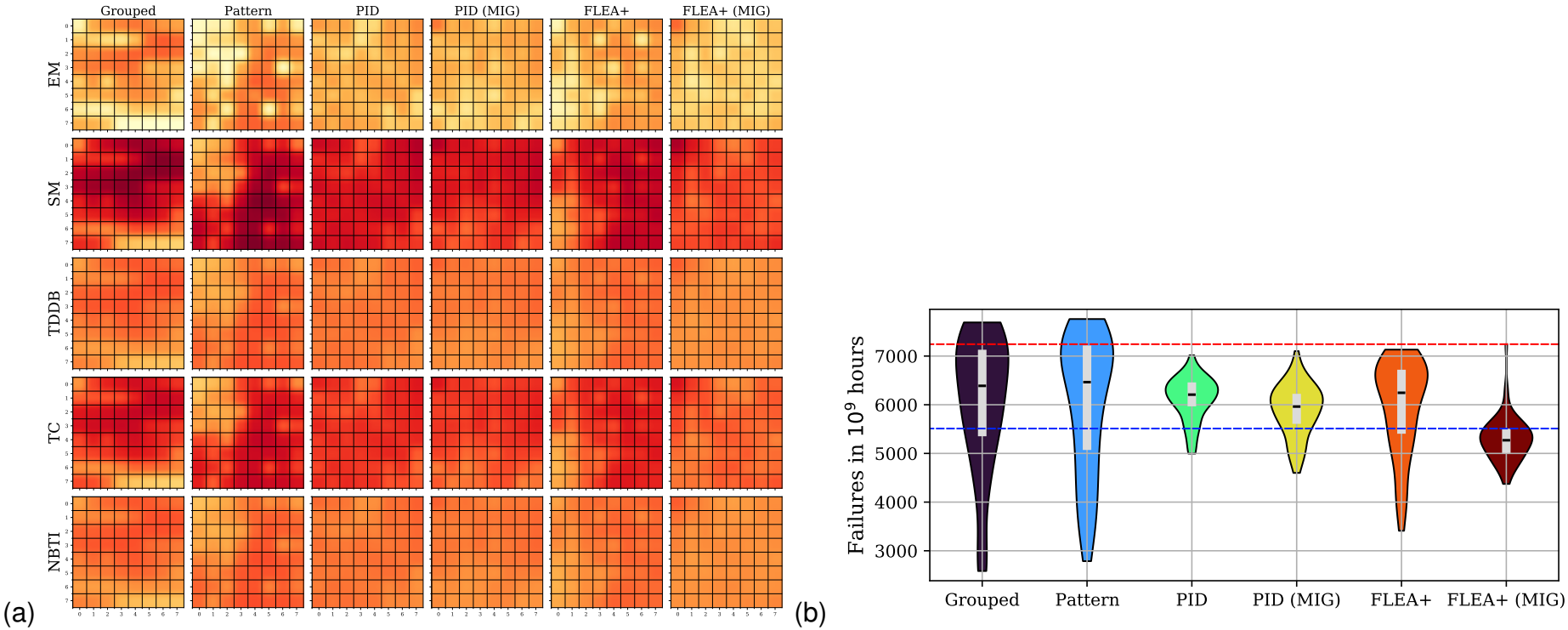


Figure C.18: 8x8 COMPUTATION 90% - (a) Manycore FIT intensity map per evaluated effect; (b) PE failure in time violin-box plot.

C.10 14x14 MIXED 50%

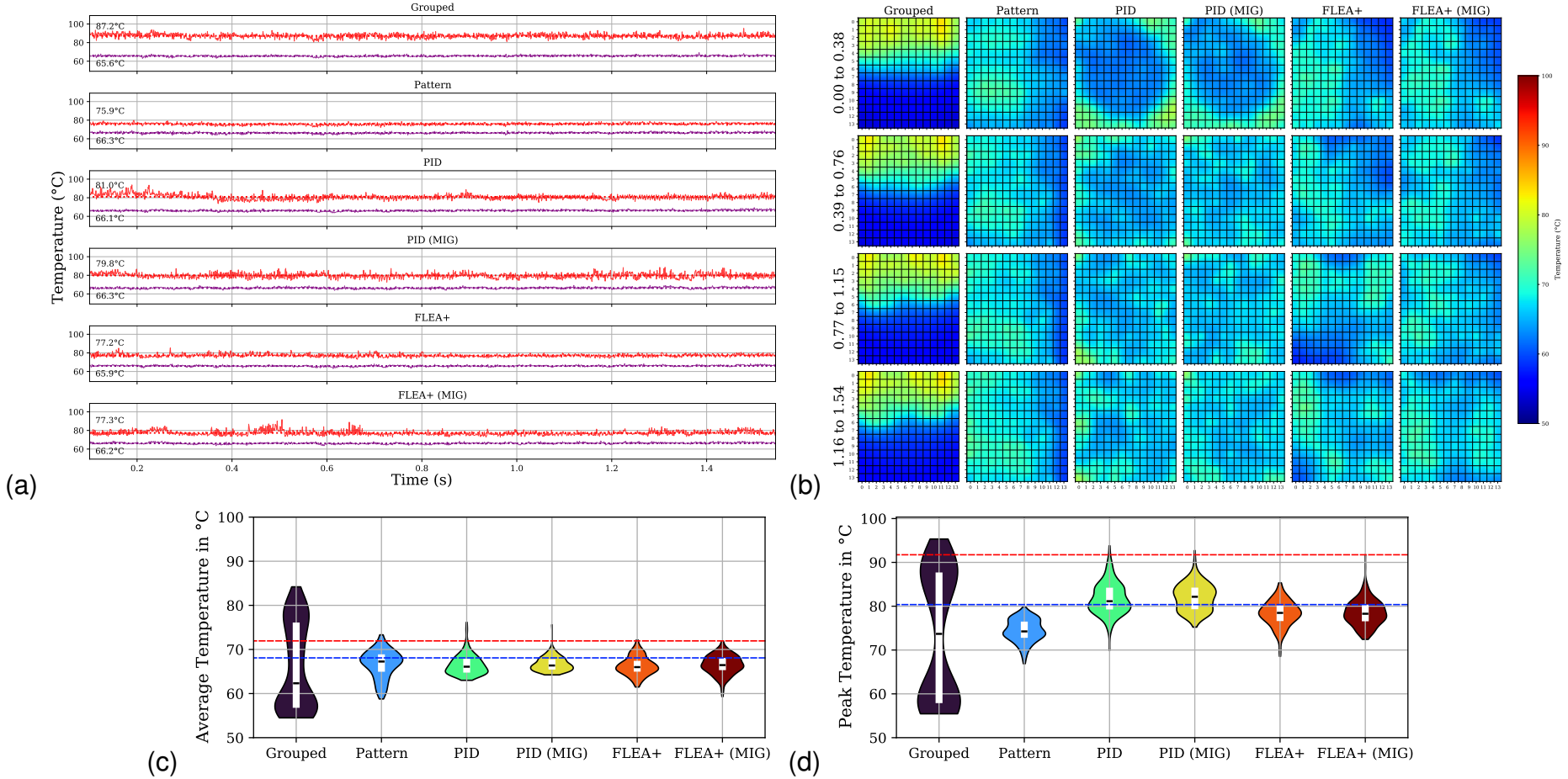


Figure C.19: 14x14 MIXED 50% - (a) average and peak temperatures; (b) average temperature snapshot; (c) PE average temperature violin-box plot; (d) PE average peak temperature violin-box plot.

