UNIVERSITY OF CALIFORNIA,
IRVINE

Performance and Power Optimization for Multi-core Systems using Multi-level Scaling

DISSERTATION

submitted in partial satisfaction of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

in Electrical and Computer Engineering

by

Munirah Almatouq

Dissertation Committee:
Professor Jean-Luc Gaudiot, Chair
Professor Nader Bagherzadeh
Professor Alexandru Nicolau

2019

DEDICATION

To my parents, who taught me to dream big
To my husband and children
Without their love and support I could never achieved my dreams

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

List of Algorithms

ACKNOWLEDGMENTS

It is a pleasure to express my gratitude towards my adviser, Professor Jean-Luc Gaudiot for all his help and patience. His knowledge, integrity, and dedication have inspired me throughout my graduate studies. Many thanks to my committee members, Professor Nader Bagherzadeh and Professor Alexandru Nicolau, for dedicating the time and effort to evaluate my work. It has been an honor to meet and work with them.

I wouldn't be here without the help and support of my family. Thank you Dad for keeping me motivated, for steering me in the right direction, for believing in me, for taking time to talk for hours about my future, and for always being there for me. Thank you Mom for all your prayers and guidance and for helping me with my little ones. I express my deepest gratitude to my husband, Naser Alsanafi for always being there for me. His unconditional love and constant support helped me to be strong.

Many thanks for Lulwah Alhubail, my dearest friend. It is hard to think of anything that she did not help me with. She encouraged and supported me in research. Shoug alsubaihi, thank you for all the advise, for many intellectual discussions and the ideas that came out of them, and for the unforgettable memories. In addition, I extend my thanks to all my colleagues in the PArallel Systems & Computer Architecture Lab (PASCAL).

I would also like to thank those who provided the financial support for my graduate studies. I have been supported by generous grants Kuwait University.

CURRICULUM VITAE

Munirah Almatouq

EDUCATION

Doctor of Philosophy in Computer Engineering **2019**
University of California, Irvine *Irvine, CA*

Masters of Science in Computer Engineering **2008**
Kuwait University *Kuwait*

Bachelor of Science in Computer Engineering **2005**
Kuwait University *Kuwait*

EXPERIENCE

Computer Engineer **2008–2010**
Kuwait Investment Authority *Kuwait*

Computer Engineer **2005–2007**
Kuwait University *Kuwait*

TEACHING EXPERIENCE

Faculty Member **2010–2013**
Public Authority for Applied Education and Training *Kuwait*

HONORS AND AWARDS

Scholarship to pursue Ph.D. in Computer Engineering **2013**
Kuwait Univesity *Kuwait*

Listed in the list of students with honor **2001–2005**
Kuwait Univesity *Kuwait*

Listed in the honorary deans list **2001–2005**
Kuwait Univesity *Kuwait*

ABSTRACT OF THE DISSERTATION

Performance and Power Optimization for Multi-core Systems using Multi-level Scaling

By

Munirah Almatouq

Doctor of Philosophy in Electrical and Computer Engineering

University of California, Irvine, 2019

Professor Jean-Luc Gaudiot, Chair

Integrating more cores per chip to increase the performance of processors has been trending for the past decade. However, this trend cannot be sustained because the reduction in power consumption per core has slowed down while the power budget per chip has not increased. Modern processor chips are becoming so power constrained to the point that not all their devices can be powered at once - this is often referred to as dark silicon. To maximize performance within these power constraints, the system must carefully select the set of resources to be used.

To solve this problem, several power management techniques such as Dynamic Voltage/Frequency Scaling (DVFS), core scaling, and resource scaling have been the subject of active research and have proven to be effective. However, most of these solutions are sub-optimal because they explore only one layer of the architecture. Although considering one layer reduces the complexity of the technique, it limits the exploitation of potential improvement in performance and energy consumption.

The problem is an order of magnitude more complex for power constrained multi-core architectures. We need power management systems that can take advantage of different scaling techniques. Many studies have been conducted on scaling with the sole objective of performance improvement. Nevertheless, few of them have considered both performance and

energy consumption in the optimization process.

This dissertation proposes an optimization technique that balances performance and energy consumption by applying a joint control of core, resource and frequency scaling. This system finds the optimal configuration for a given application and accordingly adapts the architecture configuration.

The proposed technique consists of three stages: configuration sampling, response surface models to approximate performance and energy consumption, and online optimization using a genetic algorithm (GA). To evaluate the system, experiments were conducted on a simulated 12 core architecture. Our experiments have shown that the performance could improve by 15% on average while achieving energy savings of up to 26%. Using a per-core configuration improves the performance by 25% on average and reduces the energy by 18%.

Chapter 1

Introduction

The birth of multi-core architectures was a result of the increasing power requirements of single-core architectures due to the rapid progression of speed and complexity. Moore's Law [41], which refers to the trend of doubling the number of transistors on a chip every 18 months, has been a key factor in the evolution of the microprocessor. With the discontinuation of Dennard scaling [12] (see Figure. 1.1), which led to sharp increases in power densities, powering all transistors simultaneously while keeping the chip temperature in the safe operating range is becoming quite difficult.

We are now at a point where performance and energy consumption are tightly coupled. Studies have shown that future multi-core systems will be able to power on less than 80% of their transistors in the near future, and less than 50% in the long term [14]. This problem requires the system to intelligently select the set of resources that maximizes the performance within the given power budget at all times. The introduction of multi-core architectures brought new challenges to the optimization of performance and energy consumption.

A variety of methods exist to manage power in modern systems. The basic idea of these techniques is to have some scalability in one level of the architecture resources and then

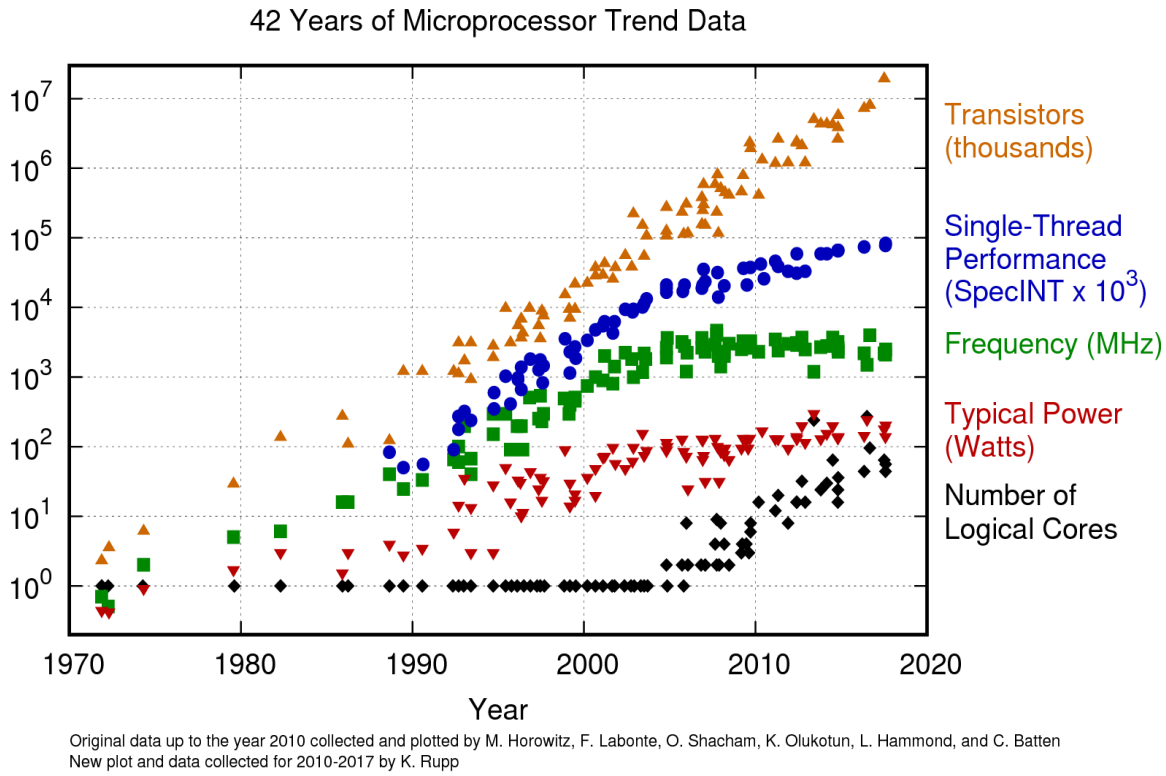


Figure 1.1: Microprocessor trend data, from [1]

dynamically adapt the resource configuration to match different applications demands. Some examples of scalable resources: the voltage/frequency level, the number of active cores, the cache size, the number of execution units, and the size of the queues buffers and registers. The role of the scaling algorithm is to make the decision how much to scale each salable resource to optimize the performance of the running application.

1.1 Power Management Techniques

Dynamic voltage/frequency scaling (DVFS) [21] has been widely used to trade performance with power. In DVFS the processor (or parts of it) is run at a less-than-maximum frequency in order to conserve power. DVFS can be implemented at different granularities, at a coarse grain we have chip-wide DVFS, and at a finer grain, we have per-core DVFS. However, the

efficiency of DVFS is decreasing because of the shift to processors with low voltage margins (near-threshold computing). Besides, when implementing DVFS in large-scale multi-core systems, the large number of dynamically scalable voltage domains makes it less cost effective.

Another power management mechanism is core level gating [32], where the core voltage domain is gated to save power. Power gating technique is applied on the circuit level where the power supply is cut on parts of the circuit. This is implemented using the sleep transistor. One limitation of gating is the latency of turning the circuit back on "wake-up" [37][19]

In addition, a number of power management approaches dynamically scale micro-architecture resources of each core to adapt to application requirements [45]. Other approaches consider cache adaptations to optimize power consumption [40].

Scaling those resources is challenging because there is no defined relationship between performance or energy consumption and the amount of each resource. For example, reducing the number of cores to half does not necessarily reduce the performance or energy by half, and may result in very different performance and energy consumption for different applications. For the scaling algorithm to be effective it should have some insight about the application behavior.

Each of these scaling techniques has its strengths which makes it useful for specific applications or platforms, but each of them has its limitations which makes it less useful for other applications or platforms. While many of these mechanisms have been researched in isolation, integrating them is necessary to achieve a potentially significant reduction in energy consumption. This integration can lead to a great increase in the complexity of the optimization algorithms.

Power management techniques can be implemented in various ways it can be static or dynamic, coarse-grained or fine-grained, the control can be global or distributed, and the technique can be reactive or predictive.

1.2 Goals and Contributions

The ultimate goal of this dissertation is to understand the effectiveness of combining different scaling techniques in improving the performance and reducing the energy consumption of multi-core systems.

This work explores the problem of optimizing both performance and energy consumption of multi-core systems. This is done by applying dynamic scaling techniques at multiple levels of the architecture including core, resource, and voltage/frequency level. Implementing scaling at multiple levels makes the configuration search space multidimensional and complex to navigate. The objective is to create a framework to manage the interplay of the scaling at different levels.

In this research, I propose an online optimization technique that scales the multi-core to the specific requirements of the running application. However, the challenge is how to determine the optimal architecture configuration at any given time. The approach of training, response surface modeling (RSM) and optimization is widely used for design space exploration (DSE) for processor customization and power optimization at design time [35][43][58]. The proposed technique relies on sampling techniques, surface fitting approximation function, and heuristic optimization to find the best configuration. Sampling is used to characterize the behavior of applications by applying different configurations. Then, a response surface model is built to approximate the application characteristics. This model is used in the optimization stage.

The configuration can be either a chip-wide where all the cores are uniformly scaled (homogeneous configuration) or a per-core configuration where each care has a different configuration (heterogeneous configuration).

Overall, I make the following contributions in this research:

- Develop a system that manages and controls the interplay of different power management techniques.
- Develop a system that considers both performance and energy consumption.
- Develop a multi-technique approach, combining sampling, approximation and multi-objective optimization, to explore the reconfiguration search space.
- implement a multi-objective optimizer based on genetic algorithm to explore the chip-wide reconfiguration search space.
- implement a multi-objective optimizer based on genetic algorithm to explore the per-core reconfiguration search space.
- Evaluate the proposed system on a simulated multi-core processor and show that our integrated approach achieves optimized performance while saving energy.
- Compare the effectiveness of applying a per-core configuration to a chip-wide configuration.

1.3 Dissertation Organization

The rest of the dissertation is organized as follows. chapter 2 discusses the existing power management techniques. This chapter presents the related work for each level of scaling and the some of research that integrated multiple scaling techniques. Chapter 3 states the problem formulation and presents the optimization methodology. This chapter explains the details of the different stages of the proposed methodology. The implementation of the optimization algorithm is discussed in Chapter 4. It explains the general dynamics of genetic algorithm and the specifics of its operators. Chapter 5 presents the evaluation methodology including benchmarks, performance and power simulation, the response surface

model implementation, and the experimentation setup. Chapter 6 describes the overhead of the proposed system including the optimization and reconfiguration overhead. Chapter 7 discusses the different experiments that were conducted to evaluated the proposed system for both the chip-wide configuration and the per-core configuration. Finally, Chapter 8 concludes this work and discusses future work.

Chapter 2

Related Work

2.1 DVFS

Several studies have been conducted on DVFS for power management. Isci et al. [21] developed different policies for global multi-core power management considering prioritization and throughput. Sharkey et al. [49] implemented coarse-grained chip-wide DVFS with fine-grained per-core fetch throttling and explored tradeoffs between local and global control. Bergamaschi et al. [5] compared per-core against chip-wide approaches and discrete versus continuous algorithms. Liu et al. [36] defines an approach to maximize the performance under a system-wide power cap considering both CPU and memory DVFS. Ravi et al. [48] proposed an integrated power gating and DVFS within the core while considering the power consumed by each node in the clock tree hierarchy. Jayaseelan et al. [23] proposed a hybrid local-global thermal management approach for multi-core systems that use global DVFS across all the cores then locally tune the performance of each core individually through architectural adaptations. Li et al. [32] considered two levels of scaling, changing the number of active cores and applying DVFS to maximize performance under a power constraint.

Eyerman et al. [15] evaluated the potential of applying fine-grain DVFS and proposed a fine-grained DVFS mechanism. Kim et al. [25] also, demonstrated the benefit of per-core DVFS for embedded processors. Teodorescu et al. [52] developed a variation-aware power management DVFS algorithms to maximize throughput at a given power budget. they used linear programming to find the best voltage and frequency levels for each of the cores in the multi-core system. Rangan et al. [47] proposed a thread migration technique where threads that require different V/F levels for power-efficient operations, are migrated to the cores that can provide an appropriate performance level instead of changing the V/F of the cores. One disadvantage of DVFS is that it can be less effective as processors shift to lower voltage margins (near-threshold computing).

2.2 Resource Scaling

A number of researches have focused on resource scaling techniques that adapt the resource of the core(s) to the application. Kontorinis et al. [27] proposed a table-driven adaptive resource scaling technique to guarantee that the peak power consumption of a processor is far lower than the sum of all core blocks. Iyer et al. [22] used a run-time profiling to optimize the configuration based on detecting the parts of the running application which have good potential for energy savings. Albonesi et al. [4] proposed using adaptive processing to dynamically tune major microprocessor resources by disabling underutilized hardware to improve performance or power efficiency. Dubach et al. [13] used a control mechanism based on a predictive model for micro-architectural adaptation. The model controls the adaptability by monitoring the behaviour of the application in different phases. Lee et al. [29] proposed a framework that is able to analyze the performance and power characteristics of adaptive micro-architectures. Huang et al. [20] propose a positional approach that uses program subroutines as the granularity for reconfiguration. Hu et al. [19] scheme for management

of multiple configurable units, utilizing the inherent capabilities of dynamic optimization systems. Hotspot are used for phase detection and adaptation. Petrica et al. [45] proposed a general-purpose multi-core architecture that dynamically adapts to varying and stringent power budgets. The micro-architecture includes reconfigurable horizontal lanes through the pipeline that allow adapting individual cores to the running application. Bitirgen et al. [8] developed a framework that manages multiple shared resources using machine learning. Meng et al.[38] described a global optimization power management framework for multi-core architectures that applies a greedy algorithm to examine the search-space and find operating points that offer good power/performance compromises. Gibson et al. [17] presented Forward flow, a scalable core design for power-constrained multi-core leveraging a modular instruction window. Forward flow represents inter-instruction dependencies via a linked list of forward pointers.

2.3 Core Scaling

Core scaling is another widely used power management technique. Some studies have been developed to determine when to shut down cores. Liu et al. [34] addressed the problem of minimizing the power dissipation of many-core systems under performance constraints by exploiting per-core DVFS with core scaling. Ghasemi et al.[16] developed a technique that simultaneously and uniformly scales the resources that are associated with each core and the number of operating cores to maximize the performance of power-constrained multi-core. Vega et al. [55] presented PAMPA a measurement-based evaluation of power management policies available in modern multi-core systems such as DVFS, core folding, and per-core power gating. It proposed coordination of the power management activities in the system to improve the robustness. Lee et al.[31] considered power-constrained GPUs and optimized the number of operating cores, the voltages and frequencies of cores, on-chip interconnects

and caches according to the application characteristics. Lee et al.[30] analyzed the effect of applying per-core power gating and DVFS on the throughput of power-constrained multi-core processors running applications with limited parallelism.

Jha et al. [24] presented an integrated power management that used Pareto-optimal per-core configurations, followed by global utility-based power allocation to reallocate power to the cores/threads. Micolet et al. [39] studied the potential for dynamic reconfiguration of multi-core processors at runtime using linear regression model to decide the number of cores to fuse at runtime to optimize for performance.

Our work differs from prior work in that it combines core scaling, resource scaling and DVFS in one optimization problem. For a given application, the goal is to find the best combination of the number of cores, resources in each core and voltage/frequency level while considering both the overall system performance and energy consumption. Previous studies have only considered overall system either performance or energy consumption.

