UNIVERSITY OF CALIFORNIA,
IRVINE

Optimization of Heterogeneous NoC for Fused CPU-GPU Architecture

DISSERTATION

submitted in partial satisfaction of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

in Computer Engineering

by

Lulwah Alhubail

Dissertation Committee:
Professor Nader Bagherzadeh, Chair
Professor Chen-Yu (Phillip) Sheu
Professor Alexander V. Veidenbaum

2019

DEDICATION

To my husband and the love of my life; Abdulaziz Murad.
To the light of my life, my kids Arwa and Sulaiman Murad.
To my precious parents, loving family, and supportive friends.

TABLE OF CONTENTS

LIST OF FIGURES

Page

LIST OF TABLES

Page

LIST OF ALGORITHMS

LIST OF ABBREVIATIONS

BS	Buffer Size
BW	Bandwidth
CR	Crossover Rate
GA	Genetic Algorithm
GPU	Graphics Processing Unit
HSA	Heterogeneous System Architecture
MC	Memory Controller
MR	Mutation Rate
NoC	Network-on-Chip
PE	Processing Element
PMX	Partially Mapped Crossover
PW	Page Walk cache
SA	Simulated Annealing
SM	Streaming Multiprocessor
SPEA	Strength Pareto Evolutionary Algorithm
TS	Tabu Search
TLP	Thread Level Parallelism
VC	Virtual Channel

ACKNOWLEDGMENTS

”Whoever does not thank people does not thank Allaah”

Narrated by Al-Tirmidhi on the Prophet peace and blessings of Allaah be upon him.

First of all, I would have never accomplished what I did without the guidance and blessings of God Almighty (Allaah) and the support of many other people in my life.

I like to express my sincerest gratitude for my advisor, Professor Nader Bagherzadeh, for his continuous guidance and support all these years. I learned a lot from him during my years of research, and without his guidance this work would have never been. It has been an honor to work with him and learn from his experience. I am, also, thankful for my committee members Professor Chen-Yu (Phillip) Sheu and Professor Alexander V. Veidenbaum.

I am so grateful for the existence of my husband, Abdulaziz Murad, in my life. Without his endless love, patience, sacrifices, and support, I would not be able to fulfill my dream. I cannot find any suitable words to thank him enough. I, also, like to thank my kids Arwa and Sulaiman Murad for accompanying me in this journey and brightening my life.

I like to extend my thanks to my precious parents, Ali Alhubail and Hallimah Alkandari, for their encouragement and prayers. Without their guidance and sacrifices, I would not be where I am now. I am also grateful for the support and love of my sisters; Mona, Hind, Maryam, and Rawan and my brother Abdullah. Also, I appreciate the patience and support of my father and mother in-laws.

Besides, I extend my gratitude for my best friends Munirah Almatooq and Shouq Alsubaihi for their support throughout my research journey. Their valuable opinions inspired me during these years and and their emotional support gave me the courage to pursue my academic goal.

I want to thank Professor Hamid Zarandi from the Amirkabir University of Technology of Iran for his help and valuable insights into this work. Many thanks to my colleagues in the Advanced Computer Architecture Group (ACAG) for their helpful comments and views.

Finally, I would like to thank Kuwait University for this opportunity of lifetime of providing me with a scholarship to attain my Ph.D degree.

CURRICULUM VITAE

Lulwah Alhubail

EDUCATION

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

Master of Science in Computer Engineering **2008**
Kuwait University *Kuwait, Kuwait*

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

RESEARCH EXPERIENCE

Graduate Research Assistant **2013–2019**
University of California, Irvine *Irvine, California*

TEACHING EXPERIENCE

Teaching Assistant **2010–2013**
Public Authority of Applied Sciences and Training *Kuwait, Kuwait*

Part-time Teacher assistant **Summer 2007**
College of Engineering and Petroleum, Kuwait University *Kuwait, Kuwait*

PROFESSIONAL EXPERIENCE

Computer Engineer **2005–2010**
Deanship of Admission and Registration, Kuwait University *Kuwait, Kuwait*

Part-time assistant **2003–2004**
Directing Office, College of Engineering and Petroleum, Kuwait University *Kuwait, Kuwait*

CERTIFICATES

Public Speaking: Activate to Captivate **2017**
Graduate Resource Center, University of California, Irvine