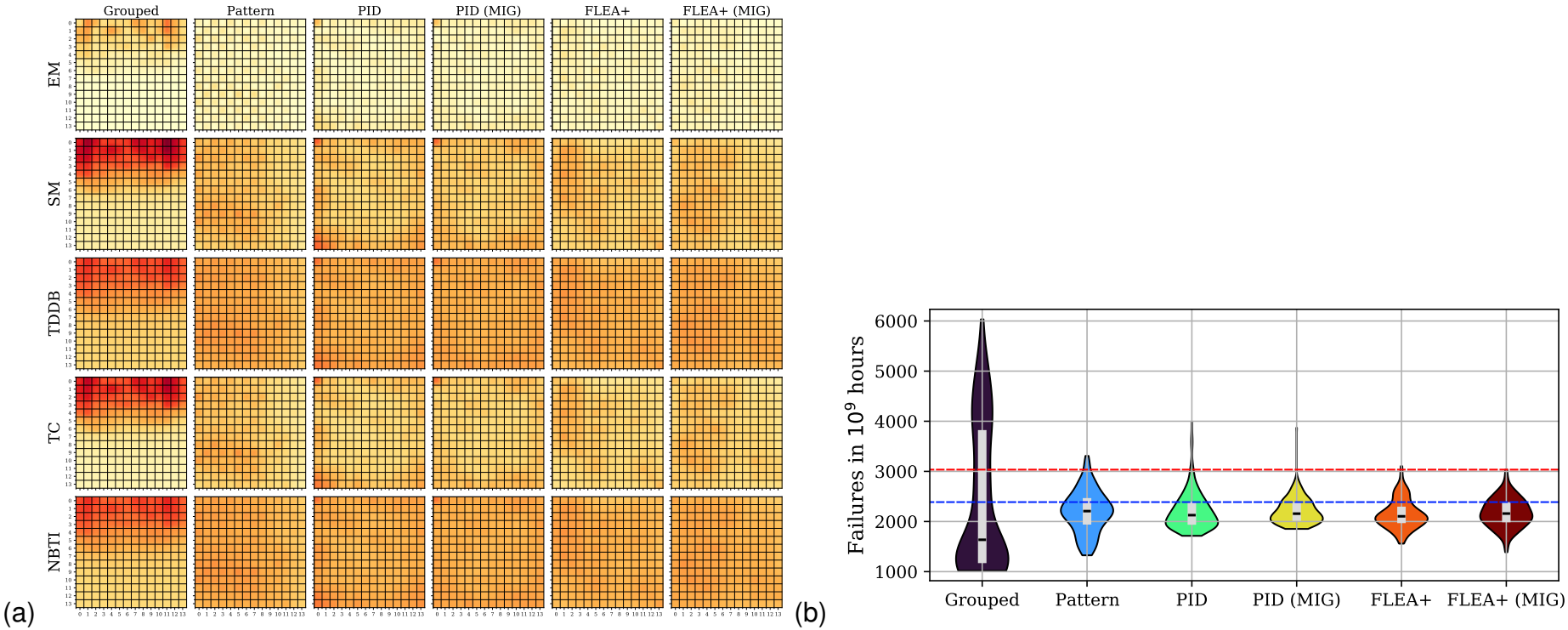


Figure C.20: 14x14 MIXED 50% - (a) Manycore FIT intensity map per evaluated effect; (b) PE failure in time violin-box plot.

C.11 14x14 MIXED 70%

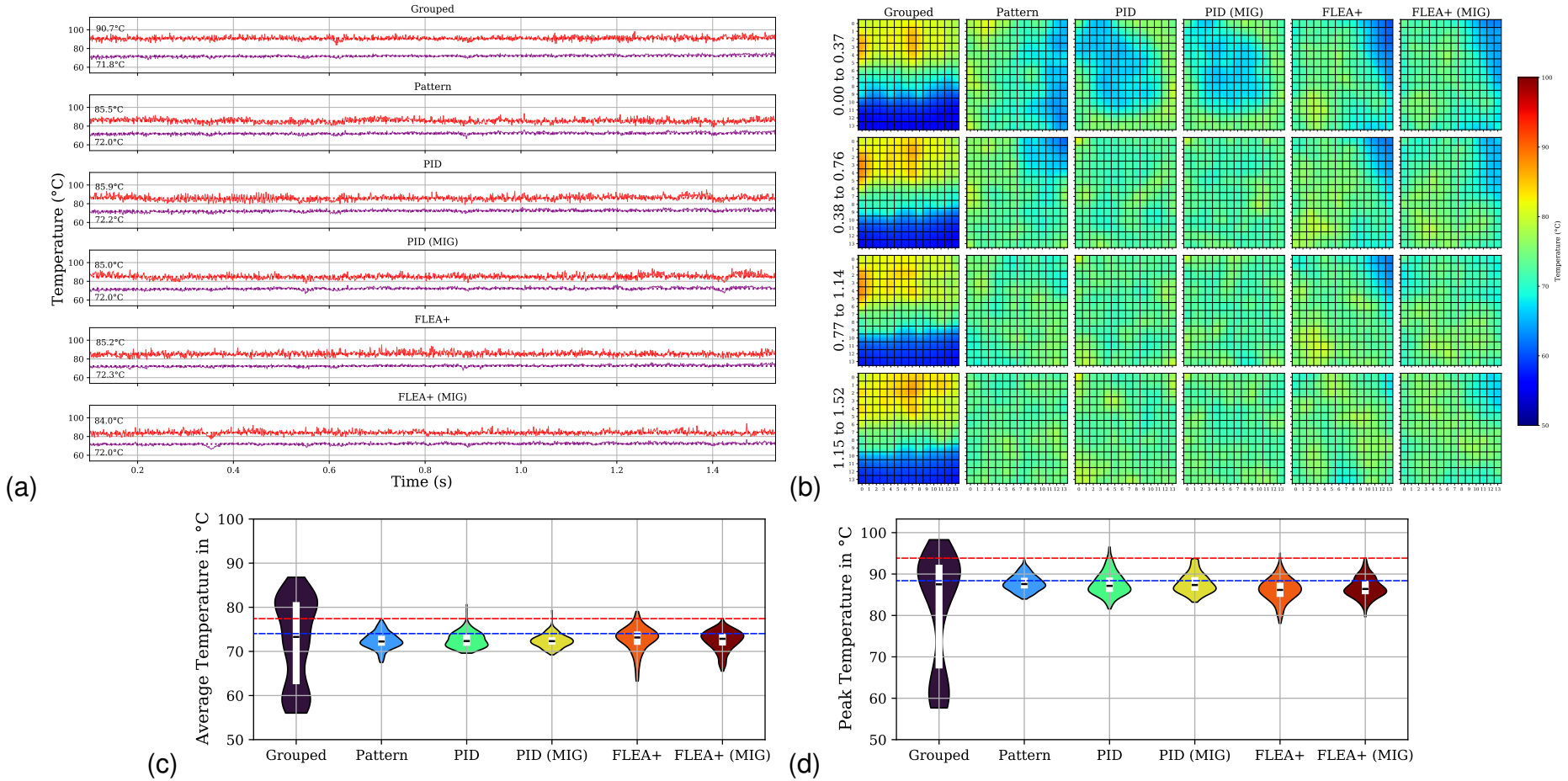


Figure C.21: 14x14 MIXED 70% - (a) average and peak temperatures; (b) average temperature snapshot; (c) PE average temperature violin-box plot; (d) PE average peak temperature violin-box plot.