Chapter 3

Optimization Methodology

The targeted multi-core architecture, as shown in Fig.3.1, has a number of adjustable knobs and the setting of these knobs define a unique configuration which represents a distinct power-performance trade-off. Hence, the role of the scaling algorithm is essential to determine the optimal configuration. The knobs that are considered for scaling can be classified into three levels: the core level, the resource level, and the Voltage/Frequency (V/F) level. Finding the optimal configuration can become a complex task as the number of cores and number of scalable resources increases and as the configuration need to be found fast.

Moreover, Adjusting these knobs is a challenge since there is no clear relationship or formula between performance or energy and the scale of each knob. Different configurations exhibit different behaviors in terms of performance and energy consumption. For an application that has a high thread level parallelism, using more cores can improve performance. On the contrary, for an application that has a high instruction level parallelism, using more resources can improve performance. In addition, for a specific application two different configurations with the same performance can consume different amounts of energy and vice versa. To find the optimal configuration the scaling algorithm must search through a wide rage of possible

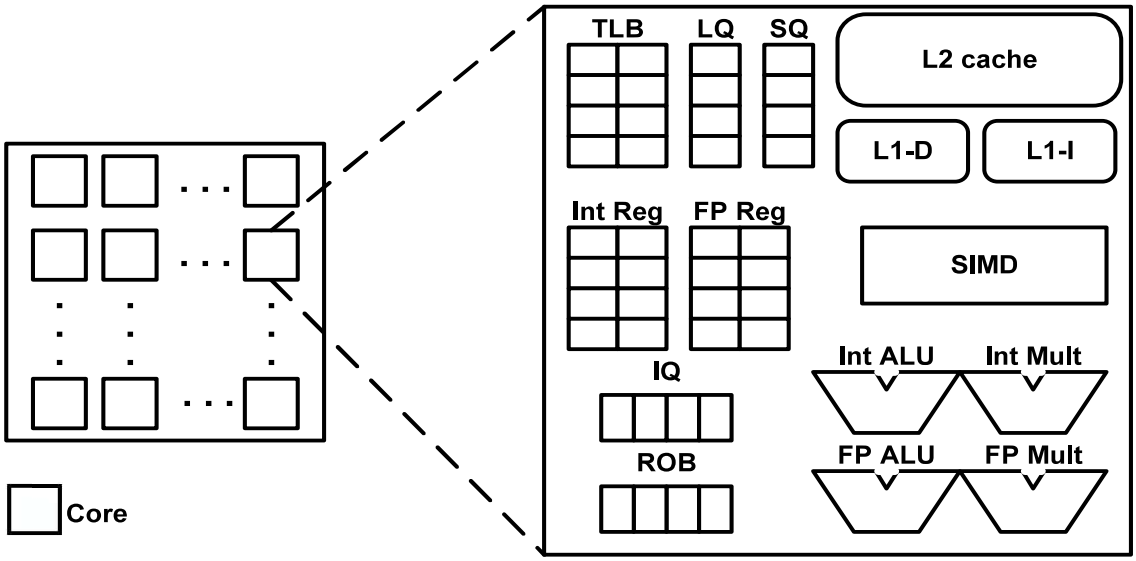


Figure 3.1: Multi-core architecture with the adjustable knobs in each core.

configurations.

3.1 Problem Formulation

The objective of the proposed optimization is to determine the number of cores, the scale of resources in each core and, the V/F level that maximizes the performance while consuming the least energy possible. The search space of this problem is multidimensional with local optima, and heuristic algorithms are known to be efficient in searching such complex spaces. Evolutionary Algorithms [18] refers to a group of heuristic methods that use mechanisms of biological evolution. They differ from other heuristics by selecting the fittest individuals in a population for reproduction and applying crossover and mutation to produce offspring. The fittest individuals evolve to next generations.

This optimization problem has two types of variables, factors (inputs) and responses (outputs). The factors are the variables that determine the configuration of the system while the responses are the measured output values of the system. In this work, a number of factors

Table 3.1: Configuration factors and corresponding values

Level	Factors	Values
Architecture	Number of cores	8, 10, 12
Cache	L2 cache size (MB)	4, 8
	L1-I cache size (KB)	16, 32
	L1-D cache size (KB)	16, 32
Core	TLB size	16, 32
	LQ entries	16, 32
	SQ entries	16, 32
	ROB entries	32, 64
	IQ entries	32, 64
	Int ALU	2, 4
	FP ALU	1, 2
	Int Mult	1, 2
	FP Multi	1, 2
	SIMD	1, 2
	Int Reg	128, 256
	FP Reg	128, 256
Voltage, Frequency	F/V (GHz/ V)	2.7/0.85, 3.2/0.9, 3.6/0.95

are considered. The factors vary from core, resource, and V/F level (see Table 3.1) and two responses the performance which can be either execution time or throughput and the energy consumption. The problem formulation is as follows.

Inputs:

- A set of Applications $A = \{a_1, a_2, \ldots, a_m\}$
- A set of the number of cores $C = \{c_1, c_2, \ldots, c_n\}$

- A set of the resource combinations $R = \{r_1, r_2, \ldots, r_s\}$
- A set of the Voltage/Frequency levels
  $V = \{v_1, v_2, \ldots, v_p\}$

Responces:

- Execution Time function: $T_1(a_i, c_j, r_k, v_l)$ which evaluates the execution time of running application a_i using c_j cores, each with r_k resources at voltage/frequency level v_l.
- Throughput function: $T_2(a_i, c_j, r_k, v_l)$ which evaluates the throughput of running application a_i using c_j cores, each with r_k resources at voltage/frequency level v_l.
- Energy consumption function: $E(a_i, c_j, r_k, v_l)$ which evaluates the energy of running application a_i using c_j cores, each with r_k resources at voltage/frequency level v_l.

Objective: In the case of a single objective optimization problem, the fitness of a solution can be easily determined by the value of the objective function. For our problem there are a variety of single objectives that can be considered:

- Maximize Throughput $T_2()$.
- Minimize Execution Eime $T_1()$.
- Minimize Energy Consumption $E()$.

However, this is a multi-objective optimization problem that considers both the overall performance (execution time/throughput) and energy consumption of a solution. Hence, a weighted fitness function f is used to combine both objectives.

$$f = w\bar{T} + (1 - w)\bar{E}; w = [0, 1] \tag{3.1}$$

where $\bar{T}$ and $\bar{E}$ are normalized to the best values reached. w is a weight value that determines the importance of each of the objective functions.

$$w \begin{cases} = 0 & f = \bar{E} \\ < 0.5 & \bar{E} \text{ is more important than } \bar{T} \\ = 0.5 & \bar{E} \text{ is as important as } \bar{T} \\ > 0.5 & \bar{T} \text{ is more important than } \bar{E} \\ = 1 & f = \bar{T} \end{cases}$$

The optimization of such objectives using traditional optimization methods directly to these functions is unfeasible. The reason is that simulation-based objective functions are often discontinuous and non-differentiable. Another obstacle of optimization based on simulation is the high computational cost simulations. Simulating a single configuration can take several hours or even days. Response surface model(or surrogate models) [53] offer a less expensive solution to handle these unmanageable functions. When using surrogate models, the optimization of the original objective is replaced by optimizing the computationally less expensive surrogate function. Surrogate models are used to approximate both response functions.

- Execution Time function: $\hat{T}_1(a_i, c_j, r_k, v_l)$ which estimates the execution time of running application a_i using c_j cores, each with r_k resources at voltage/frequency level v_l.
- Throughput function: $\hat{T}_2(a_i, c_j, r_k, v_l)$ which estimates the throughput of running application a_i using c_j cores, each with r_k resources at voltage/frequency level v_l.
- Energy consumption function: $\hat{E}(a_i, c_j, r_k, v_l)$ which estimates the energy of running application a_i using c_j cores, each with r_k resources at voltage/frequency level v_l.

The optimization process works by first examining the application behavior and then reconfigure the architecture to accommodate its requirements. To find the best configuration for a specific application, this process is implemented in three stages. First, configuration sampling, second performance and energy approximation, then optimization. These stages are discussed in details in section 3.2, section 3.3 and section 3.5 respectively.

Fig. 3.2 shows the different stages of the optimization process. The system starts with an application running on the multi-core processor. To characterize the application behavior, different configuration samples are applied and for each sample, the performance and energy consumption are recorded. Sampling techniques define a way to select sample points in the search space with the objective of maximizing the amount of information captured. Next, a surrogate model is constructed using the data obtained by sampling the search space to estimate the response functions. Finally, using the surrogate functions, the optimization algorithm determines the configuration that optimizes both performance and energy consumption and reconfigures the multi-core accordingly.

To find the optimal configuration for any application, it is important to understand the application characteristics and the way it behaves under different configurations. In the sampling stage, the behavior of the application and its hardware resource needs are recorded by changing the configuration and executing the application for periods of time with different configurations. Each application is profiled offline where it runs for a period of time under each sample configuration and its performance and energy consumption is stored. The collected data samples are used to build a response surface model that estimates the application characteristics.

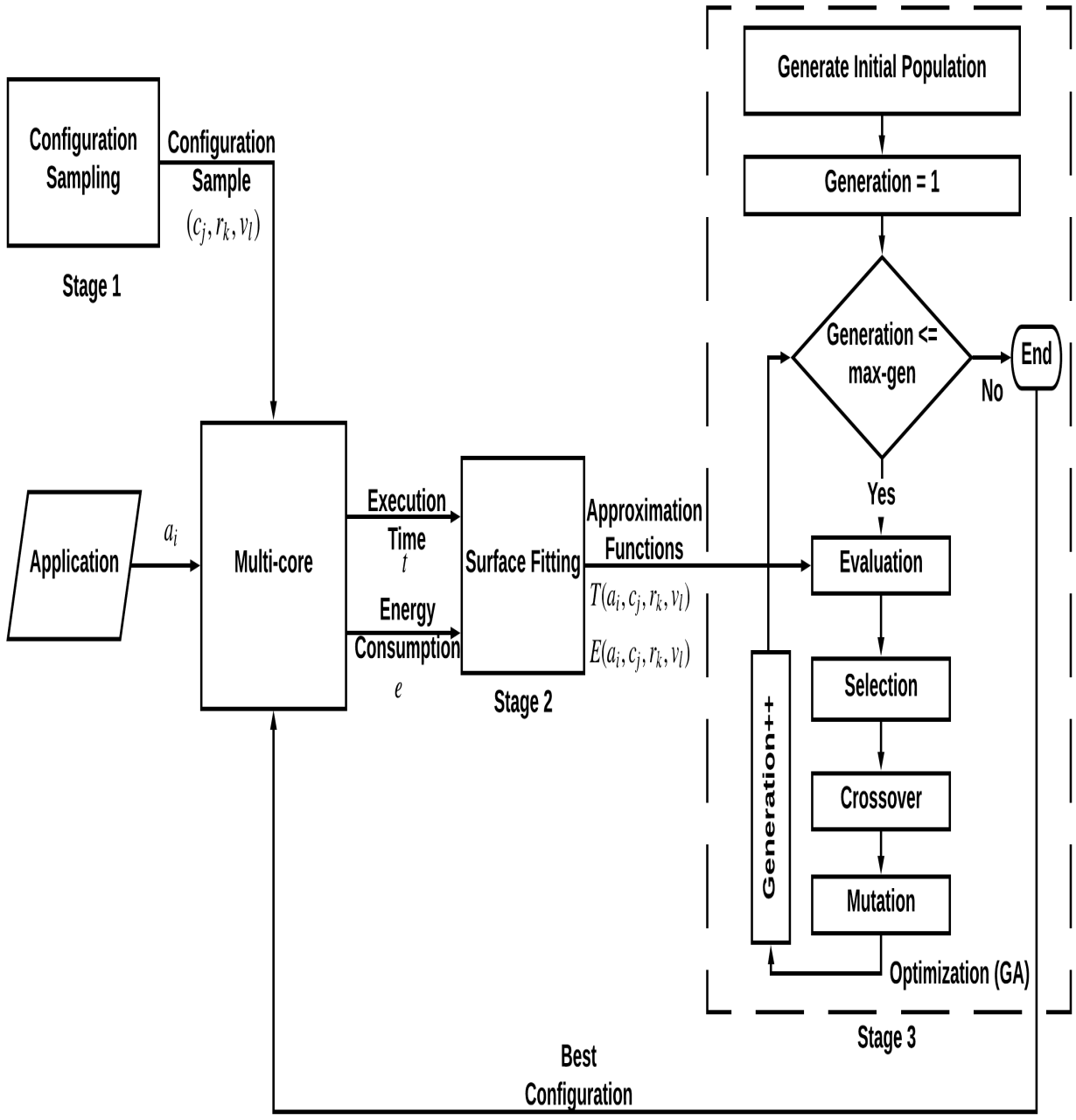


Figure 3.2: Optimization Methodology consists of three stages : Configuration Sampling, Surface Fitting, and Optimization

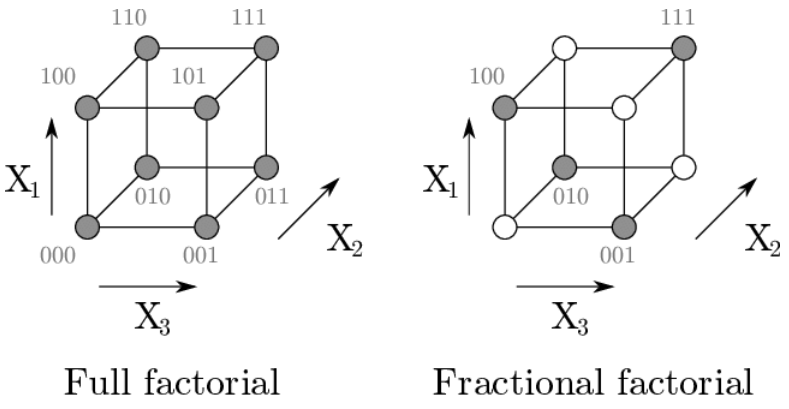


Figure 3.3: Sample points for the Full Factorial (left), and Fractional Factorial (right) designs. from [3]

3.2 Configuration Sampling

There are a number of aspects of the sampling process that affect the accuracy of the resulting response surface model. The first variable is the length of the sample, which is the amount of time the application run with a specific configuration. Another critical variable is the number of sample points or the number of configuration to sample. Both variables are important because they affect the accuracy of the response surface model and the time it takes to profile each application.

The most straightforward approach for sampling is the full factorial design, where all combinations of the factors are sampled and their effect on the response variables is measured. The advantage of the full factorial design is that it gives the most accurate response surface model because it captures both the individual effects of each factor and the interactions between the factors. However, for a large number of factors, the full factorial design will need

a huge number of samples. Moreover, the interaction of some factors does not significantly affect the response variables, hence it can be omitted. In addition when using simulation, the number of sample configurations per application is limited by the simulation time.

There are a number of alternative sampling methods used to reduce the cost of the sampling stage. One method is the Fractional Factorial design [57] which, consists of a subset of the full factorial design. This design method is based on the sparsity-of-effect principle [56] which refers to the idea that a system is usually dominated by the single factor effects and two-factor interactions, and a higher order interaction such as three-factor interactions are very rare. This means only a few effects in a factorial design are significant and should be considered. Fig 3.3 shows the sample points of a full factorial design and a fractional factorial design for 3 factors.

Taguchi Orthogonal Array (OA) design is a type of fractional factorial design. It is a highly fractional orthogonal design. It considers a selected subset of combinations of multiple factors at multiple levels. Taguchi (OA) is balanced to ensure that all levels of all factors are considered equally. For this reason, the factors can be evaluated independently of each other.

The targeted system has two types of factors, 2-level factors and 3-level factors (see Table 3.1). For sampling the 2-level factors (cache and resources), we used Taguchi $L16(2^{15})$ design. This design consists of 15 factors at 2 levels each. Resulting in 16 configurations (see Table 3.2).

Then we used a full factorial design for the 3-level factors (number of cores, V/F level). Resulting in $(3^2) = 9$ configurations, see table 3.3. To get all possible combinations, the overall number of samples is $16 * 9 = 144$ samples.

In the configuration sampling stage, an application a_i runs for short intervals that have the same number of operations for fairness, typically corresponding to 40- to 100- million instruc-

Table 3.2: Taguchi $L16(2^{15})$ design

L2	L1-I	L1-D	TLB	IntALU	IntMult	FPALU	FPMult	SIMD	LQ	SQ	ROB	IQ	IntReg	FPReg
4	16	16	16	4	2	2	2	2	32	16	32	32	128	256
8	16	16	16	2	1	1	2	2	32	32	64	64	128	128
4	32	16	16	2	2	2	1	1	32	32	64	32	256	128
8	32	16	16	4	1	1	1	1	32	16	32	64	256	256
4	16	32	16	4	1	2	1	2	16	32	32	64	256	128
8	16	32	16	2	2	1	1	2	16	16	64	32	256	256
4	32	32	16	2	1	2	2	1	16	16	64	64	128	256
8	32	32	16	4	2	1	2	1	16	32	32	32	128	128
4	16	16	32	4	2	1	2	1	16	16	64	64	256	128
8	16	16	32	2	1	2	2	1	16	32	32	32	256	256
4	32	16	32	2	2	1	1	2	16	32	32	64	128	256
8	32	16	32	4	1	2	1	2	16	16	64	32	128	128
4	16	32	32	4	1	1	1	1	32	32	64	32	128	256
8	16	32	32	2	2	2	1	1	32	16	32	64	128	128
4	32	32	32	2	1	1	2	2	32	16	32	32	256	128
8	32	32	32	4	2	2	2	2	32	32	64	64	256	256

tions depending on a benchmark and its runtime. Each interval has a different configuration (c_j, r_k, v_l).

After the execution of each sample interval, the execution time t and energy e consumption responses are obtained to build the response surface model.

Having selected the sampling technique and sampled the data, the next step is to build an approximation model and a fitting methodology. The next section describes in detail different surrogate modeling techniques.