Mentoring Excellence Program **2016**
Graduate Resource Center, University of California, Irvine

Excellence in Engineering Communication **2016**
Graduate Resource Center, University of California, Irvine

Oracle AS Discover 10g: Create Queries and Reports ED 1 PRV **2007**
Kuwait

REFEREED CONFERENCE PUBLICATIONS

Power and Performance Optimal NoC Design for CPU-GPU Architecture Using Formal Models **March 2019**
Design, Automation, and Test in Europe (DATE) conference

Synthesizing Clustered, Secured, and Hierarchical Networks through Genetic Algorithms **January 2010**
International Conference on Intelligent System Modeling and Simulation ISMS conference

ABSTRACT OF THE DISSERTATION

Optimization of Heterogeneous NoC for Fused CPU-GPU Architecture

By

Lulwah Alhubail

Doctor of Philosophy in Computer Engineering

University of California, Irvine, 2019

Professor Nader Bagherzadeh, Chair

Heterogeneous computing architectures that utilize both CPU and GPU have been the trend nowadays. Several products from AMD, Intel, and NVIDIA have emerged that fused CPU and GPU on the same chip. In such architectures, different processing elements (PEs), including many CPU cores, GPU cores, memory controllers (MCs), and caches, are connected through a common interconnection. CPU and GPU exhibit different network behaviors; CPU tends to be latency-sensitive and GPU, with its high thread level parallelism (TLP), tends to be throughput hungry. Using homogeneous interconnect for such heterogeneous processors can result in performance degradation and power increase. This dissertation focused on designing a heterogeneous mesh-style network-on-chip (NoC) to connect heterogeneous CPU-GPU processors while considering their diametric network demands.

There are many aspects to consider when designing a 2D mesh NoC. Firstly, the placement of the PEs within the mesh. Secondly, setting the NoC parameters: the size of the router's buffer, the number of virtual channels, and the bandwidth of the links. This dissertation tackled all these problems simultaneously. Moreover, to design a heterogeneous NoC, heterogeneity was explored at the router's port and link level, where each port of each router can have different buffer size and number of virtual channels, and each link can have different bandwidth. This explodes the design space and makes exploring all possible design combinations using simulation very difficult.

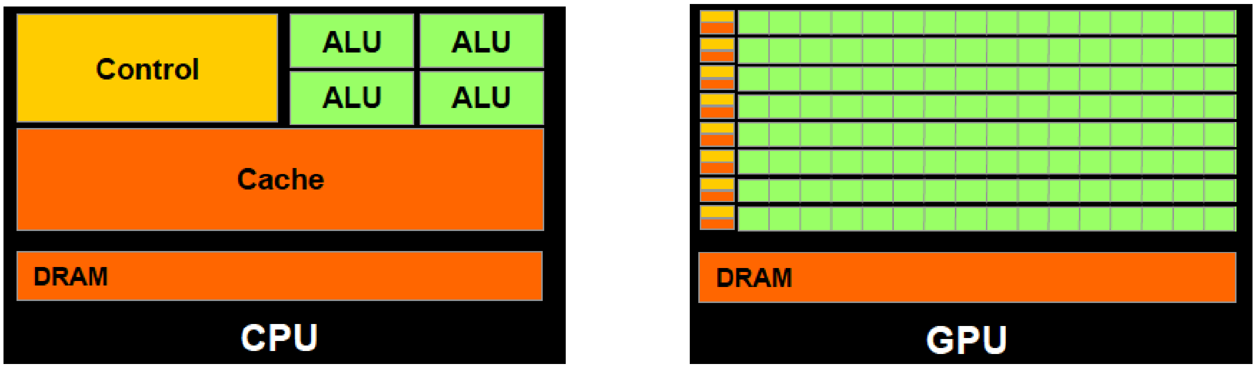
In this dissertation, heuristic-based optimization methods were proposed to obtain a near-optimal heterogeneous NoC design. Firstly, a method based on Genetic Algorithm (GA) to get a design with optimal performance in terms of the average network latency. An analytical model based on queueing theory that supports virtual channels was proposed to get a performance measure of the design. Secondly, a multi-objective method based on the Strength Pareto Evolutionary Algorithm 2 (SPEA2) to get an optimal design in terms of the performance and the power of NoC. Also, an activity based power model was proposed to get the power of the design. The optimal designs were validated using a full-system simulator.