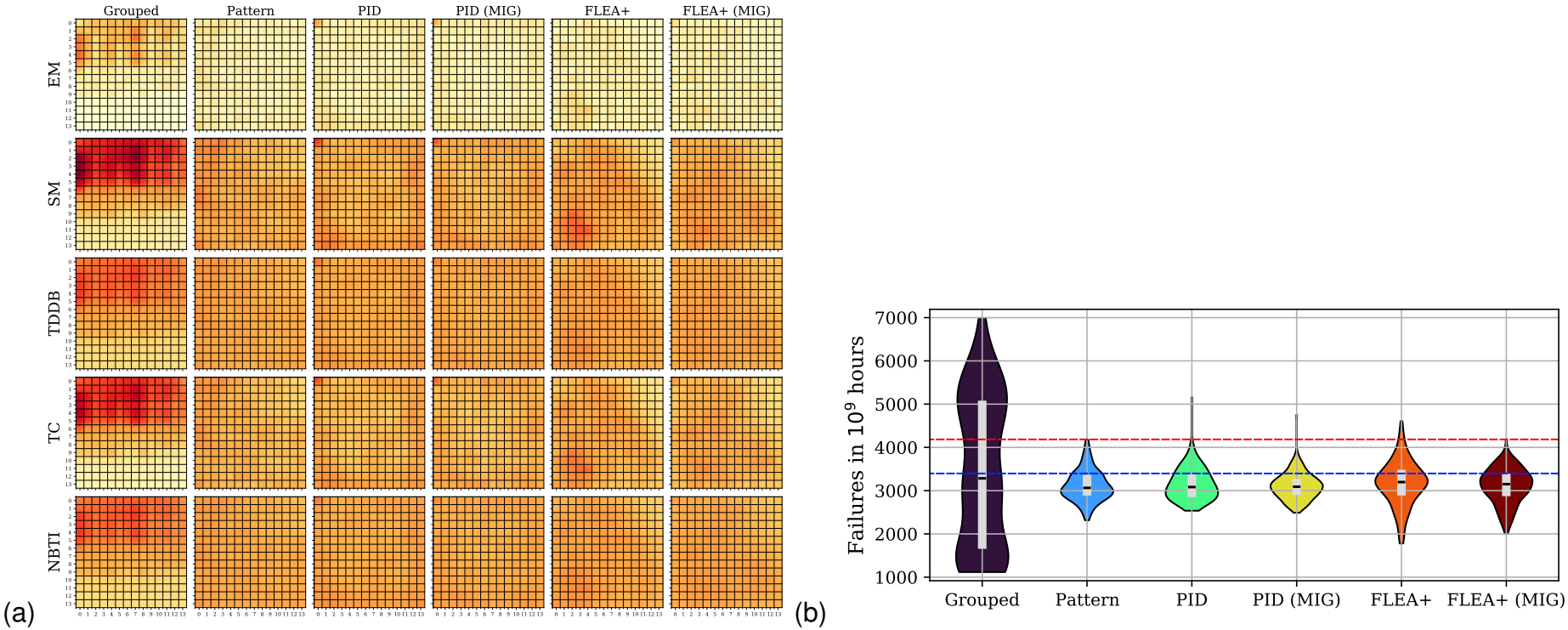


Figure C.22: 14x14 MIXED 70% - (a) Manycore FIT intensity map per evaluated effect; (b) PE failure in time violin-box plot.