Table 3.3: Full Factorial design

Cores	V/F
8	2.7/0.85
8	3.2/0.9
8	3.6/0.95
10	2.7/0.85
10	3.2/0.9
10	3.6/0.95
12	2.7/0.85
12	3.2/0.9
12	3.6/0.95

3.3 Response Approximation

Due to the absence of accurate models or functions that evaluate the performance and energy consumption of the multi-core system while considering all the factors we are scaling in this study, we instead use a response surface model to get a computationally inexpensive approximation of the response functions. Response surface models are inexpensive approximations of computationally expensive functions that we need to be optimized. In the proposed approach, each application is characterized by sampling a number of factor combinations and measuring the system response at each sample point. The main objective of using a response surface model is to construct a response function from a small subset of function evaluations. The two functions that need to be approximated are $T(a_i, c_j, r_k, v_l)$and $E(a_i, c_j, r_k, v_l)$. The next subsections explain various response surfaces models.

3.3.1 First Order Polynomial

The first order polynomial or linear function is the simplest and least time-consuming response surface model. In this model, the response function can be described as

$$\hat{y} = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3$$

It is used to efficiently optimize linear functions with linear constraints. Polynomial response surface model assumes a linear relation between the factors and the responses. In our case, the distribution of the data was more complex.

3.3.2 Second Order Polynomial

Low-order polynomial are also popular response surface models widely used in scientific areas because of its simplicity. [9]. In this model the response function can be described as

$$f(x) = \hat{y} + \epsilon$$

where $f(x)$ is the response, $\hat{y}$ is the model value, and ϵ is the error. A second order polynomial surrogate function is described

$$\hat{y} = \beta_0 + \sum_{i=1}^{k} \beta_i x_i + \sum_{i=1}^{k} \sum_{j<i}^{k} \beta_{ij} x_i x_j + \sum_{i=1}^{k} \beta_{ii} x_i^2$$

For three factors this can be expand to

$$\hat{y} = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \beta_{12} x_1 x_2 + \beta_{13} x_1 x_3 + \beta_{23} x_2 x_3 + \beta_{11} x_1^2 + \beta_{22} x_2^2 + \beta_{33} x_3^2$$

Assuming that the number of sample point is n, the system can be described as

$$Y = \beta X$$

where Y is a 1 by n column vector of the measured responses, X is a matrix of the factor values, and β is the vector of the coefficients.

$$Y = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ \cdot \\ \cdot \\ \cdot \\ y_{n-2} \\ y_{n-1} \\ y_n \end{bmatrix} \qquad \beta = \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ \beta_3 \\ \beta_{11} \\ \beta_{22} \\ \beta_{33} \\ \beta_{12} \\ \beta_{13} \\ \beta_{23} \end{bmatrix}$$

$$X = \begin{bmatrix} 1 & x_{1_1} & x_{2_1} & x_{3_1} & x_{1_1}^2 & x_{2_1}^2 & x_{3_1}^2 & x_{1_1}x_{2_1} & x_{1_1}x_{3_1} & x_{2_1}x_{3_1} \\ 1 & x_{1_2} & x_{2_2} & x_{3_2} & x_{1_2}^2 & x_{2_2}^2 & x_{3_2}^2 & x_{1_2}x_{2_2} & x_{1_2}x_{3_2} & x_{2_2}x_{3_2} \\ 1 & x_{1_3} & x_{2_3} & x_{3_3} & x_{1_3}^2 & x_{2_3}^2 & x_{3_3}^2 & x_{1_3}x_{2_3} & x_{1_3}x_{3_3} & x_{2_3}x_{3_3} \\ 1 & x_{1_4} & x_{2_4} & x_{3_4} & x_{1_4}^2 & x_{2_4}^2 & x_{3_4}^2 & x_{1_4}x_{2_4} & x_{1_4}x_{3_4} & x_{2_4}x_{3_4} \\ . & & & & & & & & & \\ . & & & & & & & & & \\ . & & & & & & & & & \\ 1 & x_{1_{n-2}} & x_{2_{n-2}} & x_{3_{n-2}} & x_{1_{n-2}}^2 & x_{2_{n-2}}^2 & x_{3_{n-2}}^2 & x_{1_{n-2}}x_{2_{n-2}} & x_{1_{n-2}}x_{3_{n-2}} & x_{2_{n-2}}x_{3_{n-2}} \\ 1 & x_{1_{n-1}} & x_{2_{n-1}} & x_{3_{n-1}} & x_{1_{n-1}}^2 & x_{2_{n-1}}^2 & x_{3_{n-1}}^2 & x_{1_{n-1}}x_{2_{n-1}} & x_{1_{n-1}}x_{3_{n-1}} & x_{2_{n-1}}x_{3_{n-1}} \\ 1 & x_{1_n} & x_{2_n} & x_{3_n} & x_{1_n}^2 & x_{2_n}^2 & x_{3_n}^2 & x_{1_n}x_{2_n} & x_{1_n}x_{3_n} & x_{2_n}x_{3_n} \end{bmatrix}$$

The first and second order polynomial functions require a large number of data points. Moreover, the approximated values are not necessarily equal to the actual values at the sampling points (non-interpolating). In general, Radial Basis Functions (RBFs) present a powerful solution to the problem of scattered data fitting, where some samples are given as data points, and we want to approximate the response at new points [42].

3.3.3 Radial Basis Functions

Radial basis function is an interpolating model that uses a combination of radially symmetric functions [10]. The radial basis function value depends on the Euclidean distance from the center c. This method works in an d dimensional Euclidean space R^d. Assuming that there are n sample points in this space $x_1, x_2, , x_n in R^d$ for which the response function $f(x_1), f(x_2), , f(x_n)$ are known. This function is unknown except at those n points. In this model the response function can be described as

$$\hat{y} = \sum_{i=1}^{n} \lambda_i \phi(||x - x_i||), x \in R^d$$

Where

- x_i are the sample points, at which the value of y is known.
- x is the point at which the approximated value $\hat{y}$ will be calculated.
- ϕ is a univariate, normally continuous function in this case radial basis function
- $||.||$ is the Euclidean distance between two d-dimensional points.
- λ_i are the coefficients of the response function.

Radial basis functions are simply a class of functions. Generally they could be employed in linear or nonlinear models.

Assuming that the number of sample point is n, the system can be described as

$$y_1(x_1) = \lambda_1\phi(||x_1 - x_1||) + \lambda_2\phi(||x_1 - x_2||) + \lambda_3\phi(||x_1 - x_3||) + \cdots + \lambda_n\phi(||x_1 - x_n||)$$
$$y_2(x_2) = \lambda_1\phi(||x_2 - x_1||) + \lambda_2\phi(||x_2 - x_2||) + \lambda_3\phi(||x_2 - x_3||) + \cdots + \lambda_n\phi(||x_2 - x_n||)$$
$$y_3(x_3) = \lambda_1\phi(||x_3 - x_1||) + \lambda_2\phi(||x_3 - x_2||) + \lambda_3\phi(||x_3 - x_3||) + \cdots + \lambda_n\phi(||x_3 - x_n||)$$
$$.$$
$$.$$
$$.$$
$$y_n(x_n) = \lambda_1\phi(||x_n - x_1||) + \lambda_2\phi(||x_n - x_2||) + \lambda_3\phi(||x_n - x_3||) + \cdots + \lambda_n\phi(||x_n - x_n||)$$

where each equation represents the response of one sample : $y_n(x_n)$

$$Y = \lambda\Phi$$

$$Y = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \cdot \\ \cdot \\ \cdot \\ y_n \end{bmatrix} \qquad \lambda = \begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \lambda_3 \\ \cdot \\ \cdot \\ \cdot \\ \lambda_n \end{bmatrix}$$

$$\Phi = \begin{bmatrix} \Phi_{11} & \Phi_{12} & \dots & \Phi_{1n} \\ \Phi_{21} & \Phi_{22} & \dots & \Phi_{2n} \\ \cdot & & & \\ \cdot & & & \\ \cdot & & & \\ \Phi_{n1} & \Phi_{n2} & \dots & \Phi_{nn} \end{bmatrix}$$

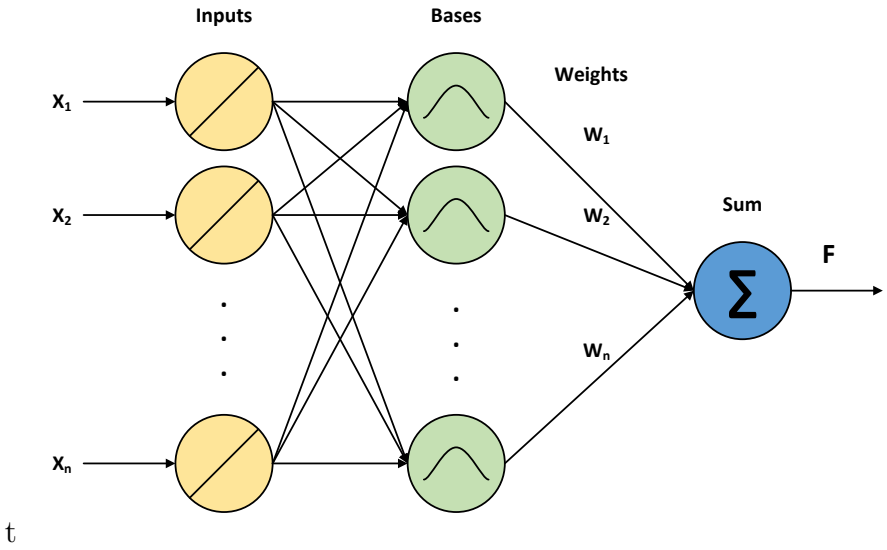


Figure 3.4: Radial Basis Function Network

3.4 Radial Basis Function Networks

Radial Basis Functions (RBFs) present a powerful solution to the problem of scattered data fitting, where some samples are given as data points and we want to approximate the response at new points [42].

The sampled data points represents an underlying behavior and we want to model this function. RBF networks can learn to approximate the underlying behavior using many Gaussian activation function. An RBF network is similar to a 2-layer neural network, see Fig. 3.4. RBF networks consists of three layers: input layer, hidden layer and output layer. In the input layer each neuron corresponds to a factor. Hidden layer has a number of neurons. Each neuron has a radial basis function. The output layer is a weighted sum of outputs from the hidden layer.

Our work incorporates RBF networks to approximate both $T(a_i, c_j, r_k, v_l)$ and $E(a_i, c_j, r_k, v_l)$.

3.5 Optimization

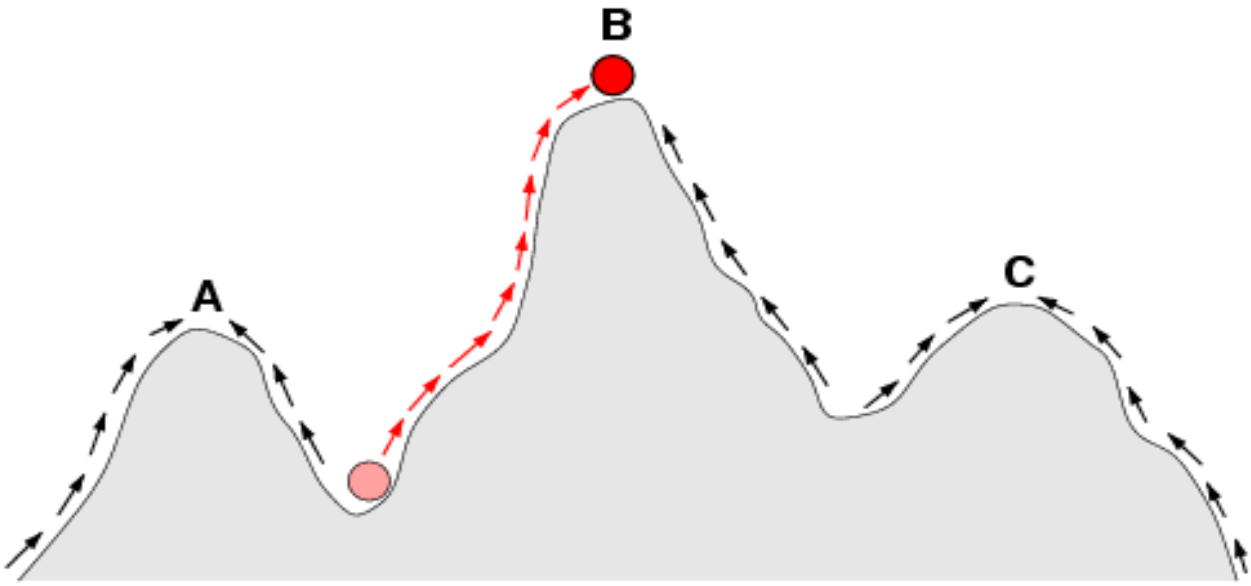
The goal of the optimization is to find the global best solution of a nonlinear function, which is a difficult task especially with the existence of multiple local optima. Generally, optimization applies mathematics to find the best set of variables to optimize an objective function. A problem with m variables $x_0, x_1, ..., x_{m-1}$ where $x_i \in \{0, ..., k-1\}$ will have a search space with k^m possible solutions. The fitness is a measure of how good the solution according to the objective function. The solution is evaluated by applying the objective function.

Figure 3.5a shows a search space for a single variable optimization problem, where the X-axis is the value of the single variable and the Y-axis is the fitness of this solution. As the number of variables increases the search space will have more dimensions see Figure 3.5b.

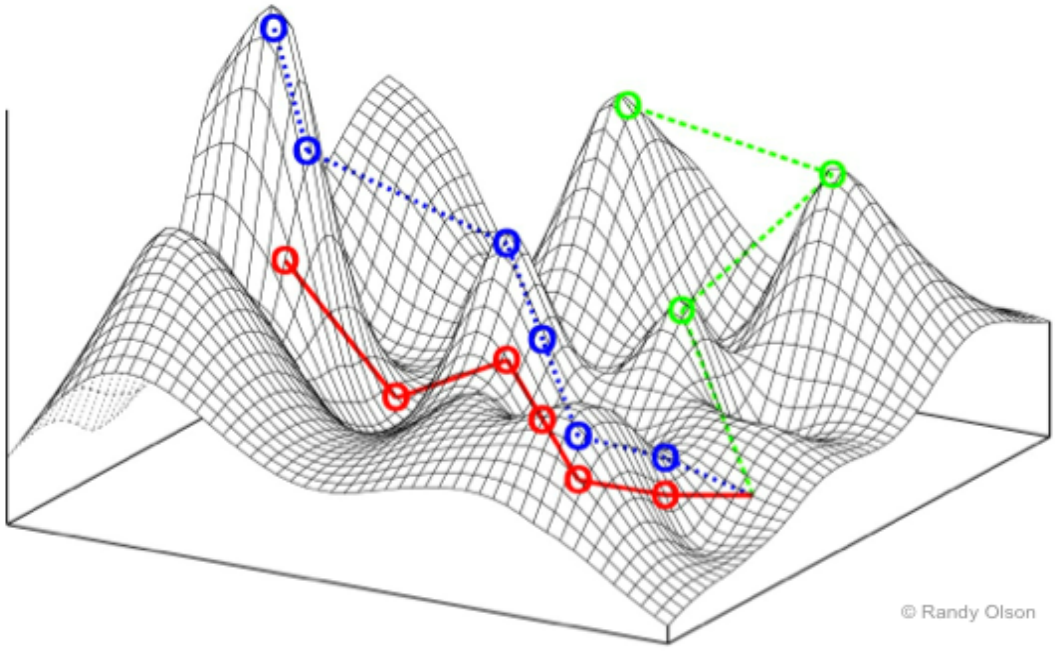
The optimization algorithm is a method for exploring the search space to find the highest or lowest point of the objective function. In the case of search spaces that are multidimensional, searching through and evaluating the huge number of variable combinations become infeasible.

Optimization algorithms can be classified into deterministic and non-deterministic approaches. Deterministic optimization includes all the algorithms that use a mathematical approach to find the optimal solution, this refers to mathematical programming. This approach follows a single path in the search space which starts at a sub-optimal solution and ends at the best solution. A non-deterministic algorithm is different from the deterministic in its ability to follow more than one path in the search space at random. Deterministic algorithms will always lead the same solution, for complex search space, this might be extremely time-consuming.

Heuristic algorithms are non-deterministic algorithms known to be effective in searching complex and unknown spaces. The computational cost of heuristic algorithms can be man-



(a) One dimensional search space for a maximization optimization. points A and C represent local optima. The ball show the movement from low fitness value high fitness. Borrowed from [2]



(b) Two dimensional search space for a maximization optimization. Borrowed from [2]

aged by trading the solution accuracy. Heuristic algorithms can be single solution based or population based algorithms. Single solution heuristics work on exploiting the current solution. On the other hand population based heuristics work on exploring the search space. Evolutionary algorithms are population based heuristics that were inspired by the mechanisms of biological evolution. The genetic algorithm [26] is a widely used heuristic algorithm that has a good balance of exploitation and exploration. This algorithm reflects the process of natural selection where the fittest individuals are selected for reproduction in order to produce offspring of the next generation. The next chapter discusses the details of the genetic algorithm.

Using the approximation functions generated from the previous stage, the optimization algorithm determines the configuration that minimizes the objective function f. Since the search space of this problem is large and complex and heuristic algorithms are known to be effective in searching such spaces, we use Genetic Algorithm to find the optimal configuration.

Chapter 4

Genetic Algorithm

Genetic Algorithm is a heuristic algorithm that uses the mechanics of natural selection and genetics to navigate search spaces. GA uses probabilistic transition not, deterministic rules. It is a population-based search technique that starts with a population of solutions. In GA, each solution is represented as a chromosome that has a fitness value. The fitness value is a measure of how good is a solution according to the objective(s). By applying operations such as selection, crossover, and mutation the solution that is fitter gets better chances to reproduce.

Selection is the process of choosing the mating candidates for the crossover. Selection is generally based on the idea that chromosomes with higher fitness values have a higher probability of contributing one or more children in the next generation. Two widely used implementations of fitness proportionate selection are the roulette wheel selection and the tournament selection.

After selection, crossover involves the exchange of genetic material between selected chromosomes (parents), to create new chromosomes (children). Various forms of this operator can be implemented such as one-point crossover, multi-point crossover, uniform crossover.