Chapter 1

Introduction

Heterogeneous Systems Architectures (HSA) are the trend nowadays. These systems do not depend only on adding more cores of the same type but also use more than one kind of processor to enhance performance and power. Graphics Processing Units (GPUs) are attractive processing cores for high performance and energy-efficient computing systems, so current high-performance computers, servers, and supercomputers heavily utilize them to scale up the throughput [1]. While conventional CPUs are based on instruction level parallelism, GPUs are designed to exploit data and thread level parallelisms for performance enhancements [36][17].

Using GPU as a standalone is promising, but combining it with CPU in heterogeneous computing systems is more awarding in terms of utilizing the unique architectural strengths of each core [34][41]. Modern CPUs are typically out-of-order cores that run at high frequency and use a hierarchy of large-sized caches to tolerate latency, see Figure 1.1a; hence they are the best match for latency-sensitive and irregular applications. GPUs, on the other hand, use a large number of in-order cores that share their control unit and operate at lower frequency and smaller sized-caches, see Figure 1.1b, so they are most suited for throughput-



(a) CPU

(b) GPU

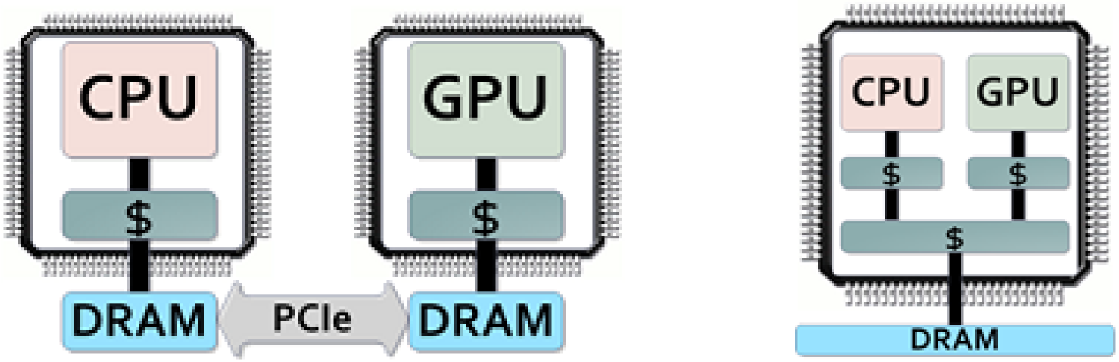
Figure 1.1: CPU vs. GPU architecture.

critical and regular applications. This same difference in their architectural that makes it appealing to combine them, imposes different challenges in totally exploiting their potentials [30]. Most importantly, how to maximize the utilization of this architecture while optimizing performance and power consumption.

1.1 Heterogeneous CPU-GPU Architecture

Heterogeneous CPU-GPU Architectures can be either discrete or fused, see Figure 1.2. In discrete architectures, CPU and GPU lie on a different chip, and they are connected through the PCIe. When running a program on GPU, the data need to be copied between the CPU (host) memory and GPU (device) memory through the PCIe. This imposes a burden on the PCIe and makes it a bottleneck, especially for applications that require co-computing.

Another design choice is to fuse both CPU and GPU on the same chip, eliminating the PCIe bottleneck. Several products from AMD [37][42], Intel [22][23], and NVIDIA [3][14] have adopted this design choice. In this architecture, two memory schemes are available. Firstly, the main memory is divided into a CPU part and a GPU part. In this case, the data still need to be copied between the two parts, but this is done by high-speed block transfer engines



(a) Discrete

(b) Fused

Figure 1.2: Discrete vs. fused CPU-GPU architecture. [1]

mitigating the slow effect of PCIe. The effectiveness of this approach compared to discrete CPU-GPU architecture has been investigated by [13]. Their study shows that the costs of data transfer can be reduced by six-fold resulting in application's performance improvement of three-fold. In the second scheme, the main memory is shared and can be accessed by both the CPU and GPU, avoiding data transfer penalty between the host memory and the device memory. This dissertation adopted the second memory scheme.