C.12 14x14 MIXED 90%

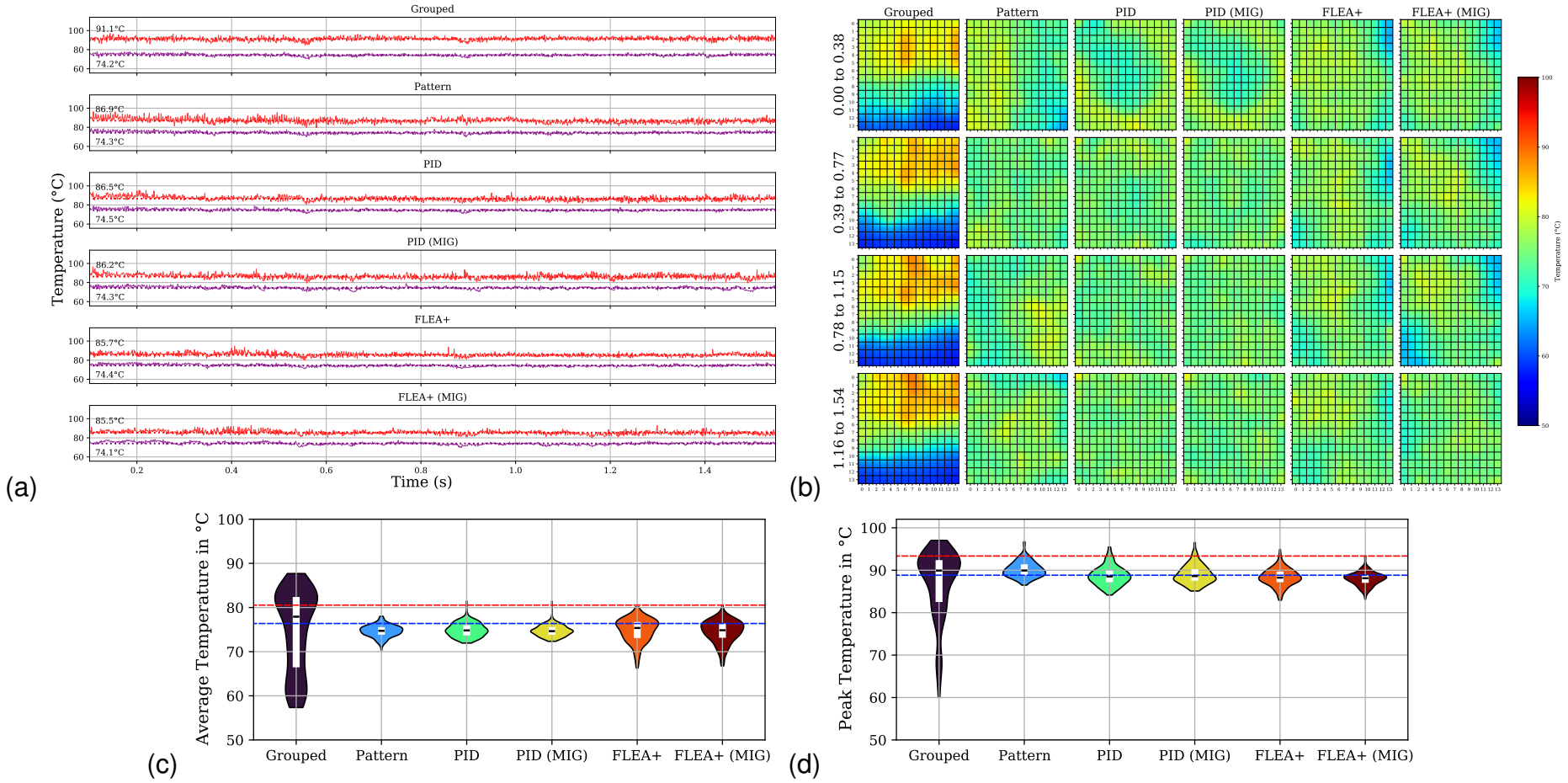


Figure C.23: 14x14 MIXED 90% - (a) average and peak temperatures; (b) average temperature snapshot; (c) PE average temperature violin-box plot; (d) PE average peak temperature violin-box plot.

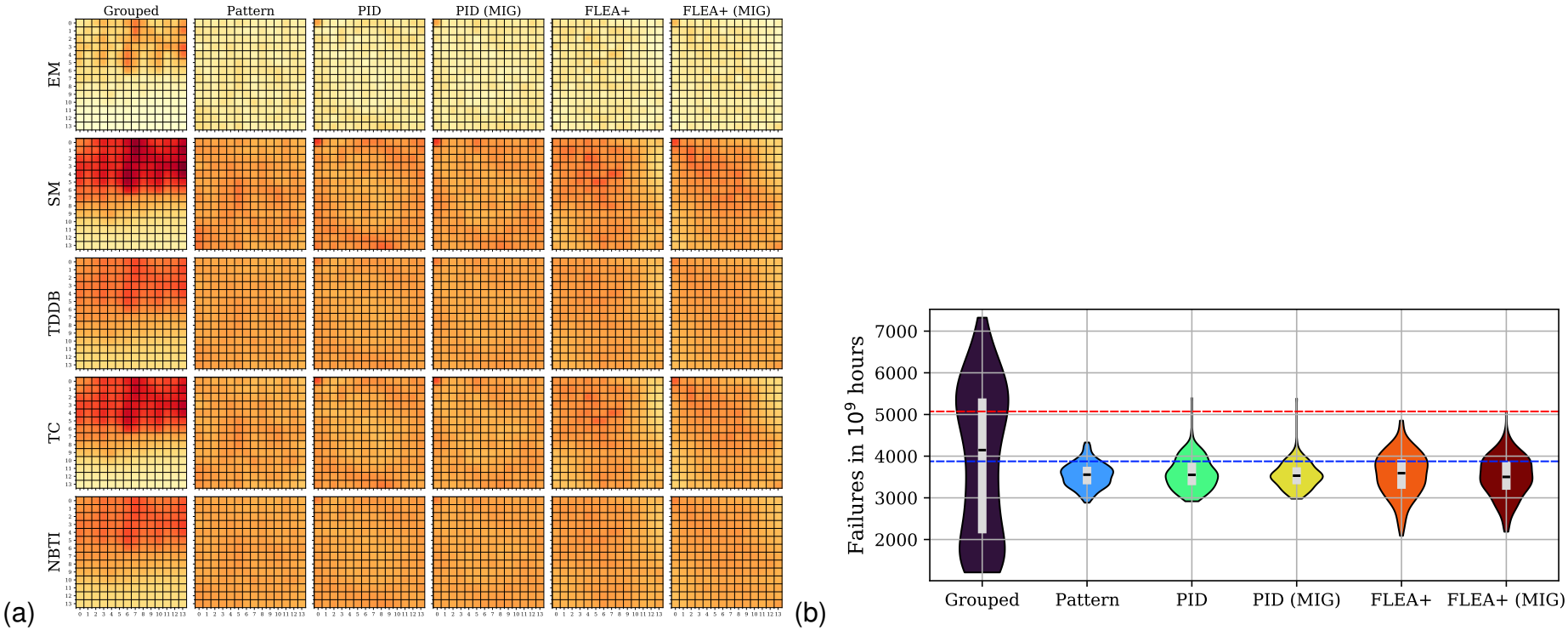


Figure C.24: 14x14 MIXED 90% - (a) Manycore FIT intensity map per evaluated effect; (b) PE failure in time violin-box plot.