The mutation operator in its purest form makes small, random changes to a chromosome. It is used to maintain and introduce diversity in the population. The implementation of the mutation depends on the chromosome representation. Some of the generic implementations of mutation are random resetting mutation, swap mutation, scramble mutation, and inversion mutation.

The replacement policy determines which chromosomes are to be moved to the next generation. It is crucial as it should ensure that the fitter chromosomes have better chances of moving to the next generation. The age based replacement policy is based on the premise that chromosomes are allowed in the population for a finite number of generation, after that, replaced no matter how good its fitness is. On the other hand, a fitness based replacement is based on the idea that children tend to replace the least fit individuals in the population. The selection of the individuals to be replaced may be done using one of the selection policies.

Some GAs employ elitism. It means the current fittest member(s) of the population is always propagated to the next generation. Elitism ensures that the best chromosome of the current population would not be replaced.

The termination criteria determine when the GA ends. Generally, GA progresses very fast with better solutions in the first generations, but this tends to saturate in the later stages where GA converges. GA can be terminated by one of the following events: when there has been no improvement in the population for a specific number of iterations, when it reaches the maximum number of generations or when the objective function value has reached a pre-defined value. The general process of GA is shown in Figure.4.1 and a pseudo-code of the proposed GA based optimization is shown in Algorithm 1

This work considered two approaches to reconfigure the multi-core system. The first approach is a chip-wide configuration where a uniform configuration is applied across all cores (homogeneous configuration). The other reconfiguration approach is the per-core config-

Algorithm 1 Pseudo-code of the proposed GA based optimization

```
1: //Initialize generation 1
2: generation = 1
3: Population = Initialize Population
4: //Termination criteria
5: while generation ≤ max_generation do
6:    EVALUATE(Population)
7:    Children = {}
8:    while size(Children) < size(Population) do
9:       if rand() ≤ CrossoverRate then
10:         {Patent1, Parent2} = SELECT (Population)
11:         {child1, child2} = CROSSOVER(Patent1, Parent2)
12:      else
13:         child1 = Parent1
14:         child2 = Parent2
15:      end if
16:      {child1, child2} = MUTATE(child1, child2)
17:      Children = ADD(child1, child2)
18:   end while
19:   EVALUATE(Children)
20:   Population = TOURNAMENT SELECTION(Population + Children)
21: end while
```

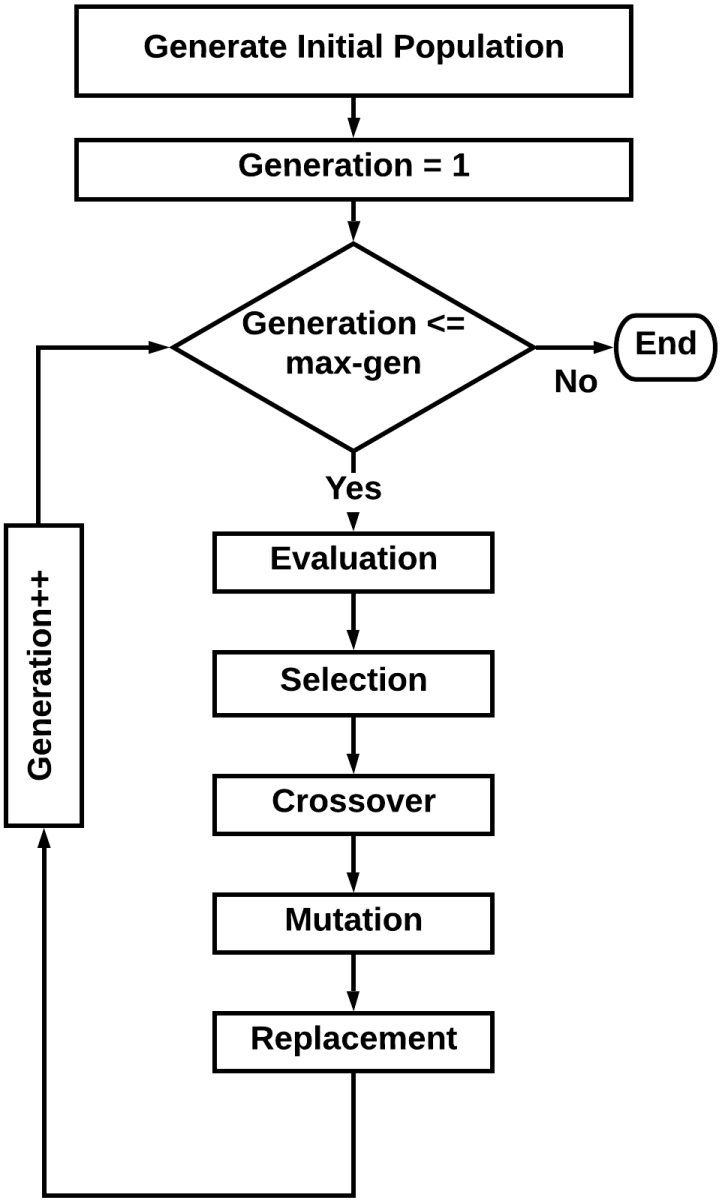


Figure 4.1: Flow chart showing the evolution process of genetic algorithm.

uration where each core has a different set of resources. Two adaptations of GA were implemented, one for each reconfiguration approach.

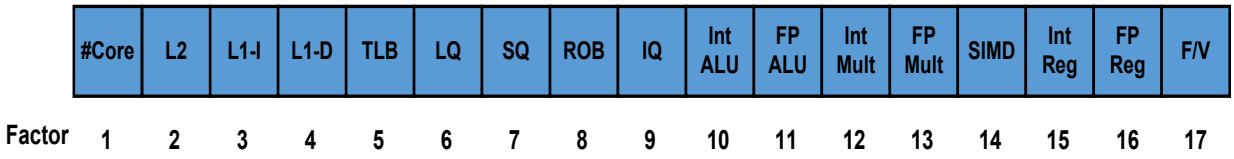


Figure 4.2: Chip-wide configuration chromosome representation.

4.1 GA for Chip-wide Configuration

In a chip-wide scaling, the factors are scaled uniformly in all cores. One configuration is applied to all the active cores in the system.

4.1.1 Chromosome Representation

Each chromosome represents a configuration for the multi-core system. In a chip-wide configuration, all the active core have the same resource configuration and the same V/F level. The chromosome is represented as an array of factors that determine the configuration of the system see Figure4.2. The first gene (Factor[1]) of the chromosome represents the number of active cores. The other genes (Factor[2] to Factor[16]) represent the resource configuration of all of the active cores and one gene (Factor[17]) represent the chip-wide V/F level.

4.1.2 Population Initialization

The population is a subset of solutions in the search space. It consists of a set of chromosomes. The population should be diverse to avoid premature convergence. The population is usually defined as a two dimensional array of [chromosome size][size population], see Figure4.3 . The initial population of GA is created randomly.

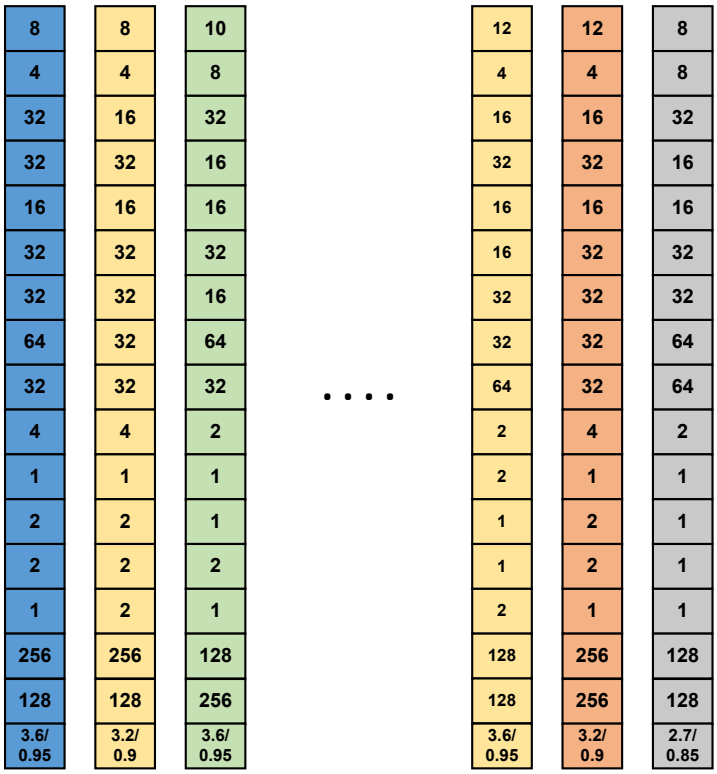


Figure 4.3: Example of Initial Population

4.1.3 Selection

To apply the crossover, two chromosomes (parents) have to be selected from the population. In this implementation, a tournament selection policy is used. A set of n (tournament selection size) chromosomes is chosen randomly from the population. From the n randomly selected chromosome, the one with the highest fitness is selected as the first parent. This process is repeated to select the second parent. The selected parents produce two children through the crossover.

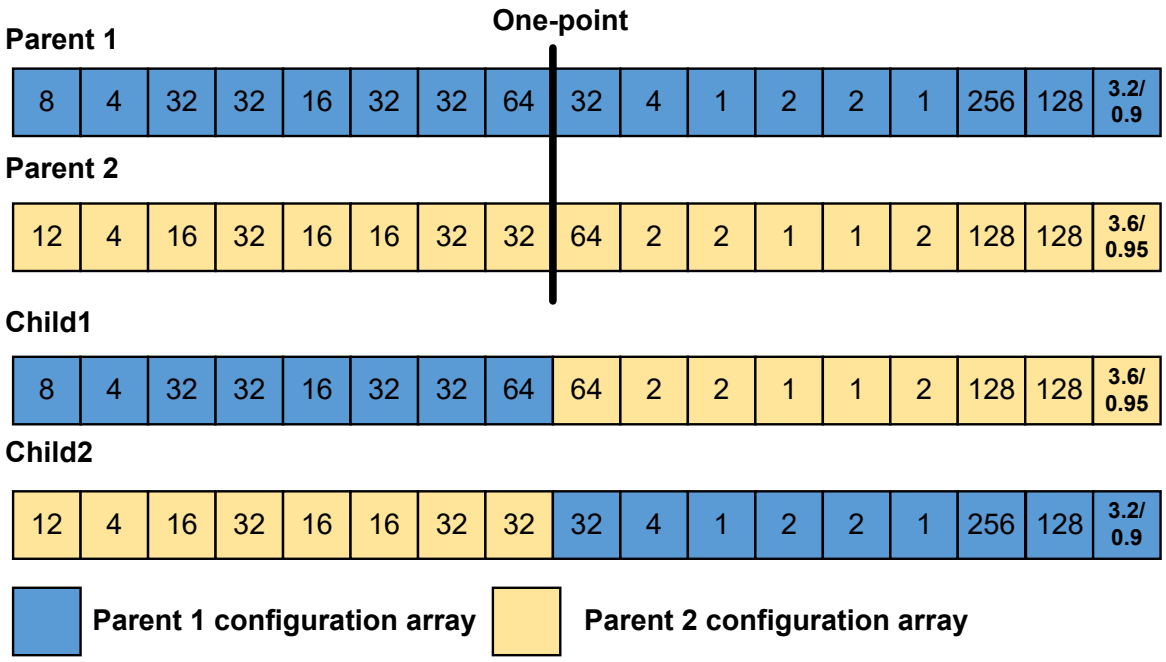


Figure 4.4: Chip-wide configuration one-point crossover operator.

4.1.4 Crossover

The crossover operator is performed on the selected parents to produce the new solutions that will be part of the next generation. The objective of the crossover operator is to exploit the search space. Crossover works by mixing the genes of the parents to create new solutions. For this operator to work, the resulting chromosome should represent a valid configuration. The crossover operator can be implemented in various ways. In this work, a one-point crossover is applied to the selected parents according to a crossover rate to generate two children, see Fig.4.4. A random point in the chromosome is selected, then the first part is copied from parent 1 to child 1 while the second part is copied from parent 2. Similarly, the first part is copied from parent 2 to child 2 while the second part is copied from parent 1.

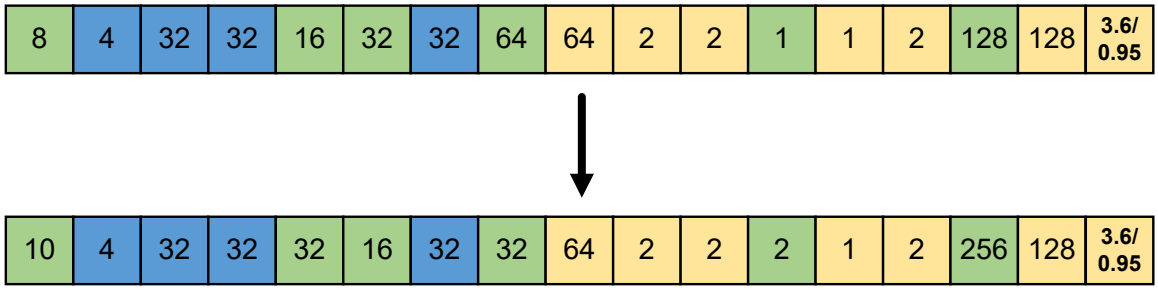


Figure 4.5: Chip-wide configuration mutation operator.

4.1.5 Mutation

The main objective of the mutation operator is to explore new areas in the search space. It introduces some diversity into the population to avoid the algorithm to be stuck in local-optima. The mutation operator makes a small random change to a chromosome. After a child is produced from the crossover operator, mutation is applied with a very low probability on each factor of the children chromosome. An example is shown in Fig. 4.5, the factors that were mutated are the number of active cores, TLB size, LQ entries, ROB size, the number of integer multipliers and the integer register file size by randomly resetting their values.

4.1.6 Replacement and Termination Criteria

This reproduction process is repeated until the number of children chromosomes is equal to the population size. Then tournament selections is used on the combined current population and children population to generate the new population for the next generation.

The evolution process is repeated until the maximum number of generations is reached and the solution with the best fitness is returned.

Factor/Core	1	2	3	4	...	n
1	L2	L2	L2	L2	...	L2
2	L1-I	L1-I	L1-I	L1-I	...	L1-I
3	L1-D	L1-D	L1-D	L1-D	...	L1-D
4	TLB	TLB	TLB	TLB	...	TLB
5	LQ	LQ	LQ	LQ	...	LQ
6	SQ	SQ	SQ	SQ	...	SQ
7	ROB	ROB	ROB	ROB	...	ROB
8	IQ	IQ	IQ	IQ	...	IQ
9	Int ALU	Int ALU	Int ALU	Int ALU	...	Int ALU
10	FP ALU	FP ALU	FP ALU	FP ALU	...	FP ALU
11	Int Mult	Int Mult	Int Mult	Int Mult	...	Int Mult
12	FP Mult	FP Mult	FP Mult	FP Mult	...	FP Mult
13	SIMD	SIMD	SIMD	SIMD	...	SIMD
14	Int Reg	Int Reg	Int Reg	Int Reg	...	Int Reg
15	FP Reg	FP Reg	FP Reg	FP Reg	...	FP Reg
16	F/V	F/V	F/V	F/V	...	F/V

Figure 4.6: Per-core Configuration Chromosome Representation.

4.2 GA for Per-core configuration

In a Per-core scaling each core has a different configuration.

4.2.1 Chromosome Representation

Each chromosome represents a configuration for the multi-core system. In a per-core configuration, each core have its own resource configuration and the V/F level. The chromosome is represented as a m by n matrix where m is the number of configurable factors for a single core and n is the number of cores. Each column of this matrix represents the configuration of the corresponding core, see Fig.4.6.

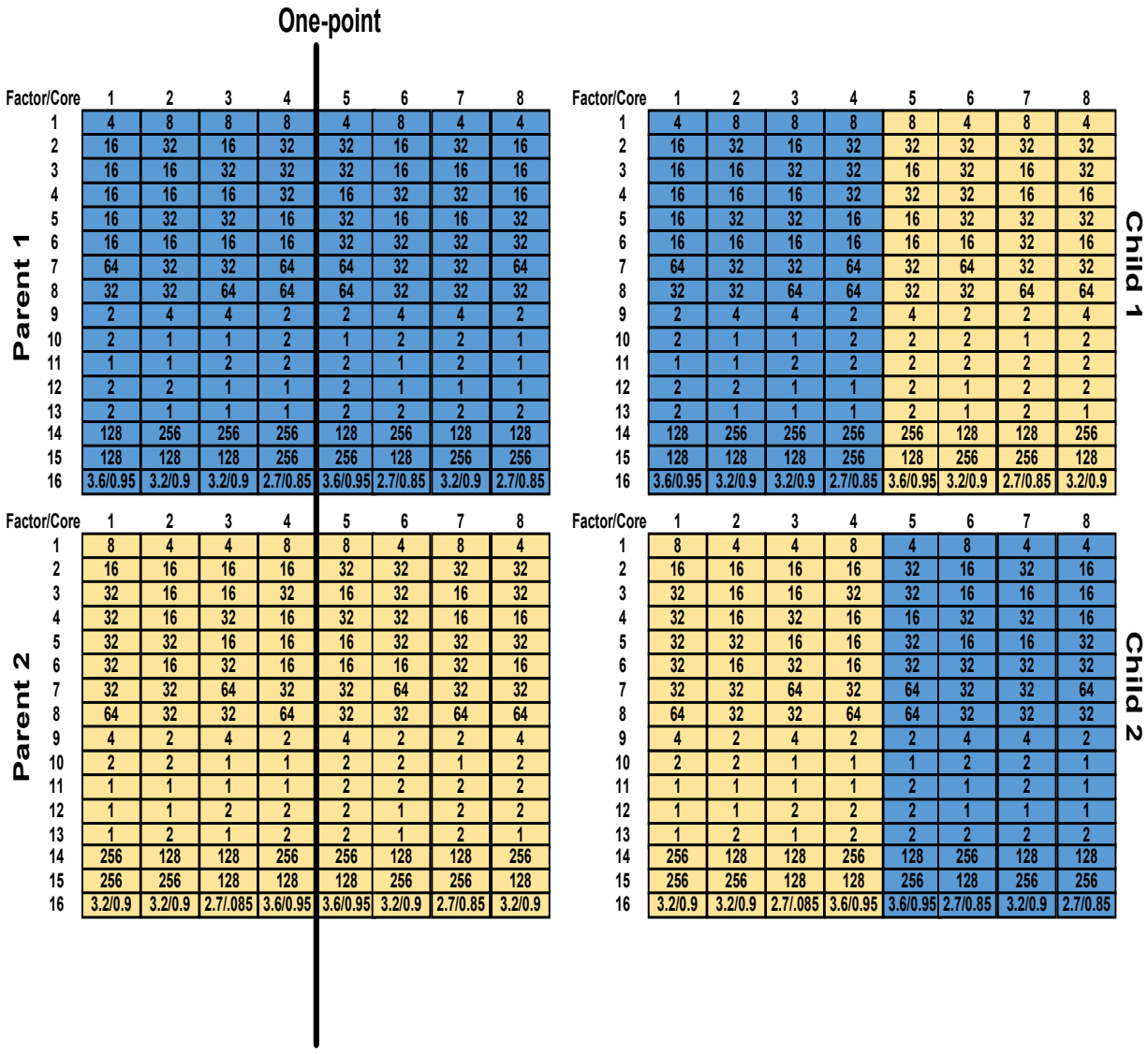


Figure 4.7: Per-core configuration Crossover Operator

4.2.2 Crossover

The selection step is implemented in a similar manner to the chip wide configuration. After the two parents are selected a one-point crossover is applied, see Fig.4.7. A random point is selected, then the first child is created by coping the first set of columns from parent 1 and the second set of columns from parent 2. Similarly, the first set of columns is copied from parent 2 to child 2 while the second set of columns is copied from parent 1.