A closer look at the fused CPU-GPU architecture is shown in Figure 1.3. It consists of many CPU cores and many GPU cores (Streaming Multiprocessor (SM) in NVIDIA's term), each with its private L1 cache. Both CPU and GPU utilize a common interconnection network to the shared L2 cache, memory controllers (MCs), and fully shared physical memory leading to other resources sharing challenges. The difference between the CPU and GPU intensifies the shared resource contention. Especially, the high degree of thread level parallelism (TLP) nature of GPU which leads to frequent network injections.

Since the interconnection network connects all the components and all the communications traverse through it, this dissertation focused on designing an efficient interconnection that takes the difference in architecture and the varying needs of the CPU and GPU into consideration. Industry fused architectures such as Intel Sandy [22] and Ivy [23] bridge use a

[1] Source: `https://sites.google.com/site/fusionsimulator/`

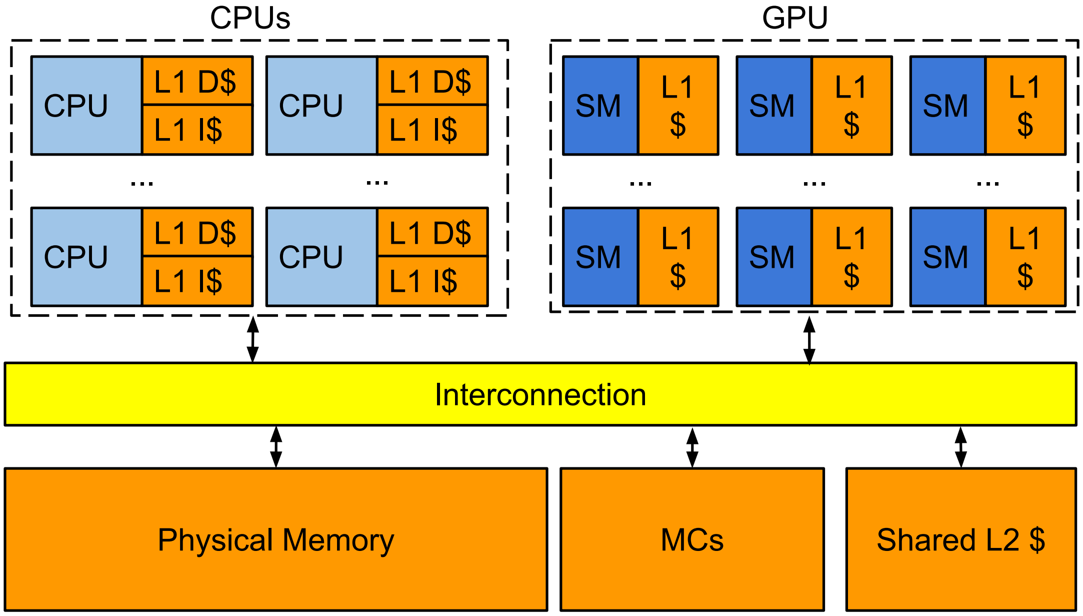


Figure 1.3: Fused CPU-GPU architecture with many CPU cores and GPU cores sharing an interconnection to the shared L2 cache, MCs and physical memory.

bi-directional ring style bus interconnection while AMD Fusion [42] and NVIDIA's Project Denver [14] adopt a crossbar interconnection. While these interconnections may provide satisfactory performance, they might not be as scalable as a mesh style Network-on-Chip (NoC), which is known for its reliability and scalability.

1.2 Network-on-Chip

A 2D mesh NoC is composed of a network of routers; each is connected to a PE that can be a computational processor or a memory, see Figure 1.4. The router can have up to five ports, depending on its position in the mesh, each with n virtual channels (VCs) with a fixed buffer (BS) size b, that is used to transmit the packet over a link of bandwidth (BW) w.