C.13 14x14 COMPUTATION 50%

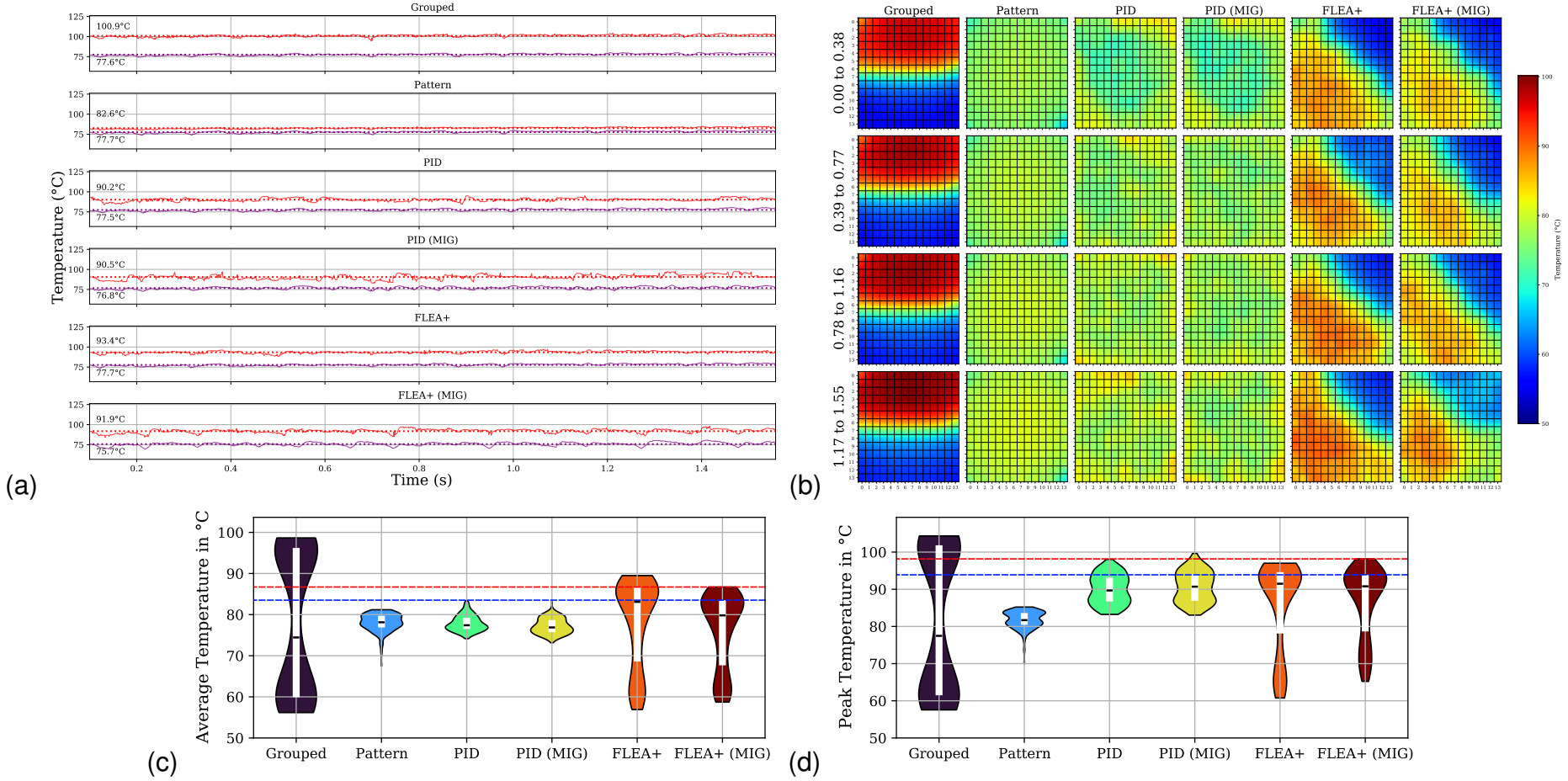


Figure C.25: 14x14 COMPUTATION 50% - (a) average and peak temperatures; (b) average temperature snapshot; (c) PE average temperature violin-box plot; (d) PE average peak temperature violin-box plot.

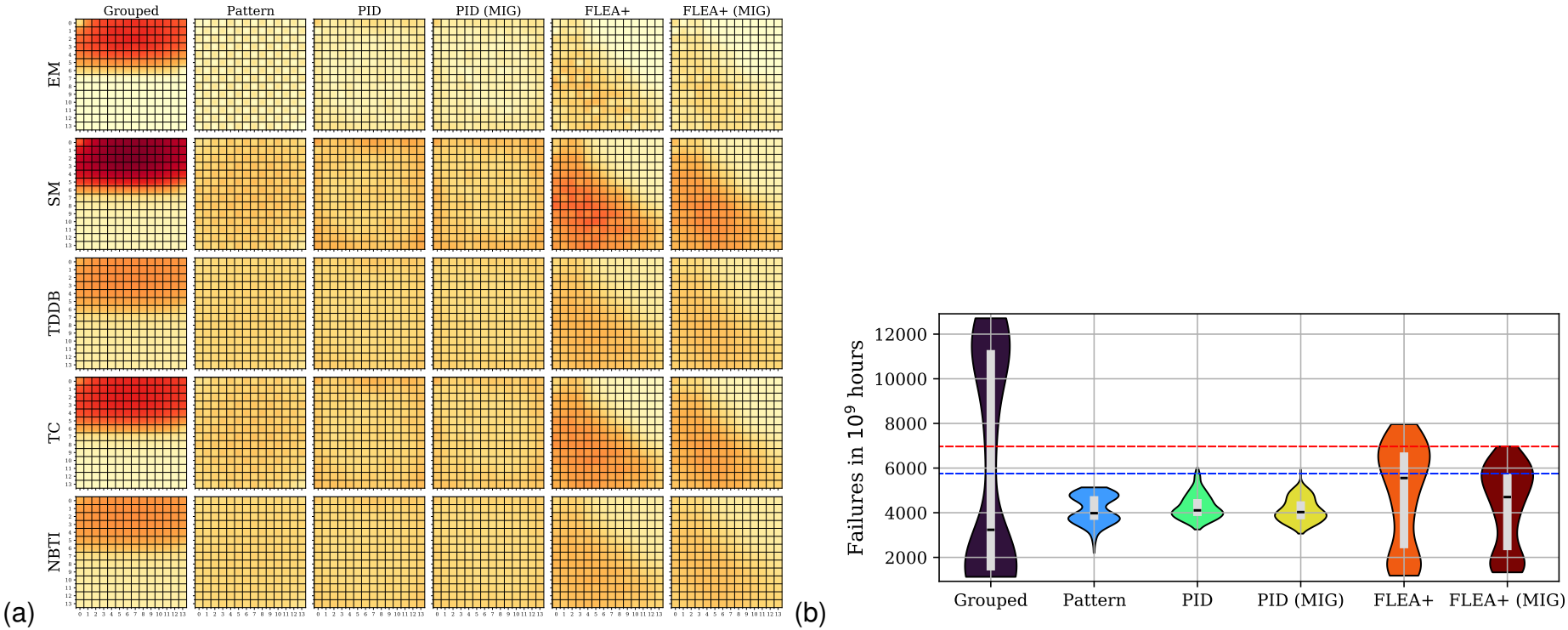


Figure C.26: 14x14 COMPUTATION 50% - (a) Manycore FIT intensity map per evaluated effect; (b) PE failure in time violin-box plot.