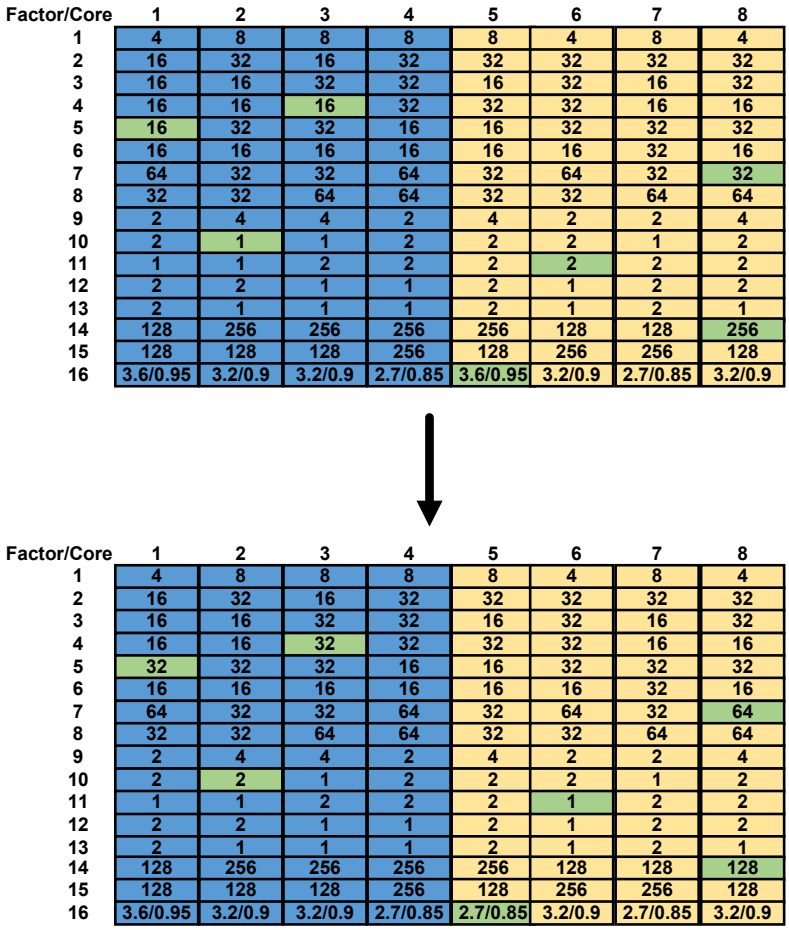


Figure 4.8: Per-core Configuration Mutation Operator.

4.2.3 Mutation

The mutation operator makes a small random change to a chromosome. After a child is produced from the crossover operator, mutation is applied with a very low probability on each factor of the children chromosome. An example is shown in Fig. 4.8, the factors that were mutated are the LQ entries of core1, number of FP ALUs of core2, TLB size of core3, V/F level of core5, number of Int multipliers of core6, ROB size, and Int register file size of core8.

4.3 Elitism

Elitism is implemented by copying the best solution found so far into the next generation. This will improve the algorithm performance by ensuring that no time is wasted re-discovering previously discarded solutions. Without elitism the crossover and mutation operators will likely change the best solution and it will not be present in the new generation.

4.4 Parameters

There are a number of parameters that can affect the performance and execution time of GA. First, the crossover rate which determine how often the crossover operator will be performed. Crossover is made in a way that new chromosomes, are created by mixing genetic material from good chromosomes and the new chromosomes could be better than the old ones. However, it is good to leave some chromosomes from the current population survive to next generation with out change.

Another parameter is the mutation rate which determine how often parts of chromosome will be mutated. If there is no mutation, offspring will be copied without any change to the next generation. If mutation is performed, part of chromosome is changed. Although, mutation is applied to prevent local-optima, it should not occur very often, because then the GA will change to a random search.

Population size is the number of chromosomes in one generation. If there are too few chromosomes, only a small part of search space will be explored. On the other hand, it is not useful to increase population size, because it will slow the GA. It is a trade off between speed and accuracy.

Chapter 5

Evaluation Methodology

The proposed scaling technique require a lot of hardware modifications to support the re-configuration process. Every scalable resource in the micro-architecture needs to have the ability to be scaled. This can be implement by physical gating mechanisms which include adding sleep transistors to power down each part of the scaled resources. Moreover, a logical correctness mechanisms should be implemented to ensure proper operation when resources are scaled. Due to the required modification to the hardware, evaluating the proposed optimization on a real system is not applicable and simulation is more suitable.

In order to evaluate the proposed optimization framework, a full-system simulator, Gem5 [7] was used to simulate a 12 core multi-core system and model the performance function. Mcpat [33], power modeling framework ,was used to model the power consumption (both static and dynamic)

5.1 Gem5

Gem5 is an open source simulation tool that combines features from the M5 Simulator [15] and the General Execution-driven Multiprocessor Simulator (GEMS) Toolset [16]. It can be used to simulate most commercial ISAs, including ARM, ALPHA and x86, with full system features. Also, it incorporates Ruby, which can be used to simulate the memory hierarchy including cache coherence protocols and Garnet which models the interconnect to build a customized NoC topologies.

Gem5 has an object oriented design. The main object is the SimObject. All the major components of gem5 are SimObjects, including cores, caches, NoCs, pipelines, and memory controllers. Every SimObject is represented by two classes, one in Python and one in C++. The C++ class defines the performance critical functions of the modeled component, such as the state, behavior, and performance-critical simulation model. On the other hand, the Python class contains the parameters and specifications.

Gem5 offer two simulation modes. The first mode is the System call Emulation (SE). It can run single applications and uses a simplified address translation model. In this mode there is no scheduling and system calls are emulated through the host operating system. SE simulation is very fast, usually used for testing the basic functionality. The second simulation mode is the Full System (FS). This mode simulates a full system model including caches, interrupts, exceptions, fault handlers, and an operating system.

Gem5 provides a number of CPU models each simulate different level of details in memory access and instruction execution. The simplest model is the AtomicSimple which is a single IPC CPU. TimingSimple model has a more detailed memory model, it uses functions to send cache requests and handle responses. .The InOrder model implements more detail than the simple models, using abstractions for pipeline components, such as ALU, FPU, and Branch Predictor. O3CPU models an Out-of-Order CPU Including the five stages of the pipeline.

Table 5.1: Parameters of the Simulated System

Parameter	Values
Cores	12 OoO
fetch/issue/retire	4/4/4
Technology	32nm
Coherence Protocol	MOESI
NoC topology	Crossbar switch
V/F	0.9V/3.2GHz
L2 cache	8 GB, 16-way
L1-D, L1-I cache	32KB, 4-way
LQ, SQ, TLB	32 entries
Int ALU/ FP ALU	4/2 units
Int multiplier, FP multiplier, SIMD	2 units
Int Register file, FP Register file	256 entries
ROB, IQ	64 entries

There are two memory system models, the classic memory system and the Ruby memory system. The classic model is a fast and easily configurable memory system, while the Ruby model is used to accurately simulate cache memory systems and a variety of cache coherence protocols including MSI, MESI, MOESI, AMDs hammer.

With the wide variety of capabilities and components in gem5, it provides between speed, flexibility, and accuracy, where the simulation speed increases with the use of less detailed models, and the accuracy increases with the use of more detailed models.

To evaluate the proposed online system, we used the full system mode and ruby memory system to simulate a 12 core processor, see Table 5.1 for more detail.

5.2 McPAT

Multi-core Power, Area, and Timing (McPAT) is an integrated power, area, and timing modeling framework for multi-threaded and multi-core processors. McPAT includes models for the basic components of a multi-core, including in-order and out-of-order processor cores, networks-on-chip, shared caches, integrated memory controllers, and multiple-domain clocking. At the circuit and technology levels, McPAT supports area, dynamic, short-circuit, and leakage power modeling, for CMOS, SOI, and double-gate transistors. McPAT has a XML interface to make it compatible with different performance simulators. The XML configuration file specifies the level of configuration details. It also provides default values of the architectural parameters. The xml file has to contain all components that are considered for power consumption.

For each component McPAT determine:

- Area (mm^2): The area of the component.
- Peak Dynamic (W): Power of maximum switching activity.
- Subthreshold Leakage (W): Even though the transistor is logically turned OFF, there is a non-zero leakage current through the channel.
- Subthreshold Leakage with power gating (W): The subthreshold leakage with a technique to reduce the power consumption.
- Gate Leakage (W): Is the current leaking through the gate terminal, and varies greatly with the state of the device.
- Runtime Dynamic (W): The power for charging and discharging the capacitor when the circuit switches state.

Table 5.2: Parsec Workloads and the Input Set

Program	Application Domain	Problem Size
Blackscholes	Finalcial Analysis	16,384 options
Bodytrack	Computer Vision	2 frames, 2,000 particles
Canneal	Engineering	200,000 elements
Dedup	Enterprise Storage	31 MB data
Facesim	Animation	1 frame, 372,126 tetrahedra
Ferret	Similarity Search	64 queries, 13,787 images
Fluidanimate	Animation	5 frames, 100,000 particles
Freqmine	Data Mining	500,000 transactions
Streamcluster	Data Mining	8,192 points per block, 1 block
Swaptions	Financial Analysis	32 swaptions, 10,000 simulations
x264	Media Processing	32 frames, 640 x 360 pixels

The only coding needed was creating a script to parse the information from gem5 to McPAT. The script reads the parameters and statistics from gem5 configuration and result files, and generate a McPAT configuration file from the McPAT template file.

5.3 Benchmarks

PARSEC (The Princeton Application Repository for Shared-Memory Computers)[6] is one of the popular benchmark suites for parallel programming. It provides a variety of applications selected from several application domains to cover different areas in parallel programming. In this work 8 out of the 13 application were tested (see Table 5.2)

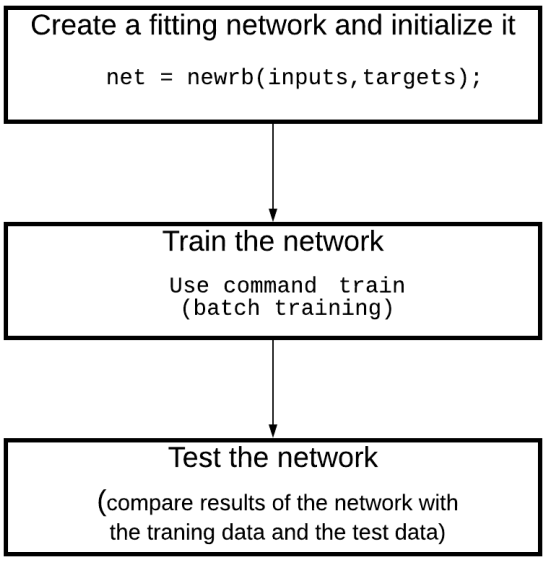


Figure 5.1: Basic steps for creating a RBFN

5.4 Matlab

MATrix LABoratory (Matlab) is a multi-paradigm numerical computing environment and proprietary programming language developed by MathWorks. MATLAB is widely used in all areas of applied mathematics, research, and in the industry. It is a great tool for solving algebraic and differential equations and for numerical integration. It is also, a programming language, and is one of the easiest programming languages for writing mathematical programs. MATLAB has some tool boxes useful for signal processing, image processing, optimization.

Deep Learning Toolbox [54] (formerly Neural Network Toolbox) is a framework for designing and implementing deep neural networks. It provides different neural networks to perform classification and regression on data. It has a graphical interface that is used to visualize activations, edit network architectures, and monitor training progress. In this work this toolbox was used to build the RBFN for the response functions fitting.

RBFN can be designed using two different functions. The first function is newrbe(). This

function creates radial basis networks with as many neurons as there are inputs in the training data. The second function is newrb().This function iteratively creates a radial basis network one neuron at a time. Neurons are added to the network until the sum-squared error falls beneath an error goal or a maximum number of neurons has been reached. Typically, Newrb result in fewer neurons than newrbe

After creating the network, network training can be done using one of the training functions. Some of the most widely used functions are

- traingd: a network training function that updates weight and bias values according to gradient descent.
- traingdm: a network training function that updates weight and bias values according to gradient descent with momentum. It is generally faster than traingd
- Traingdx: a network training function that updates weight and bias values according to gradient descent momentum and an adaptive learning rate. It has faster training time than traingd
- trainrp: is a network training function that updates weight and bias values according to the resilient backpropagation algorithm (Rprop). It has a fast convergence and minimal storage requirements.
- Trainlm: is a network training function that updates weight and bias values according to Levenberg-Marquardt optimization. It is the fastest backpropagation algorithm but it does require more memory than other algorithms. This function was used in the RBFN

There are several parameters associated with training process such as learning rate, error goal, and epochs which specifies the number of iterations.

Table 5.3: GA Parameters

Parameter	Values
Population size	32
Crossover Rate	0.8
Mutation Rate	0.5
Max Generations	25

Once the RBFN is trained and tested the response approximation function is created using genFunction() which generates a complete stand-alone function for simulating a neural network including all settings, weight and bias values, module functions, and calculations in one file. The resulted function is called by the GA evaluation stage.

5.5 GA

The Genetic Algorithm was written in C++. A number of GA parameter values were tested to find the best GA parameters (see Table 5.3).

5.6 Summary

To put it all together, the proposed system consists of three stages: sampling stage, surface fitting stage, and optimization stage. This system was evaluated on a simulated multi-core. Gem5 simulator was used to simulate the architecture and give the performance readings. McPAT was used for the energy consumption modeling. During the sampling, an application runs for several sampling intervals, and each interval has a dierent combination of factors. The applications that were tested are from PARSEC benchmark suite. A fractional factorial design was used to define the sampling points. Once both the performance and energy consumption responses are measured, a performance and power approximation functions are

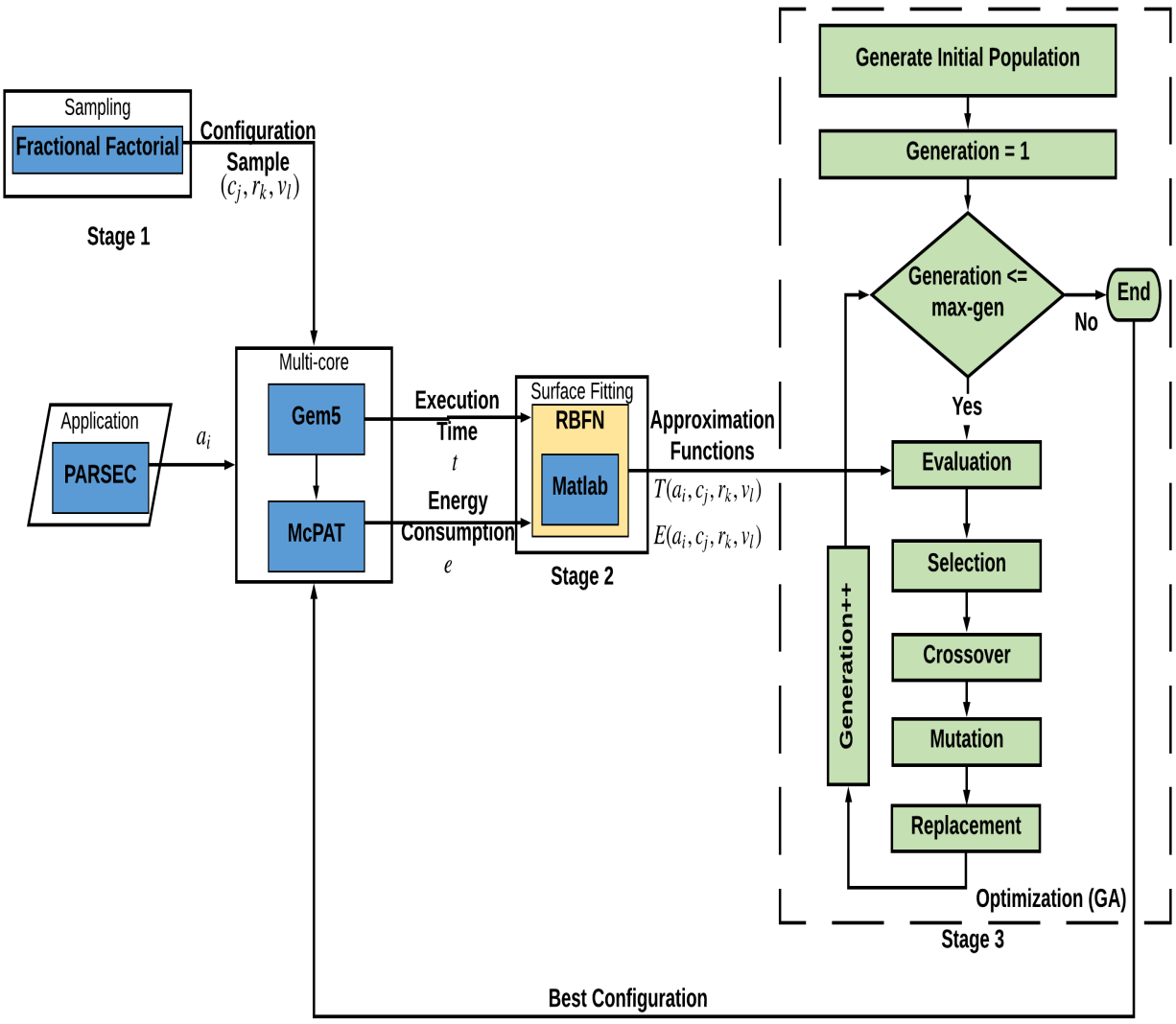


Figure 5.2: Evaluation methodology showing different simulators and applications used to evaluate the proposed system

created using RBFN on the sampled data points. Finally, using the approximation functions, the GA determines the combination of factors that optimizes the overall performance and energy consumption. The system operates with this conguration for the remainder execution time. Figure 5.2 shows the evaluation methodology.