Designing a 2D mesh NoC involves solving different sub-problems. Firstly, processing ele-

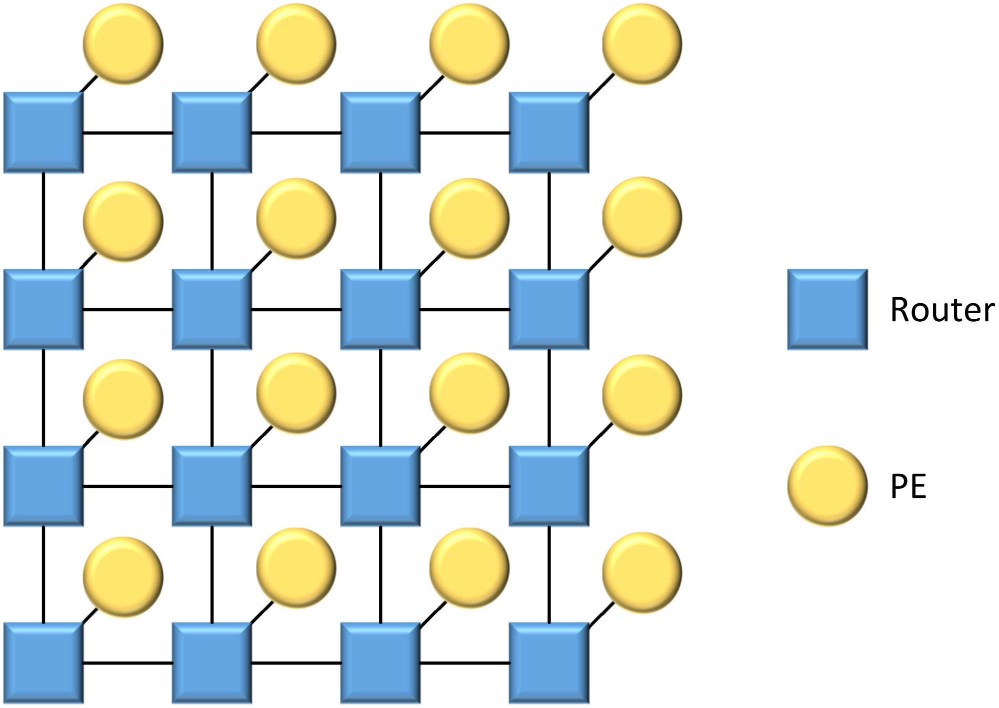


Figure 1.4: A 4 x 4 2D mesh NoC.

ments (PEs) mapping to the routers of the mesh. Mapping problem is considered to be an NP-hard problem. The placement (mapping) of PEs within the mesh significantly affects the performance and power of the system. Secondly, configuring the NoC. The configurations can include route allocation, setting links' latency, choosing the buffer size, choosing the number of virtual channels, and the links' bandwidth, etc.

While designing homogeneous 2D NoC with identical n, b, and w values for all the routers and links is relatively easy, using a homogeneous NoC to connect heterogeneous cores with different communication demands can affect the performance and power of the system. When running applications simultaneously on CPU and GPU cores, interference between the applications is highly expected [27]. CPU tends to be latency sensitive and GPU bandwidth hungry, and with its high level of TLP, it generates massive traffic that can interfere with the CPU traffic.