C.14 14x14 COMPUTATION 70%

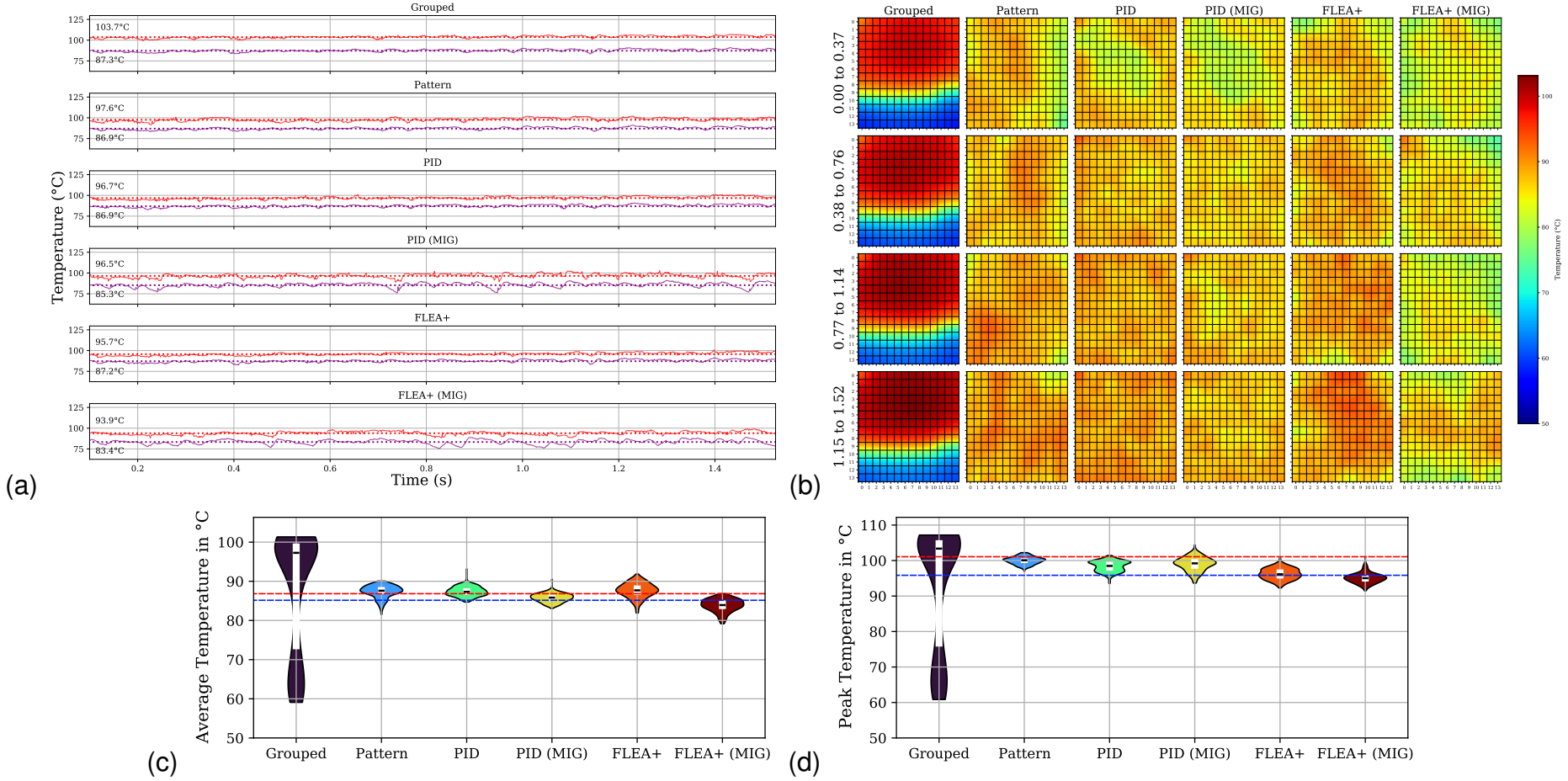


Figure C.27: 14x14 COMPUTATION 70% - (a) average and peak temperatures; (b) average temperature snapshot; (c) PE average temperature violin-box plot; (d) PE average peak temperature violin-box plot.

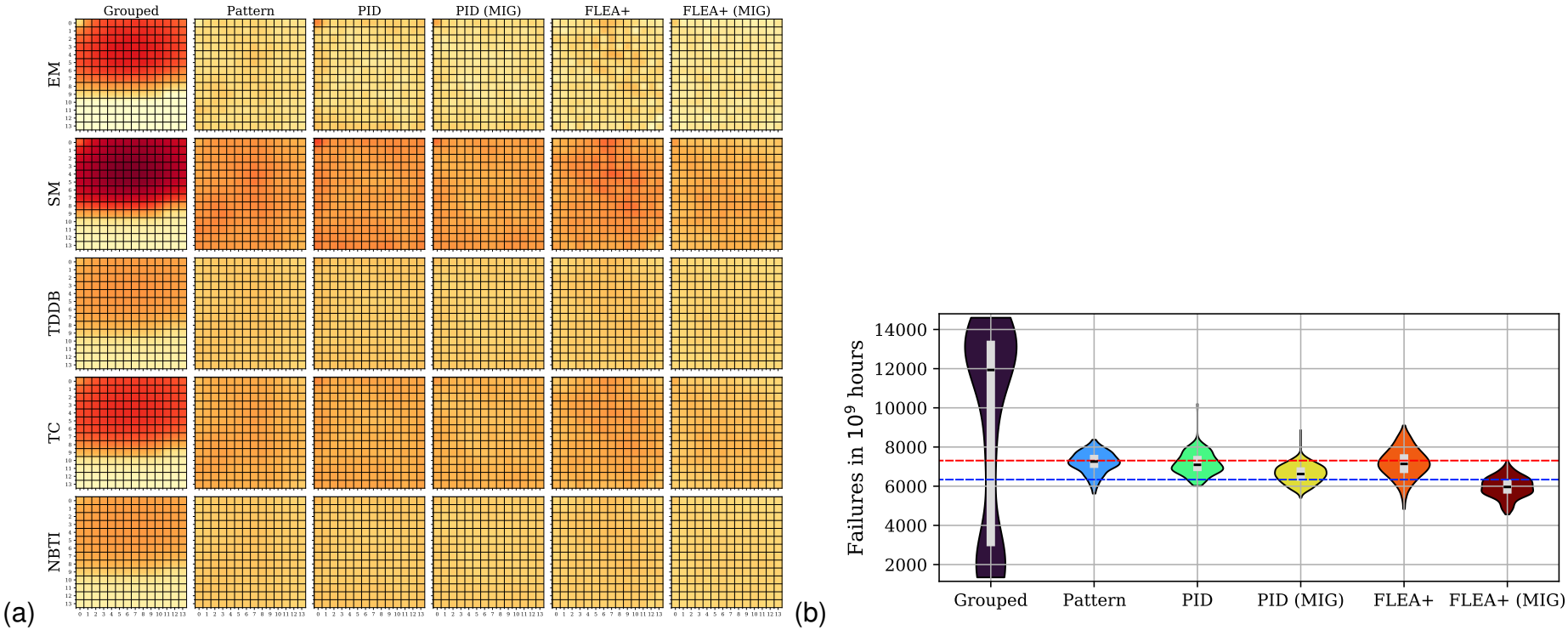


Figure C.28: 14x14 COMPUTATION 70% - (a) Manycore FIT intensity map per evaluated effect; (b) PE failure in time violin-box plot.