Chapter 6

Runtime System Overhead

The overhead of proposed system is distributed on the different stages as follow:

6.1 Configuration Sampling

The configuration sampling is done off-line due to long cycle-level simulation time. For the sampling, 144 sample configurations were applied for a defined execution interval of the benchmarks. The number of sampled configuration is constrained by the simulation time limitation but it can be more when using real hardware.

To change from one configuration to the other we gradually turn on more resources to go from small scale to larger scale to avoid the cost of scaling down resources (i.e., flushing cache and core pipelines) as much as possible.

6.2 Core Scaling

Power-gating techniques proposed to reduce leakage power and to implement microprocessor deep sleep states, such as C6. Intel Core i7 microprocessors implement power-gating transistors to shut off idle cores [28]. To change the number of cores, the runtime system informs the thread scheduler so that it can change the number of running software threads. We assume it takes a few μ seconds to switch threads between cores [51].

6.3 Memory Scaling

Memory components can be either set-associative memories such as L2, L1-I, L1-D, and BTB or fully-associative memories such as ROB, TLB, IQ, and LSQ. Dynamic memory scaling techniques have been widely used in commercial processors to reduce leakage power consumption at runtime [74]. For set-associative memories it is possible to shut down a subset of arrays that compose a set. This reduces the leakage power consumption, but it does not affect the dynamic power. This is because it does not reduce the switching capacitance of accessed arrays

For fully-associative usually are designed using content-addressable memory (CAM) and some combinational circuits. It is possible to shut down a subset of total entries to reduce leakage power consumption without impacting the critical path delay [44]. Large fully-associative memory component can have multiple CAM arrays that are connected in a hierarchical way. In this case, disabling a subset of CAM arrays can also reduce dynamic power consumption

Caches are designed in a way that their size can be easily adapted for different architectures. The cache structure is typically composed of multiple arrays and each array is equipped with

a local power-gating device that can turn it on/off independently [46]. In the case of cache scale down, the cache controller writes back all the dirty cache lines to the main memory and then tern off the cache arrays that were scaled the cache size. This is done by scanning all the cache lines in the cache arrays that are to be turned off.

- For cache lines with modified state (M), the cache controller writes back the data to the main memory and updates the line state to invalid (I).
- For cache lines with exclusive (E) state, the cache controller just updates the line state to (I).
- For cache lines with shared (S) state, the cache controller sends an invalidation request to all the sharing processes and waits to get invalidation acknowledgements. Then, the the data is written back to the main memory and the lines are updated to (I).

6.4 Micro-architecture reconfiguration

Power gating can be implemented at a fine grain [11], where each standard cell has a sleep transistor, or at a coarse grain, where clusters of gates in the same voltage domain have an array of sleep transistors distributed in a ring or grid style [50]. Fine-grained sleep transistor usually lead to higher area overhead.

To scale down micro-architecture resources in a core, First, wait for any unresolved cache misses to be serviced. Second, wait for all the instructions existing in the IQ and ROB to be executed. Then we shut down parts of the resources of the cores using the power-gating technique.

For scaling the V/F, We assume the availability of on-die voltage regulation to enable fast chip wide DVFS with a range between 2.7 GHz at 0.85 V to 3.6 GHz at 0.95 V.

6.5 Optimization

The computational cost of the optimization process is composed of two parts: surface fitting model and GA. For surface fitting model we used RBF. Its overhead can be substantially reduced by developing optimized RBF code.

The GA has a complexity of

$$O\Big(P * G * O(Fit) * \big(CR * O(C) + MR * O(M)\big)\Big)$$

,where P is the population size, G is the number of generations, CR is the crossover rate and MR is the mutation rate, $O(Fit)$ is the complexity of the fitness function, $O(C)$ is the complexity of the crossover operator and $O(M)$ is the complexity of the mutation operator.

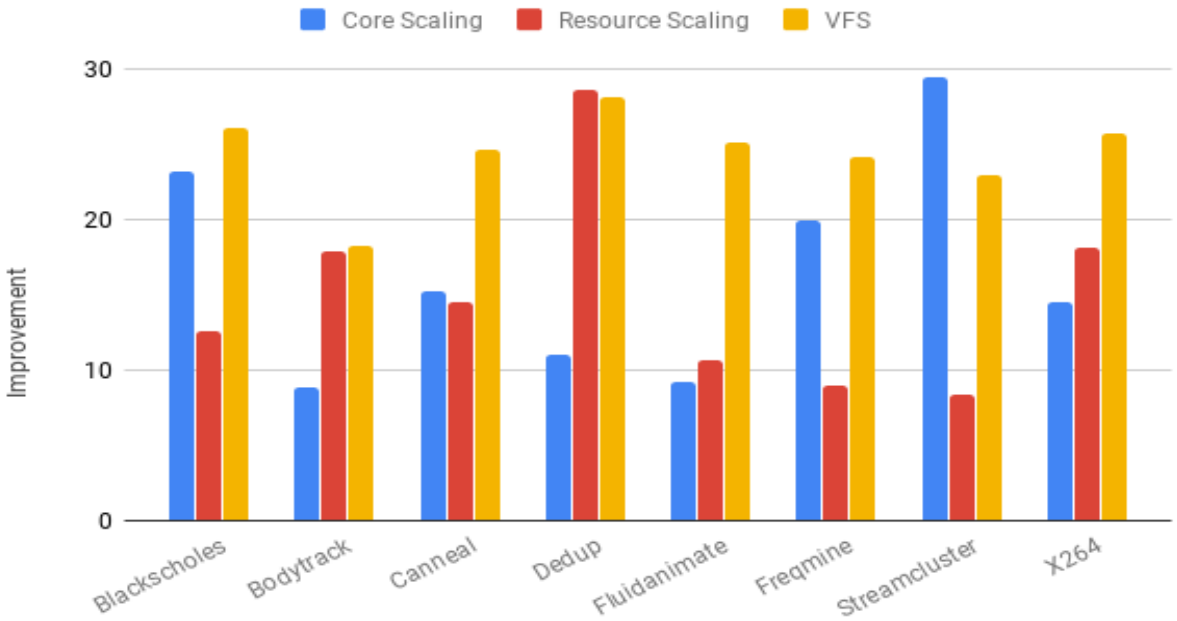
Chapter 7

Experimental Results

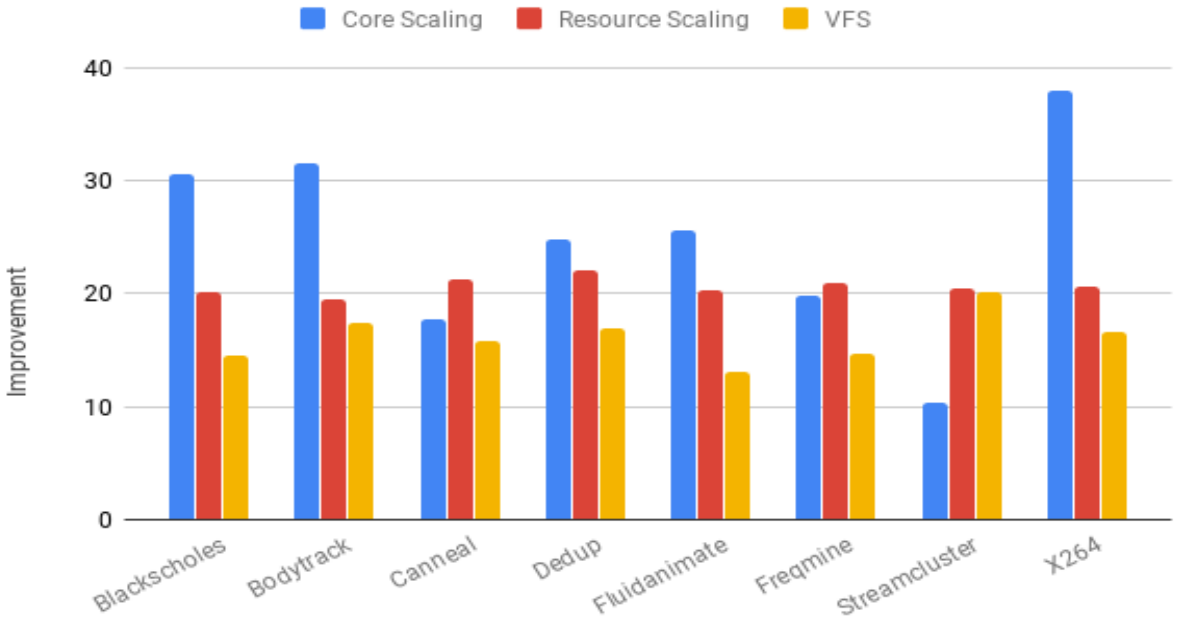
7.1 Isolated Scaling

To evaluate our approach, we first tested each of the alternative power management schemes separately. Our results showed that each application require a distinct processor configuration to maximize performance and a distinct configuration to minimize energy consumption. The amount of parallelism in an application plays a key role in determining the best configuration. For some applications, using more cores leads to higher performance than using more resources per core. For other applications, using more resources leads to leads to higher performance than using more cores. Secondly, different applications exhibit different performance and power trade-offs.

Figure.7.1 shows the average improvement in execution time and energy consumption of the different benchmarks using DVFS, core scaling and resource scaling. To test each technique in isolation we scaled only the factors that correspond to that technique and fixed the other factors, then took the average of improvements.



(a) Improvement in Execution time



(b) Improvement in Energy consumption

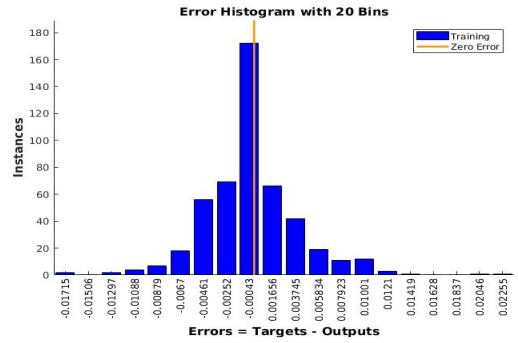
Figure 7.1: Comparing different power management techniques in terms of (a) improvement in Execution time and (b) improvement in Energy consumption

The results show a variation of cases. Our first observation is that DVFS gives the best improvement in execution time for most of the applications but it leads to the least improvement in energy consumption. Core scaling beats resource scaling for Blackscholes, Canneal, Freqmine, and Streamcluster in terms of improving the execution time. However, resource scaling outperforms it in terms of energy consumption for the same set of benchmarks except for Blackscholes. On the other hand, for Bodytrack, Dedup, Fluidanimate and x264, resource scaling results in higher improvement in execution time but less saving in energy consumption than core scaling. In the case of Blackscholes core scaling beats resource scaling in both execution time and energy consumption. It can be concluded that there is no one power management method that works best for all benchmarks for both performance and energy consumption.

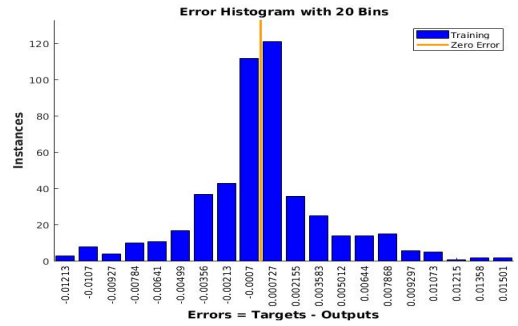
Using DVFS alone, energy consumption can be reduced by 16% and performance can be improved by 24% on average. On the other hand, core scaling reduces energy consumption by 25% on average and improves performance by 16%. In addition, resource scaling reduces energy consumption by 21% and performance improves by 15% on average.

7.2 Response Surface Model

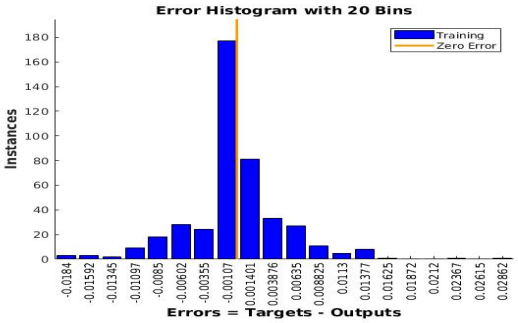
The error of the RBNF for approximating the execution time and energy consumption for each application is provided in Figure 7.2. On average the error is less than 1% in both directions, meaning that responses are both overestimated and underestimated.



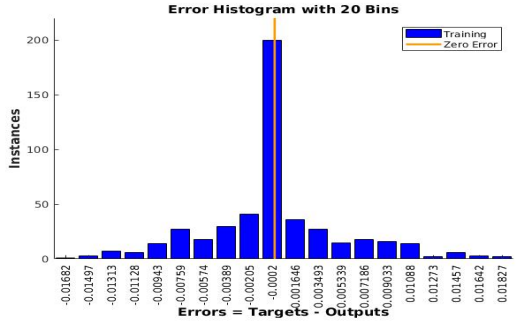
(a) Blackscholes



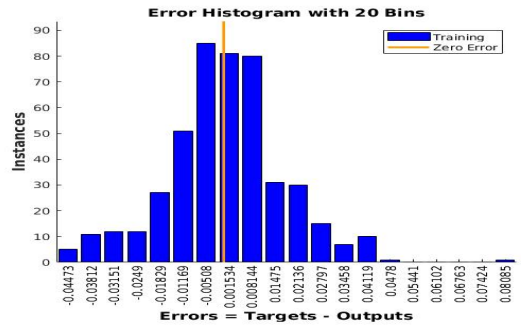
(b) Bodytrack



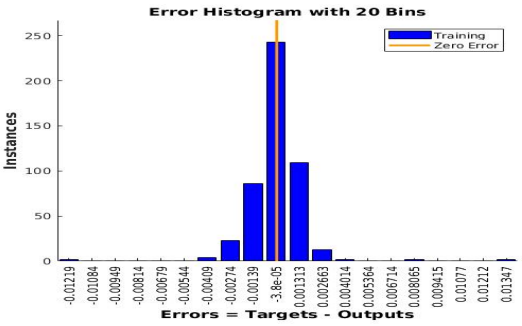
(c) Canneal



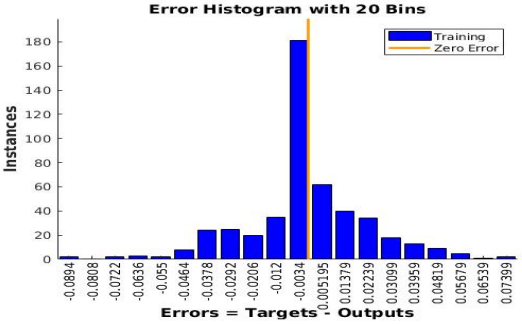
(d) Dedup



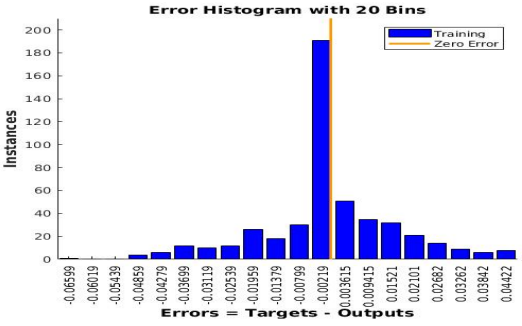
(e) Fluidanimate



(f) Freqmine



(g) Steamcluster



(h) x264

Figure 7.2: RBFN accuracy measured as percent error between the predicted and real values of response function

Table 7.1: Baseline Configuration

Level	Factors	Values
Architecture	Number of cores	8
Cache	L2 cache size (MB)	8
	L1-I cache size (KB)	32
	L1-D cache size (KB)	32
Core	TLB size	32
	LQ entries	32
	SQ entries	32
	ROB entries	64
	IQ entries	64
	Int ALU	4
	FP ALU	2
	Int Mult	2
	FP Multi	2
	SIMD	2
	Int Reg	256
	FP Reg	256
Voltage, Frequency	F/V (GHz/ V)	3.2/0.9

7.3 Chip-wide Configuration

The main benefit of our approach is that it allows different power management techniques to be combined to optimize both performance and energy consumption. We implemented the GA under three different objectives; GA with the objective of optimizing execution time only (GA perf), GA with the objective of optimizing energy consumption only(GA energy) and GA optimizing both (GA Both).