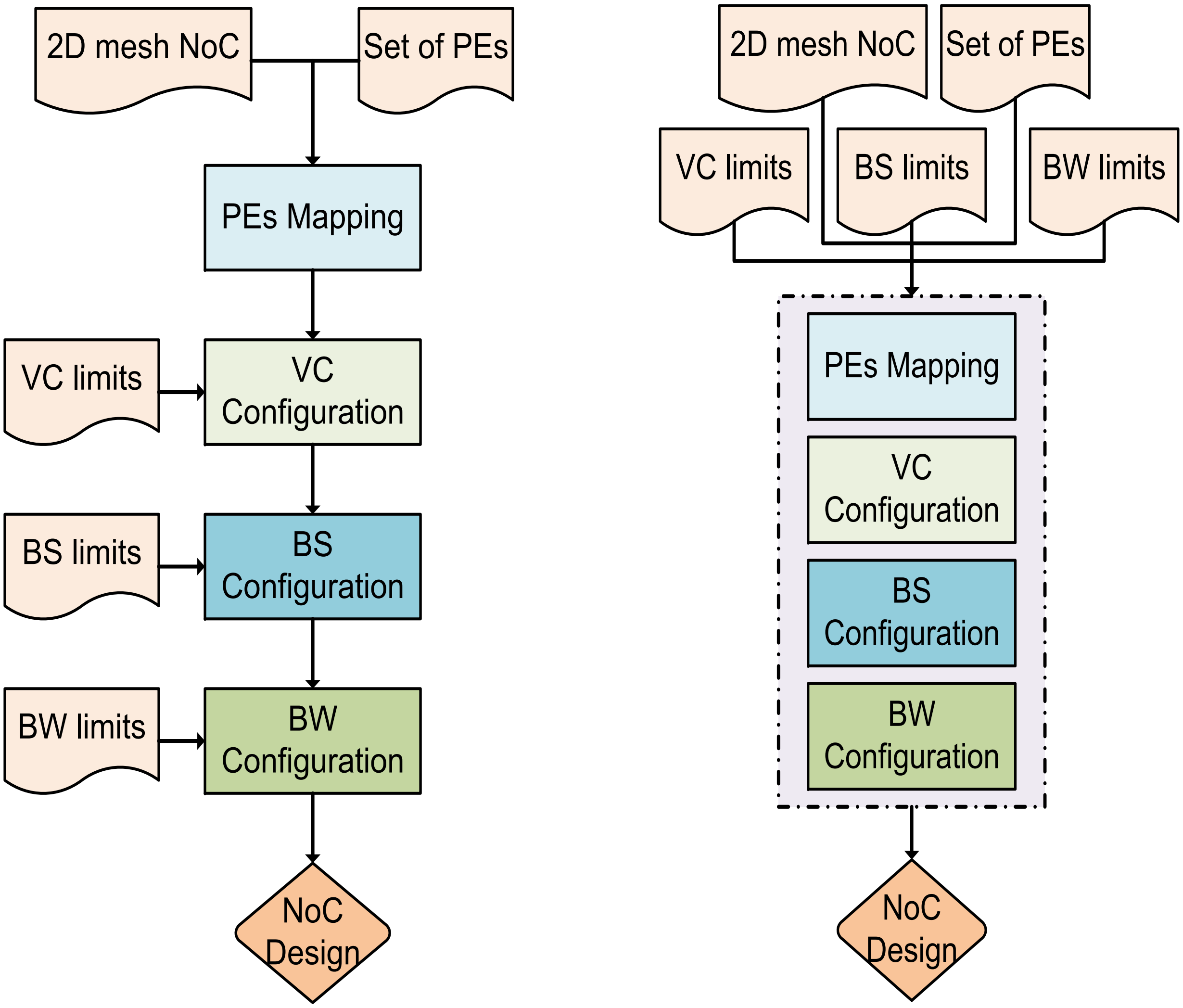
Designing a heterogeneous mesh can be challenging and can be considered on different levels:

the router level and the port level. On the router level, n and b values could be different from one router to another, while the same within all ports of the same router. On the port level, even within the same router n and b values could be different for each port. Commonly, increasing the number of VCs enhances performance but consumes more power especially when the buffers consume about 35% of the router power [29].

1.3 Dissertation Contribution

The focus of this dissertation is to provide a design methodology of heterogeneous 2D mesh NoC which targets a fused heterogeneous CPU-GPU architecture. The design must consider the diametric network demands of the CPU and GPU and aim to improve the performance and power of the system. Designing a 2D NoC involves solving different sub-problems: mapping PEs to the routers of the mesh, finding the number of virtual channels and the buffer size for each port of each router, and finding the bandwidth of each link in the mesh. There could be two approaches to solve these problems, iteratively or simultaneously, see Figure 1.5. The iterative method is more straightforward than the simultaneous method. Though, it limits the search space by solving a problem depending on the solution of the previous; This dissertation adopted a simultaneous approach.

Mapping is an NP-hard problem, and considering different NoC configurations (virtual channels, buffer size, and links' bandwidth) on the port level at the same time, expands the search space. Moreover, obtaining a design that satisfies two contradicting objectives, performance and power, complicates the problem. This complexity and scope proliferate as the size of the mesh increases. Usually, designers rely on simulation to explore different design possibilities. For the proposed design problem, exploring all the possible design combinations using simulation is time-consuming and not practical. Alternatively, analytical models can be used to evaluate multiple design choices faster and accurate enough. Moreover, combining the



(a) Iterative approach

(b) Simultaneous approach

Figure 1.5: Iterative vs. simultaneous NoC design approach.

analytical models with a heuristic method can help explore many if not all possible design choices and obtain a near-optimal design choice that satisfies the intended objectives.