C.15 14x14 COMPUTATION 90%

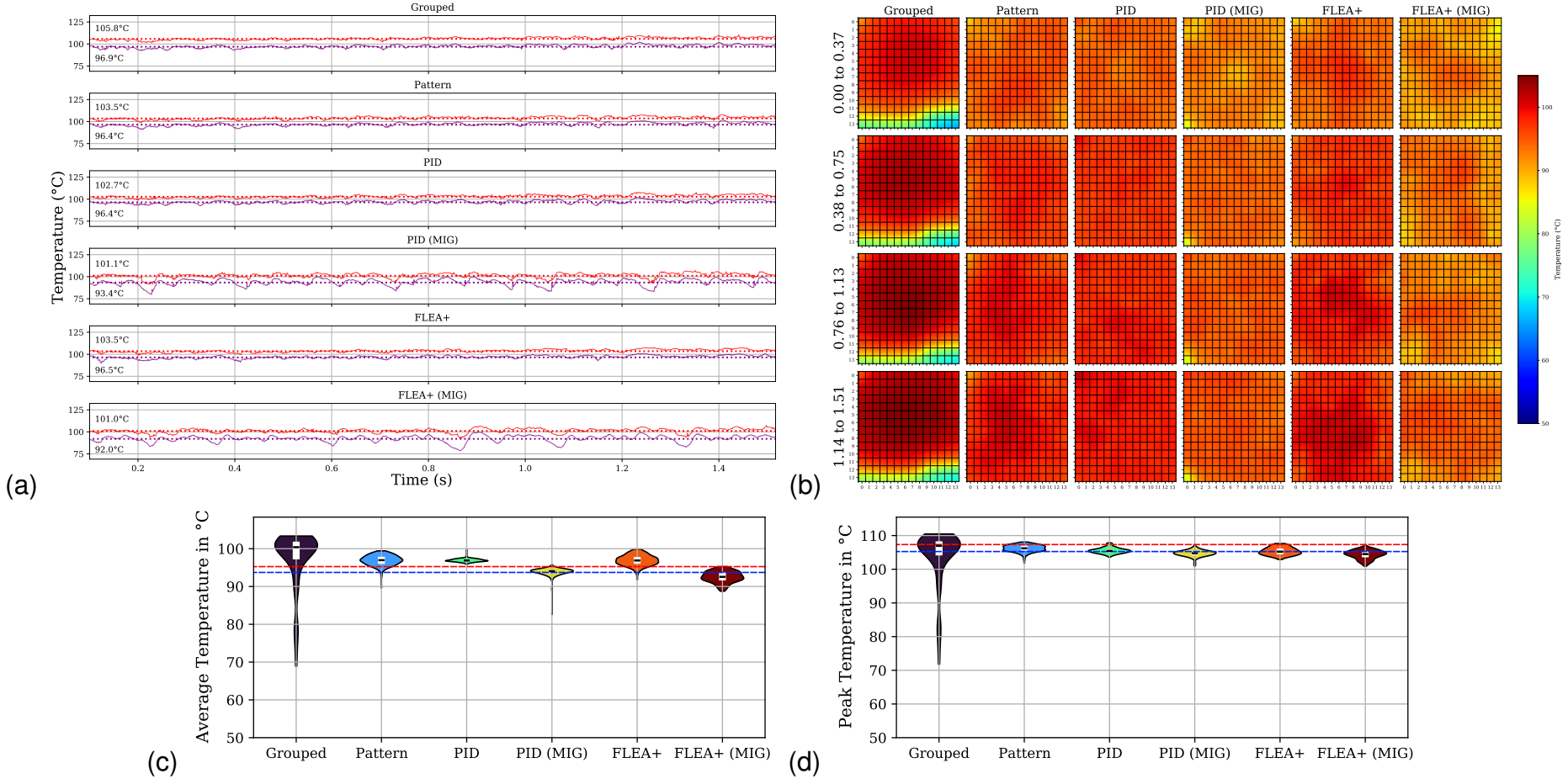


Figure C.29: 14x14 COMPUTATION 90% - (a) average and peak temperatures; (b) average temperature snapshot; (c) PE average temperature violin-box plot; (d) PE average peak temperature violin-box plot.

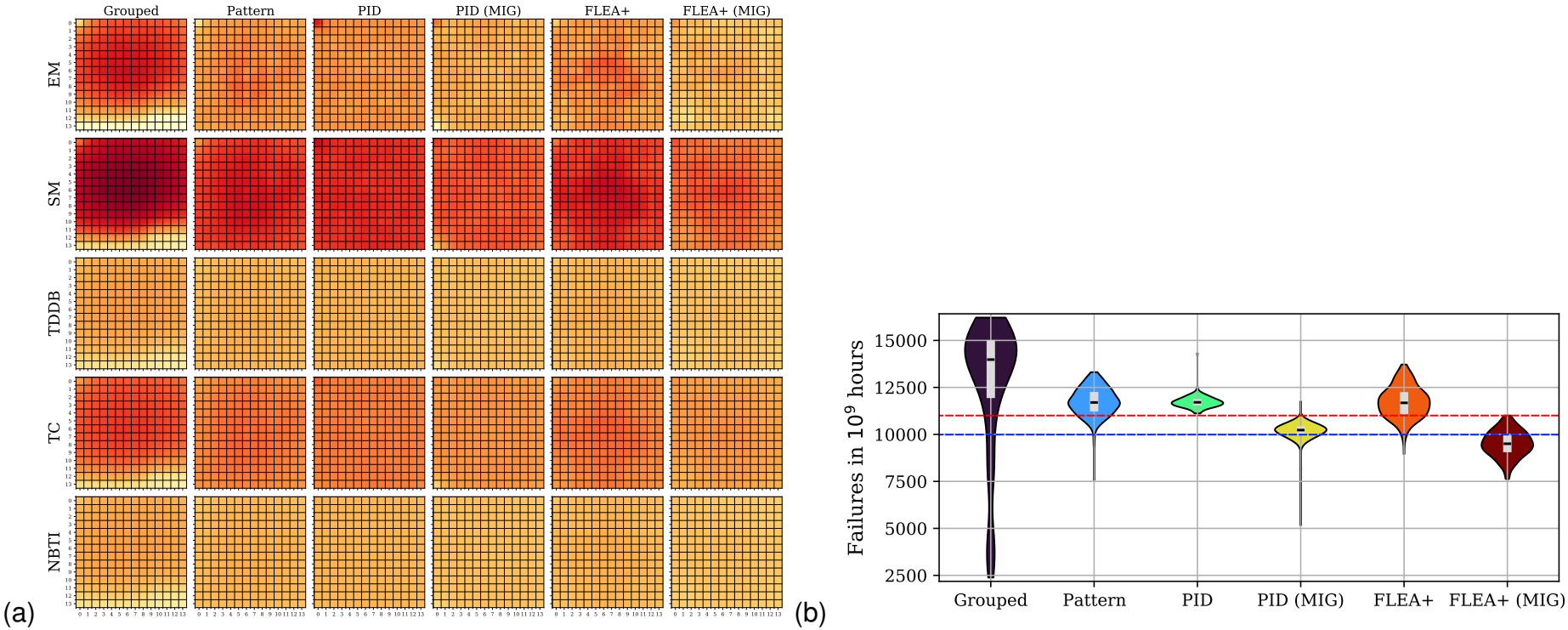


Figure C.30: 14x14 COMPUTATION 90% - (a) Manycore FIT intensity map per evaluated effect; (b) PE failure in time violin-box plot.