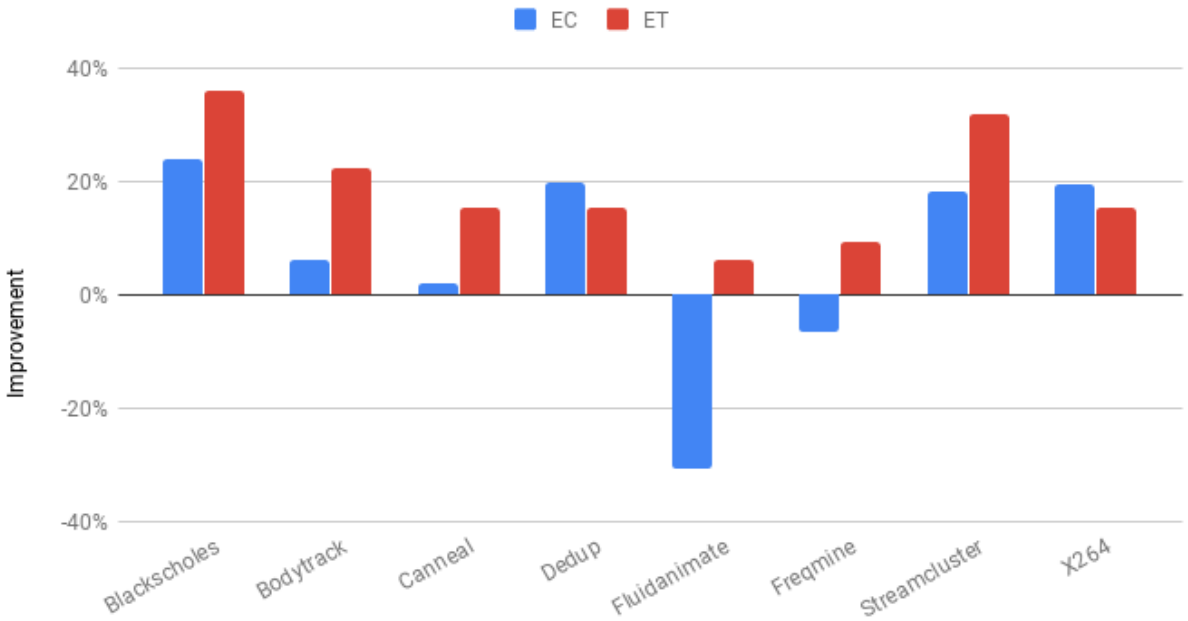
Figure.7.3 compares the improvement in energy consumption and execution time of the benchmarks using GA performance and GA energy. The improvement is based on comparing the best configurations of the final generation of GA to the baseline configuration shown in table 7.1.

For most of the benchmarks optimizing the execution time only leads to a reduction in energy consumption. In those cases the energy is saved because of a faster execution time. On the other hand, some benchmarks (i.e., Fluidanimate, and Freqmine) exhibit an increase in energy consumption of at most 30% when optimizing for execution time.

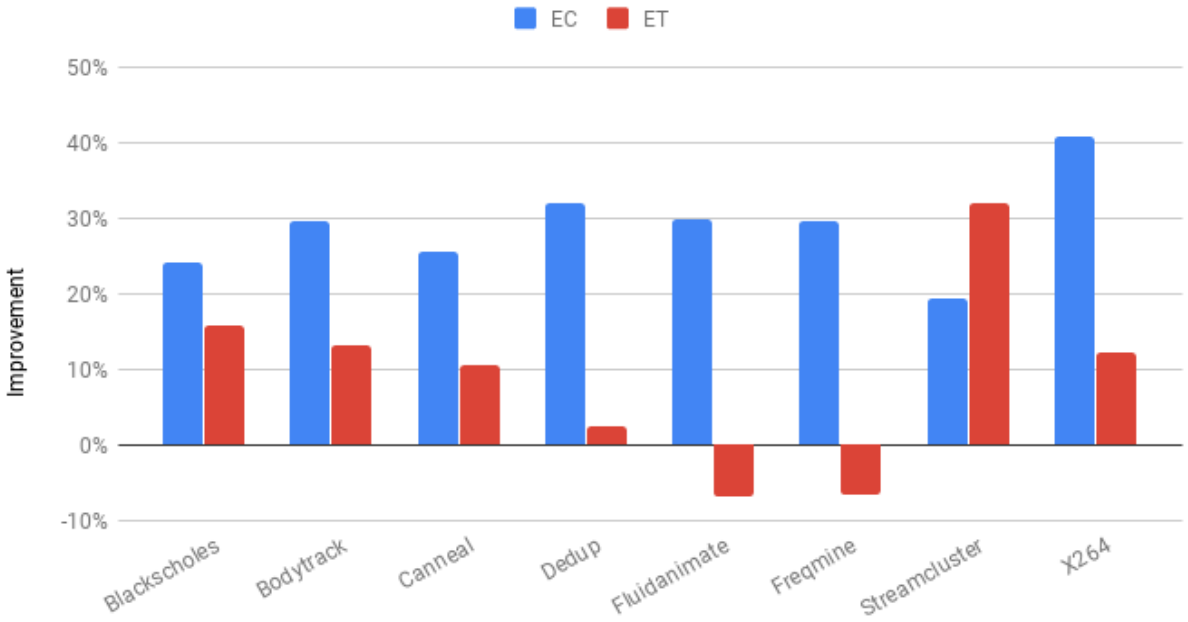
Similarly, optimizing energy consumption in most cases leads to an execution time improvement due to the faster execution time. Nevertheless, GA EC cannot reach the best solution in terms of performance because it aims to find the configuration with the least energy consumed without considering its execution time. In some cases (i.e., Streamcluster), the configuration with the least energy consumption happens to be the one with the shortest execution time as well.

Some applications have higher potential for improvement, particularly because they spend most of their execution time with few running threads. On the other hand, other applications are good examples of applications with little or no margin for improvement because they run several CPU-intensive threads most of the time.

When optimizing the two objectives simultaneously, the GA explores the search space to find the configuration that balances out the trade-off between execution time and energy consumption. Our results in Figure 7.4 show that in all cases there is an improvement in both execution time and energy consumption. On average, GA that optimizes both execution time and energy consumption, achieve performance improvement of 15% and energy savings of 26%.



(a) GA perf



(b) GA energy

Figure 7.3: Comparing improvement in Execution time and improvement in Energy consumption using (a) GA perf and (b) GA energy

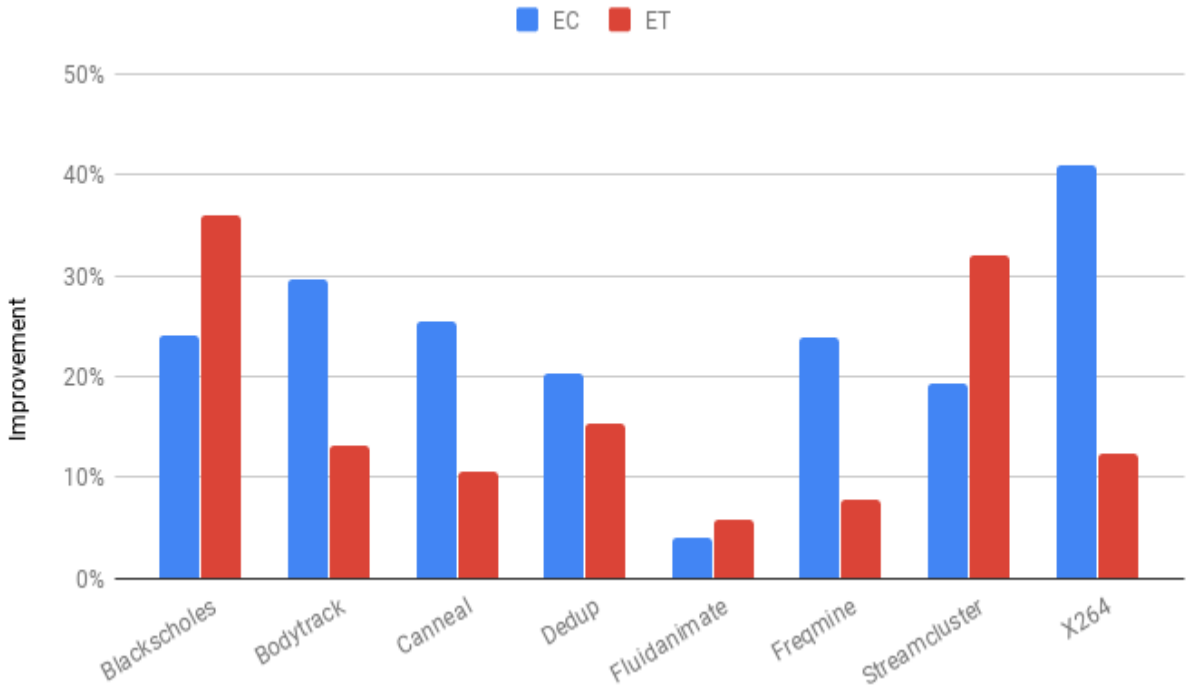
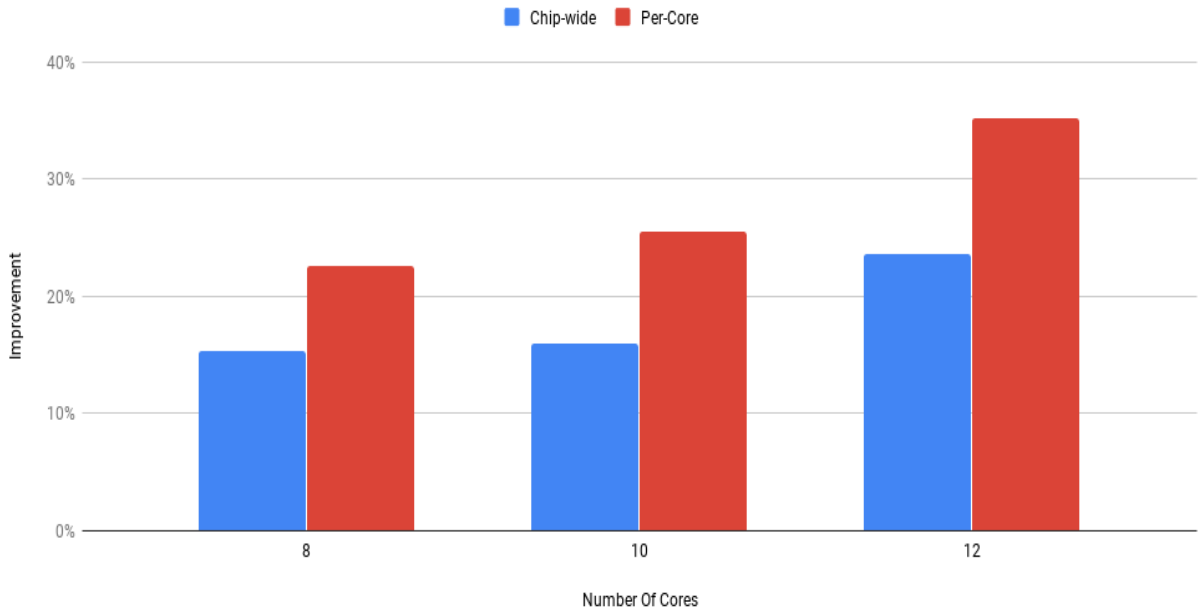


Figure 7.4: Comparing improvement in Execution time and improvement in Energy consumption using GA optimizing both

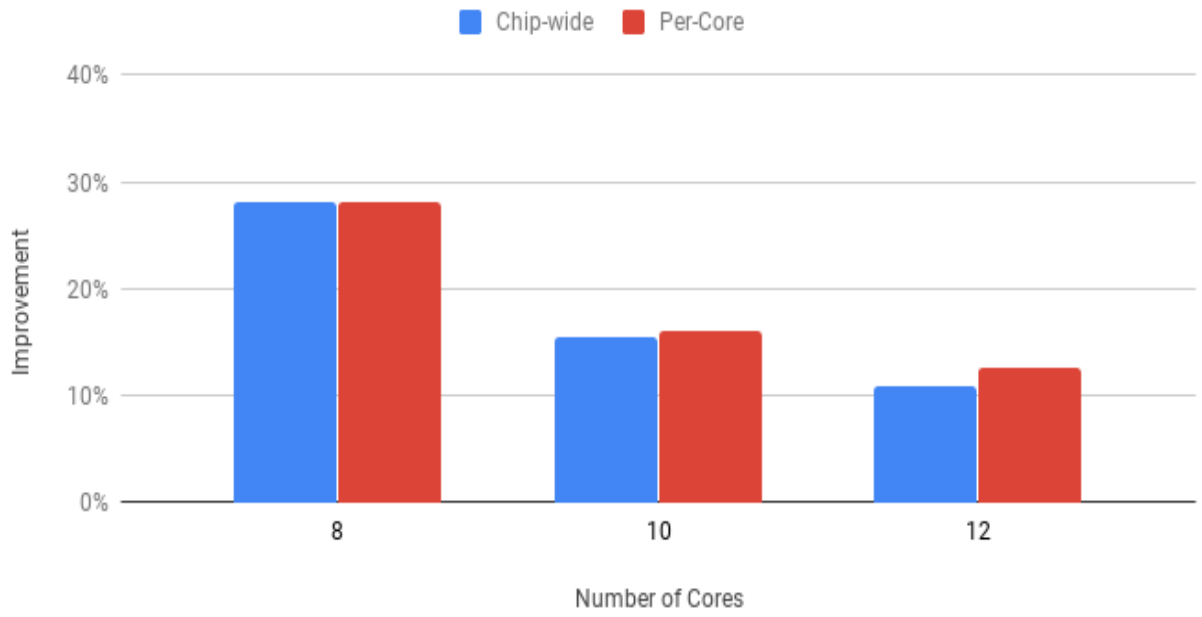
7.4 Pre-core Configuration

To evaluate the per-core configuration to a chip-wide configuration, we construct 3 multi-program workloads composed of PARSEC benchmarks, one for 8 core configuration, one for 10 core configuration and one for 12 core configuration. Each core runs one application. In this case the number of cores is fixed (not scaled) but each core will have a different configuration according to the application it is running. The system performance is the aggregate throughput achieved by all cores in the system. The system energy is the summation of the energy consumption of all cores in the system.The improvement of both chip-wide and per-core configuration is based on comparing the best configurations of the final generation of GA to the baseline.

Figure.7.5 compares the improvement in energy consumption and performance of the 3 work-



(a) Improvement in Performance (IPC)



(b) Improvement in Energy consumption

Figure 7.5: Comparing chip-wide and per-core configuration in terms of (a) improvement in IPC and (b) improvement in Energy consumption

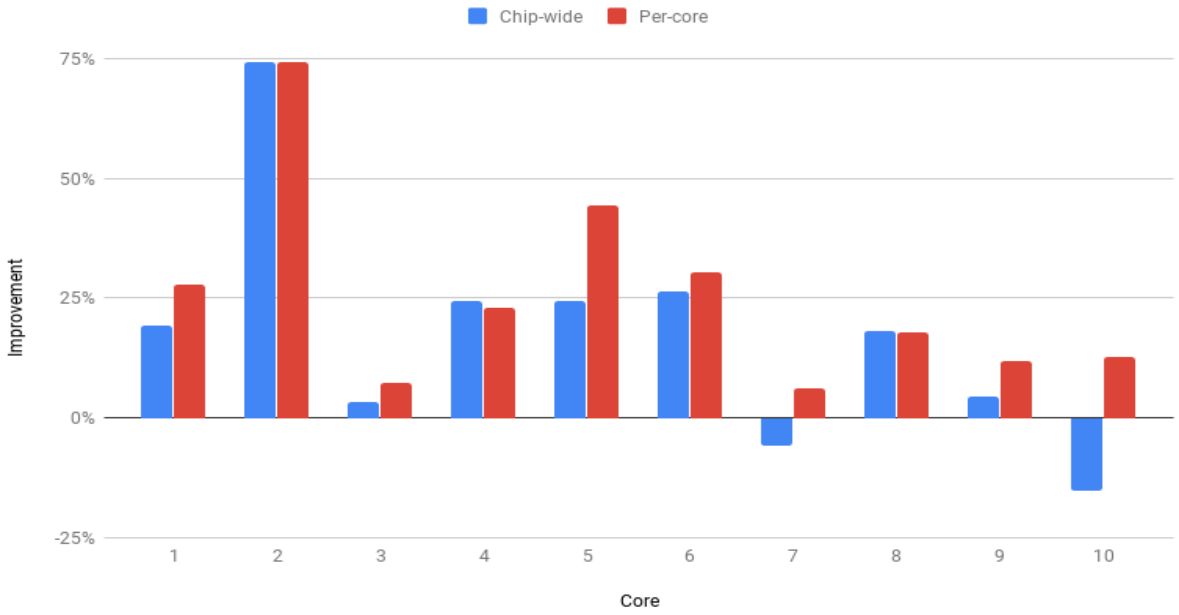
loads using chip-wide and Per-core configuration. The first observation is that per-core configuration always result in more improvement than chip-wide configuration. That is because it allows for more flexibility when scaling so that each core will have a more tailored configuration to its workload.

The results show two types of trends. Figure 7.5a shows that as the number of cores increases the improvement in the system performance increases for both chip-wide configuration and per-core configuration. In addition the increase in improvement of the per-core configuration over the chip-wide configuration also grow with the number of cores.