In this dissertation, the performance of the NoC is represented as the average packet latency. An analytical model that supports different buffer size per port is presented to get a measure of the performance of NoC design. The model is extended to support varying virtual channels per port. This model is used within an optimization method that is based on GA to get a heterogeneous NoC design with optimal performance. This design aims to solve three sub-problems simultaneously, PE placement, buffer size, and virtual channels configuration per router's ports. While the objective of this design is only the performance of the NoC, the power is considered indirectly by considering the NoC area within the optimization. The NoC area is represented as the total number of buffers in the design since the buffers of the routers in NoC represent over 75% of the total area of the interconnect [29].

A multi-objective optimization method based on SPEA2 is proposed to solve the same three sub-problems and get a Pareto-optimal heterogeneous NoC design set that satisfies performance and power. An activity-based power model is proposed to get a measure of the NoC power and is used within the optimization method.

The optimization method based on SPEA2 is extended to solve the heterogeneous bandwidth sub-problem in addition to the other three sub-problems. Both, the performance and power models are extended to support the heterogeneous bandwidth in the evaluation of the design.

Finally, the proposed methods are evaluated using full system simulator. The obtained optimal designs are validated and compared against other NoC design strategies.

The contribution of this dissertation can be summarized as follows:

- Present and develop a G/G/1 queueing theory-based model that supports arbitrary buffers per router's ports of [24] to estimate the average packet latency of NoC.

- Adjust the inaccuracy of the presented model.

- Extend the model to support arbitrary virtual channels per router's ports.

- Extend the model to support heterogeneous links' bandwidth.

- Propose an activity-based power model to estimate the power of NoC.

- Propose a method based on GA to get an optimal performance heterogeneous NoC design that solves three sub-problems, PE placement, buffer size, and virtual channels assignments per port.

- Propose a method based on SPEA2 to get an optimal heterogeneous NoC design Pareto set that satisfies two objectives: performance and power of NoC. Each NoC design solves three sub-problems, PE placement, buffer size and virtual channels assignment per port.

- Extend the SPEA2-based method to get an optimal Pareto-set for heterogeneous NoC design that solves four sub-problems: PE placement, buffer size per port configuration, virtual channels per port configuration, and bandwidth assignment per link.

- Evaluate the proposed methods using full system simulator and compare them with other design strategies.

1.4 Dissertation Organization

This dissertation is organized as follows, Chapter 2 summarizes the different NoC design approaches available in the literature. These approaches are categorized according to the type of NoC design, homogeneous or heterogeneous, and the targeted architecture, CMP or CPU-GPU architecture.

Chapter 3 aims to explain the proposed analytical model that is used to get the performance of the NoC. It starts by comparing different analytical models in the literature that estimates the average packet latency of NoC. Next, it presents the chosen model that supports arbitrary buffers. It shows how the inaccuracy of the model is adjusted and further extended to support random virtual channels and link's bandwidth. Finally, it concludes with a discussion of the accuracy of the proposed model against simulation.

Chapter 4 explains the activity-based power model that is used to get the power of NoC. This model supports heterogeneous routers with heterogeneous buffer size and virtual channels per port. It also shows how this model is extended so that each link can have a heterogeneous bandwidth.

A detail description of the proposed optimization methods is presented in Chapter 5. It starts with a description of the problem and a discussion of different approaches to tackle complex problems with large design space. Next, it explains the GA-based method by first presenting a description of the targeted NoC design that includes three sub-problems. Then, it provides a detailed description of how GA with its different operators is adopted to solve the NoC design problem. It introduces the SPEA2-based method used to solve the same problem, next, showing in details the implementation of its different operators. Finally, it describes the SPEA2-BW based approach, which is the extended version of the SPEA2-based method that solves four sub-problems. It explains in detail the addition of bandwidth as a target in the design and how the SPEA2 operators are modified to include its effect.

The goal of Chapter 6 is to evaluate the proposed method and validate them against other NoC design approaches. It starts by describing the evaluation methodology that follows different stages, the evaluation environment including the simulators and benchmarks, the evaluation criteria, and different NoC design approaches for comparison. It provides the evaluation of each proposed method. It concludes by comparing all the proposed methods, GA, SPEA2, SPEA2-BW, and shows the improvement in different evaluation criteria that each provides.

Chapter 7 concludes this dissertation summarizing all the contribution achieved. It, also, provides possible directions to extend this work.