C.16 14x14 - RANDOM

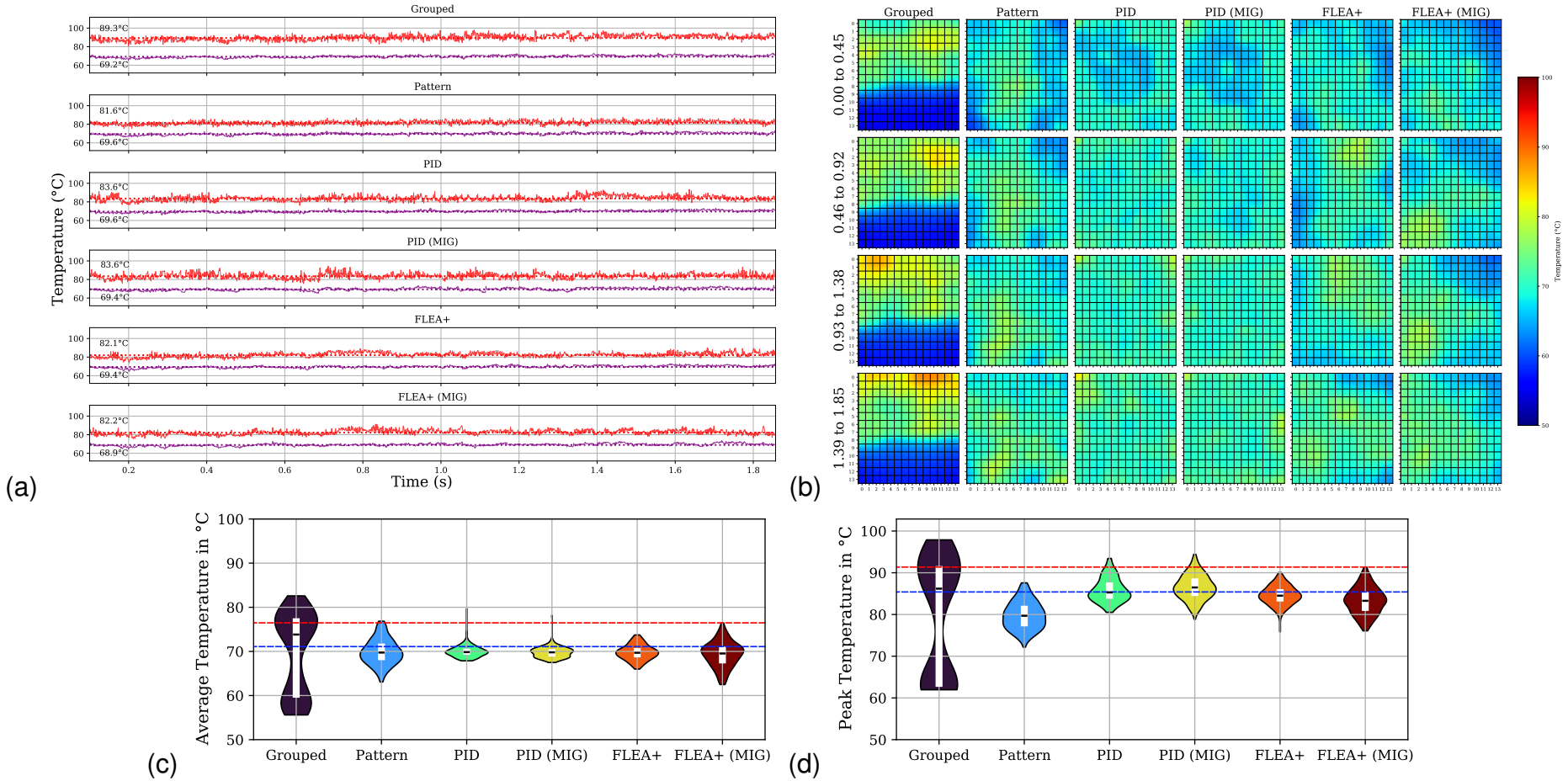


Figure C.31: 14x14 RANDOM - (a) average and peak temperatures; (b) average temperature snapshot; (c) PE average temperature violin-box plot; (d) PE average peak temperature violin-box plot.

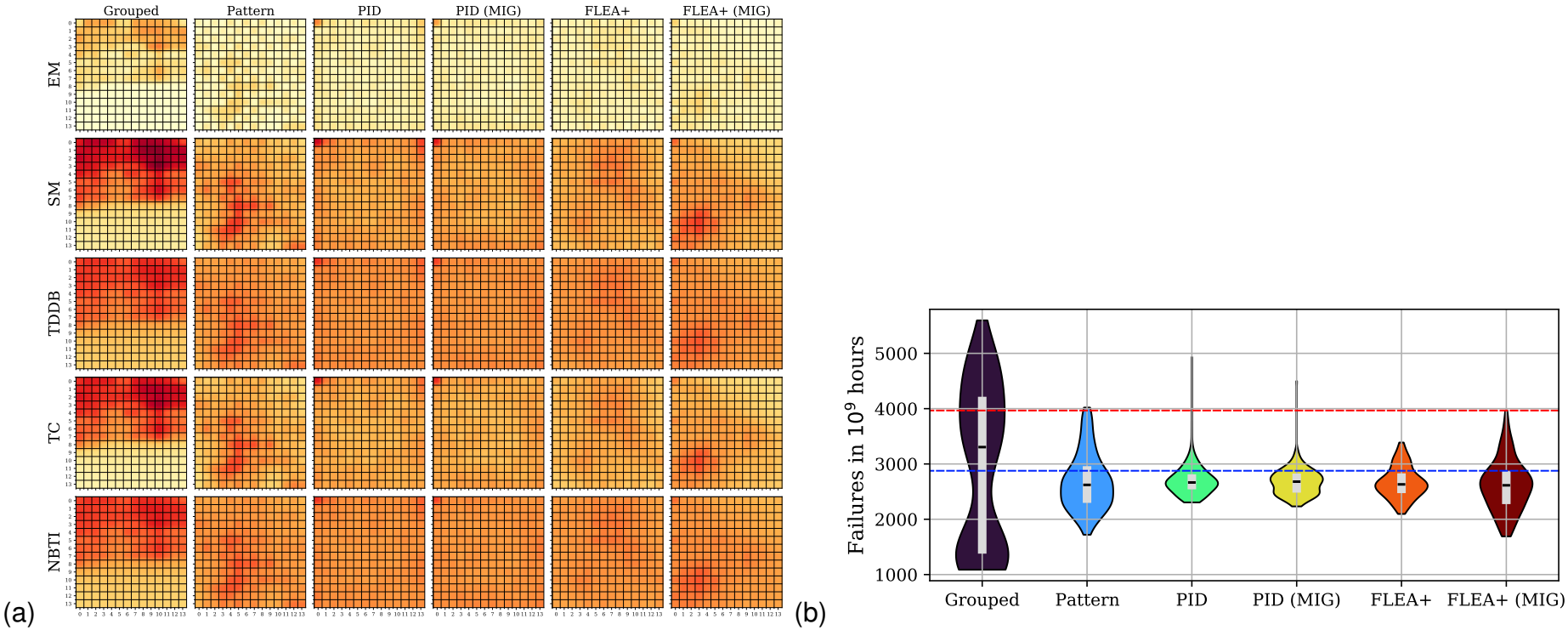


Figure C.32: 14x14 RANDOM - (a) Manycore FIT intensity map per evaluated effect; (b) PE failure in time violin-box plot.

Pontifícia Universidade Católica do Rio Grande do Sul
Pró-Reitoria de Pesquisa e Pós-Graduação
Av. Ipiranga, 6681 – Prédio 1 – Térreo
Porto Alegre – RS – Brasil
Fone: (51) 3320-3513
E-mail: propesq@pucrs.br
Site: www.pucrs.br