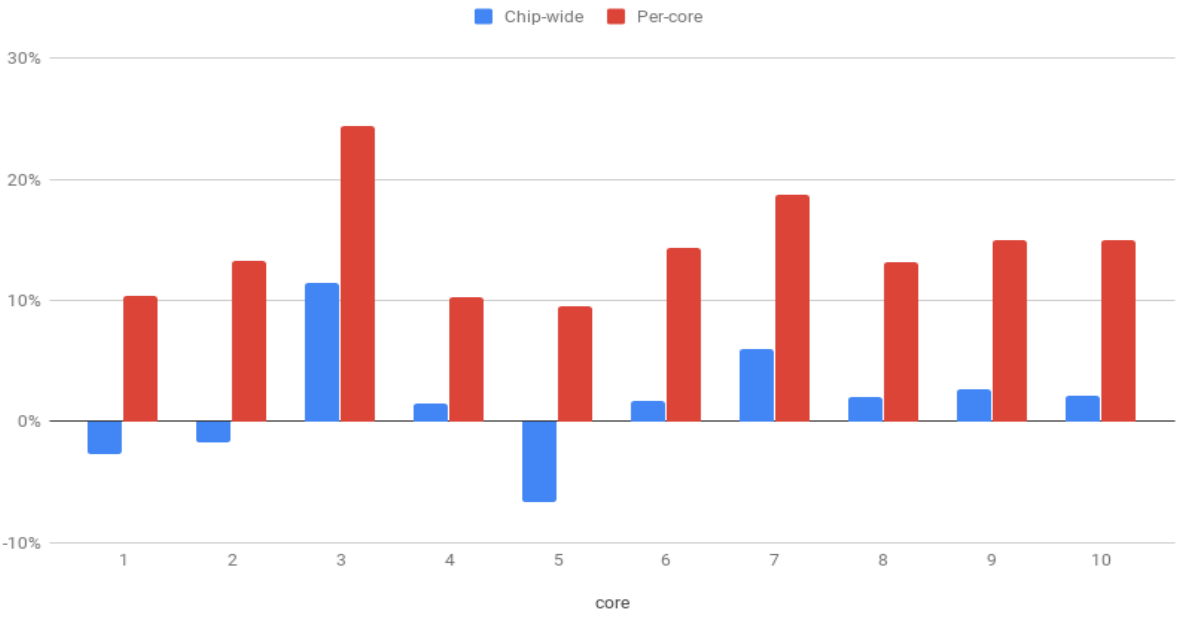
Figure 7.5b shows that as the number of cores increases the improvement in the system energy consumption decreases for both chip-wide configuration and per-core configuration. In addition the increase in improvement of the per-core configuration over the chip-wide configuration is minimal due to the increase of leakage energy with the increase of the number of cores.

Figure 7.6 shows the comparison between chip-wide and per-core configurations at the core level for a 10-core workload. For some cores (1,2, and 5) the chip-wide configuration improves the core IPC but it leads to consuming more energy. For other cores (7 and 10), the energy consumption was reduced at the cost of the performance. In chip-wide configuration, the multi-objective GA searches for a uniform configuration that optimizes both the overall system IPC and energy consumption but this would not insure that all the cores will achieve improvement in both performance and energy consumption.

In the case of per-core configuration, the GA has more freedom to search for a configuration that is tailored to the specific workload of each core. The results in Figure 7.6 show that all the cores achieved an improvement in both performance and energy consumption.



(a) Improvement in Performance (IPC)



(b) Improvement in Energy consumption

Figure 7.6: Comparing chip-wide and per-core configuration for each core for the 10 core workload in terms of (a) improvement in IPC and (b) improvement in Energy consumption

Chapter 8

Conclusions and Future Work

The problem of Dark Silicon limits the number of devices that can be simultaneously active in a multicore system. In other words, the portion of a chip that can be turned on at any given point is limited due to the chip power constraints and heat dissipation. We have demonstrated that there is a clear need for algorithms that control the interplay between different power management techniques as power is increasingly constrained. We have shown the limitations of isolated power management techniques in exploring the reconfiguration search space because considering a single granularity of the architecture may lead to sub-optimal solutions. In addition, reconfiguring with the sole objective of improving the performance can lead to consuming even more power than needed.

We have studied the effectiveness of combining different scaling techniques in improving the performance and reducing the energy consumption of multicore systems. We have achieved our other objective to show the importance of considering both systems performance and energy consumption when selecting the optimal reconfiguration. To achieve these goals, we have devised a multi-technique approach which integrates scaling techniques at multiple granularities to explore new potentials for improvement in performance and energy consump-

tion. First, we developed a methodology for characterizing the application behavior using sampling and response surface model. Then, these models can be used in a Genetic Algorithm based multi-objective optimization algorithm to explore the reconfiguration search space.

For evaluation, experiments were conducted on a simulated 12 core architecture. First, the different scaling techniques were evaluated independently for multiple benchmarks to optimize performance and energy consumption. We concluded that there is no one power management method that works best for all benchmarks for both performance and energy consumption. The second experiment consisted in applying the chip-wide reconfiguration. Our experiments have shown that the performance could improve by 15% on average while achieving energy savings of up to 26%. Finally using a per-core configuration improves the performance by 25% on average and reduces the energy by 18%.

The work presented in this dissertation can be extended in a number of ways:

- Our work has focused on multi-threaded workloads from PARSEC benchmark suite. However, a variety of workloads is available for our simulation infrastructure. This study can be extended to include workloads from a variety of industry or open-source standards, such as shared memory and communication bases workloads.

- Another extension could be the use of a method other than sampling to characterize the application. One option is to monitor different metrics and activities like queues utilization, miss rates, and issue rates to make reconfiguration decisions.

- Our multi-objective optimization is based on genetic algorithms. We use the weighted sum approach for the fitness function. We could instead test other ways to implement the multi-objective optimization such as Strength Pareto Evolutionary Algorithm SPEA . Also, a possible extension to this work is to investigate alternative heuristic algorithms. For example, simulated annealing (SA), Particle swarm optimization (PSO)

and Differential Evolution (DE). Finally, we can apply a similar methodology on heterogeneous CPU-GPU architectures where cores can be scaled differently.

Bibliography

[1] 42 Years of Microprocessor Trend Data. https://www.karlrupp.net/2018/02/42-years-of-microprocessor-trend-data/.

[2] Fitness landscape. https://en.wikipedia.org/wiki/Fitness_landscape.

[3] Reducing a 3 factor full factorial design to a half fraction design. https://www.researchgate.net/figure/Reducing-a-3-factor-full-factorial-design-to-a-half-fraction-design_fig4_256117633.

[4] D. H. Albonesi, R. Balasubramonian, S. Dropsbo, S. Dwarkadas, E. G. Friedman, M. C. Huang, V. Kursun, G. Magklis, M. L. Scott, G. Semeraro, et al. Dynamically tuning processor resources with adaptive processing. *Computer*, 36(12):49–58, 2003.

[5] R. Bergamaschi, G. Han, A. Buyuktosunoglu, H. Patel, I. Nair, G. Dittmann, G. Janssen, N. Dhanwada, Z. Hu, P. Bose, et al. Exploring power management in multi-core systems. In *Proceedings of the 2008 Asia and South Pacific Design Automation Conference*, pages 708–713. IEEE Computer Society Press, 2008.

[6] C. Bienia and K. Li. Parsec 2.0: A new benchmark suite for chip-multiprocessors. In *Proceedings of the 5th Annual Workshop on Modeling, Benchmarking and Simulation*, volume 2011, 2009.

[7] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, et al. The gem5 simulator. *ACM SIGARCH Computer Architecture News*, 39(2):1–7, 2011.

[8] R. Bitirgen, E. Ipek, and J. F. Martinez. Coordinated management of multiple interacting resources in chip multiprocessors: A machine learning approach. In *Proceedings of the 41st annual IEEE/ACM International Symposium on Microarchitecture*, pages 318–329. IEEE Computer Society, 2008.

[9] G. E. Box and N. R. Draper. *Empirical model-building and response surfaces.* John Wiley & Sons, 1987.

[10] M. D. Buhmann. *Radial basis functions: theory and implementations*, volume 12. Cambridge university press, 2003.

[11] D.-S. Chiou, D.-C. Juan, Y.-T. Chen, and S.-C. Chang. Fine-grained sleep transistor sizing algorithm for leakage power minimization. In *Proceedings of the 44th annual Design Automation Conference*, pages 81–86. ACM, 2007.

[12] R. H. Dennard, F. H. Gaensslen, V. L. Rideout, E. Bassous, and A. R. LeBlanc. Design of ion-implanted mosfet's with very small physical dimensions. *IEEE Journal of Solid-State Circuits*, 9(5):256–268, 1974.

[13] C. Dubach, T. M. Jones, E. V. Bonilla, and M. F. O'Boyle. A predictive model for dynamic microarchitectural adaptivity control. In *Proceedings of the 2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture*, pages 485–496. IEEE Computer Society, 2010.

[14] H. Esmaeilzadeh, E. Blem, R. S. Amant, K. Sankaralingam, and D. Burger. Dark silicon and the end of multicore scaling. *IEEE Micro*, 32(3):122–134, 2012.

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

[16] H. R. Ghasemi and N. S. Kim. Rcs: runtime resource and core scaling for power-constrained multi-core processors. In *Proceedings of the 23rd international conference on Parallel architectures and compilation*, pages 251–262. ACM, 2014.

[17] D. Gibson and D. A. Wood. Forwardflow: a scalable core for power-constrained cmps. In *ACM SIGARCH Computer Architecture News*, volume 38, pages 14–25. ACM, 2010.

[18] J. H. Holland. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press, 1992.

[19] Z. Hu, A. Buyuktosunoglu, V. Srinivasan, V. Zyuban, H. Jacobson, and P. Bose. Microarchitectural techniques for power gating of execution units. In *Proceedings of the 2004 international symposium on Low power electronics and design*, pages 32–37. ACM, 2004.

[20] M. C. Huang, J. Renau, and J. Torrellas. Positional adaptation of processors: application to energy reduction. In *30th Annual International Symposium on Computer Architecture, 2003. Proceedings.*, pages 157–168. IEEE, 2003.

[21] C. Isci, A. Buyuktosunoglu, C.-Y. Cher, P. Bose, and M. Martonosi. An analysis of efficient multi-core global power management policies: Maximizing performance for a given power budget. In *Microarchitecture, 2006. MICRO-39. 39th Annual IEEE/ACM International Symposium on*, pages 347–358. IEEE, 2006.

[22] A. Iyer and D. Marculescu. Microarchitecture-level power management. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 10(3):230–239, 2002.

[23] R. Jayaseelan and T. Mitra. A hybrid local-global approach for multi-core thermal management. In *Proceedings of the 2009 International Conference on Computer-Aided Design*, pages 314–320. ACM, 2009.

[24] S. S. Jha, W. Heirman, A. Falcón, T. E. Carlson, K. Van Craeynest, J. Tubella, A. González, and L. Eeckhout. Chrysso: An integrated power manager for constrained many-core processors. In *Proceedings of the 12th ACM International Conference on Computing Frontiers*, page 19. ACM, 2015.

[25] W. Kim, M. S. Gupta, G.-Y. Wei, and D. Brooks. System level analysis of fast, per-core dvfs using on-chip switching regulators. In *2008 IEEE 14th International Symposium on High Performance Computer Architecture*, pages 123–134. IEEE, 2008.

[26] A. Konak, D. W. Coit, and A. E. Smith. Multi-objective optimization using genetic algorithms: A tutorial. *Reliability Engineering & System Safety*, 91(9):992–1007, 2006.

[27] V. Kontorinis, A. Shayan, D. M. Tullsen, and R. Kumar. Reducing peak power with a table-driven adaptive processor core. In *Proceedings of the 42nd annual IEEE/ACM international symposium on microarchitecture*, pages 189–200. ACM, 2009.

[28] R. Kumar and G. Hinton. A family of 45nm ia processors. In *2009 IEEE International Solid-State Circuits Conference-Digest of Technical Papers*, pages 58–59. IEEE, 2009.

[29] B. C. Lee and D. Brooks. Efficiency trends and limits from comprehensive microarchitectural adaptivity. *ACM SIGARCH computer architecture news*, 36(1):36–47, 2008.

[30] J. Lee and N. S. Kim. Optimizing throughput of power-and thermal-constrained multicore processors using dvfs and per-core power-gating. In *Design Automation Conference, 2009. DAC'09. 46th ACM/IEEE*, pages 47–50. IEEE, 2009.

[31] J. Lee, V. Sathisha, M. Schulte, K. Compton, and N. S. Kim. Improving throughput of power-constrained gpus using dynamic voltage/frequency and core scaling. In *Parallel Architectures and Compilation Techniques (PACT), 2011 International Conference on*, pages 111–120. IEEE, 2011.

[32] J. Li and J. F. Martinez. Dynamic power-performance adaptation of parallel computation on chip multiprocessors. In *High-Performance Computer Architecture, 2006. The Twelfth International Symposium on*, pages 77–87. IEEE, 2006.

[33] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi. Mcpat: an integrated power, area, and timing modeling framework for multicore and manycore architectures. In *Microarchitecture, 2009. MICRO-42. 42nd Annual IEEE/ACM International Symposium on*, pages 469–480. IEEE, 2009.

[34] B. Liu, M. H. Foroozannejad, S. Ghiasi, and B. M. Baas. Optimizing power of many-core systems by exploiting dynamic voltage, frequency and core scaling. In *Circuits and Systems (MWSCAS), 2015 IEEE 58th International Midwest Symposium on*, pages 1–4. IEEE, 2015.

[35] H.-Y. Liu and L. P. Carloni. On learning-based methods for design-space exploration with high-level synthesis. In *Proceedings of the 50th annual design automation conference*, page 50. ACM, 2013.

[36] Y. Liu, G. Cox, Q. Deng, S. C. Draper, and R. Bianchini. Fastcap: An efficient and fair algorithm for power capping in many-core systems. In *Performance Analysis of Systems and Software (ISPASS), 2016 IEEE International Symposium on*, pages 57–68. IEEE, 2016.

[37] N. Madan, A. Buyuktosunoglu, P. Bose, and M. Annavaram. Guarded power gating in a multi-core setting. In *International Symposium on Computer Architecture*, pages 198–210. Springer, 2010.

[38] K. Meng, R. Joseph, R. P. Dick, and L. Shang. Multi-optimization power management for chip multiprocessors. In *Proceedings of the 17th international conference on Parallel architectures and compilation techniques*, pages 177–186. ACM, 2008.

[39] P.-J. Micolet, A. Smith, and C. Dubach. A study of dynamic phase adaptation using a dynamic multicore processor. *ACM Transactions on Embedded Computing Systems (TECS)*, 16(5s):121, 2017.

[40] S. Mittal, Y. Cao, and Z. Zhang. Master: A multicore cache energy-saving technique using dynamic cache reconfiguration. *IEEE Transactions on very large scale integration (VLSI) systems*, 22(8):1653–1665, 2014.

[41] G. E. Moore. Cramming more components onto integrated circuits, electronics magazine, 1965.

[42] J. MüLler, C. A. Shoemaker, and R. Piché. So-mi: A surrogate model algorithm for computationally expensive nonlinear mixed-integer black-box global optimization problems. *Computers & Operations Research*, 40(5):1383–1400, 2013.

[43] G. Palermo, C. Silvano, and V. Zaccaria. Respir: a response surface-based pareto iterative refinement for application-specific design space exploration. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 28(12):1816–1829, 2009.

[44] I. Park, C. L. Ooi, and T. Vijaykumar. Reducing design complexity of the load/store queue. In *Proceedings of the 36th annual IEEE/ACM International Symposium on Microarchitecture*, page 411. IEEE Computer Society, 2003.

[45] P. Petrica, A. M. Izraelevitz, D. H. Albonesi, and C. A. Shoemaker. Flicker: A dynamically adaptive architecture for power limited multicore systems. In *ACM SIGARCH computer architecture news*, volume 41, pages 13–23. ACM, 2013.

[46] M. Powell, S.-H. Yang, B. Falsafi, K. Roy, and T. Vijaykumar. Gated-v dd: a circuit technique to reduce leakage in deep-submicron cache memories. In *Proceedings of the 2000 international symposium on Low power electronics and design*, pages 90–95. ACM, 2000.

[47] K. K. Rangan, G.-Y. Wei, and D. Brooks. Thread motion: fine-grained power management for multi-core systems. In *ACM SIGARCH Computer Architecture News*, volume 37, pages 302–313. ACM, 2009.

[48] G. S. Ravi and M. H. Lipasti. Charstar: Clock hierarchy aware resource scaling in tiled architectures. *ACM SIGARCH Computer Architecture News*, 45(2):147–160, 2017.

[49] J. Sharkey, A. Buyuktosunoglu, and P. Bose. Evaluating design tradeoffs in on-chip power management for cmps. In *Low Power Electronics and Design (ISLPED), 2007 ACM/IEEE International Symposium on*, pages 44–49. IEEE, 2007.

[50] K. Shi and D. Howard. Sleep transistor design and implementation-simple concepts yet challenges to be optimum. In *2006 International Symposium on VLSI Design, Automation and Test*, pages 1–4. IEEE, 2006.

[51] R. Strong, J. Mudigonda, J. C. Mogul, N. Binkert, and D. Tullsen. Fast switching of threads between cores. *ACM SIGOPS Operating Systems Review*, 43(2):35–45, 2009.

[52] R. Teodorescu and J. Torrellas. Variation-aware application scheduling and power management for chip multiprocessors. In *ACM SIGARCH computer architecture news*, volume 36, pages 363–374. IEEE Computer Society, 2008.

[53] R. Unal, R. Lepsch, and M. McMillin. Response surface model building and multidisciplinary optimization using d-optimal designs. In *7th AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization*, page 4759, 1998.

[54] A. Vedaldi and K. Lenc. Matconvnet: Convolutional neural networks for matlab. In *Proceedings of the 23rd ACM international conference on Multimedia*, pages 689–692. ACM, 2015.

[55] A. Vega, A. Buyuktosunoglu, H. Hanson, P. Bose, and S. Ramani. Crank it up or dial it down: coordinated multiprocessor frequency and folding control. In *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 210–221. ACM, 2013.

[56] D. Wei, Z. Cui, and J. Chen. Uncertainty quantification using polynomial chaos expansion with points of monomial cubature rules. *Computers & Structures*, 86(23-24):2102–2108, 2008.

[57] C. J. Wu and M. S. Hamada. *Experiments: planning, analysis, and optimization*, volume 552. John Wiley & Sons, 2011.

[58] S. Xydis, G. Palermo, V. Zaccaria, and C. Silvano. Spirit: spectral-aware pareto iterative refinement optimization for supervised high-level synthesis. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 34(1):155–159, 2015.