Chapter 2

Related Work

In NoC design, an NoC can either be heterogeneous with heterogeneous routers and links or homogeneous where all the routers and the links have the same configurations. Moreover, the cores connected through the NoC can be homogeneous, that is of the same type or heterogeneous. The research in the NoC design can be classified into three categories: 1) Research that focuses on designing a homogeneous NoC, 2) Research that focuses on heterogeneous NoC design for CMP, and 3) Research that concentrates on heterogeneous NoC design connecting heterogeneous cores, specifically CPU and GPU.

2.1 Homogeneous NoC Design

In the homogeneous NoC domain, many works focused on PE or IP mapping sub-problem; IP can be homogeneous cores or CPU cores and special accelerator like DSP.

Ascia et al. [6] proposed an optimization method based on SPEA to solve the problem of mapping IP to mesh NoC. This method finds the Pareto optimal mapping in terms of performance and power.

Tei et al. [39] solved the IP mapping problem by using a GA-based technique. Their technique combines network partitioning and a heuristic crossover. The objective of their method is to minimize communication cost.

Jena et al. [21] considered using GA in two sequential phases optimization. The purpose of the first phase is to find an optimal task mapping to cores while the goal of the second phase is to obtain an optimal IP mapping to NoC. The objective of both stages is to optimize energy consumption and maximum link bandwidth.

Shin et al. [35] used GA to solve four design stages iteratively. The first stage is task mapping to IP, then mapping the IPs to tiles, choosing the routing path between communicating tiles, and finally optimizing link speed assignment. The ultimate objective of their method was to optimize energy consumption.

But all these works were on homogeneous NoC and did not consider fused CPU and GPU architectures.

2.2 Heterogeneous NoC for Homogeneous CMP

Some works considered heterogeneous NoC for homogeneous CMPs. For example, Mishra et al. [29] proposed a heterogeneous NoC that incorporates two types of routers, big and small. The big router has more VCs and wider links while the small router has less VCs and narrower links. They compared six different layouts for the placement of these two types of routers on homogeneous CMP, in terms of throughput, latency, and power. They also used a fixed buffer size for all the virtual channels.

Zhao et al. [43] considered using buffered and bufferless routers and compared eight different placements of these two types of routers on homogeneous CMP with all the buffered routers having the same size. They also proposed buffered-router aware mapping to map application threads near the buffered routers and buffered-router aware routing algorithm to move data between the buffered routers.

While these two works were based on empirical studies to compare the NoC under different configurations, Ben-Itzhak et al. [8] proposed a design methodology to optimize the area of NoC under end-to-end latency constraints. They employed simulated annealing (SA) to optimize the capacity of each link of each router and the number of virtual channels. They designed a new router that can be adjustable based on how much bandwidth each port needs. They didn't consider the effect of the buffer size.

2.3 Heterogeneous NoC for CPU-GPU Architecture

There are not so many works in the literature that studied heterogeneous NoC design for fused CPU-GPU architecture. Lee et al. presented in [26] an adaptive virtual channel partitioning for heterogeneous architecture. Their design assumed a homogeneous 2D mesh connecting CPU and GPU with separate injection queues for CPU packets and GPU packets. They proposed a feedback-directed virtual channel partitioning mechanism between the CPU traffic and GPU traffic to balance on-chip network bandwidth. The NoC in their work was homogeneous, and the placement of the cores within the mesh was not studied. In other work, [27], they surveyed the behavior of ring NoC when running CPU and GPU simultaneously. They investigated the effect of different design choices such as the number of VCs and physical channels, arbitration policy, and link configurations under four different placement of the PEs. Based on their findings they proposed an optimal ring network for heterogeneous CPU-GPU platforms. Their work focused on ring interconnection, which is not scalable, also didn't consider the effect of buffer size.

Fang et al. [16] studied the placement of two types of routers, buffered and bufferless, in CPU-GPU architecture connected with a mesh NoC. Based on the type of PE connected to the router (CPU, GPU, or MC), they classified the routers in NoC into three categories. Then, they compared the NoC speedup and energy of all the possible eight buffer's placement combinations. They also, proposed a unidirectional control flow to control the flow between buffered and bufferless routers to guarantee the elimination of flits deflection. However, they didn't evaluate the placement of PEs, and they just considered one aspect of router heterogeneity; that is the buffer size. Even for buffered routers, they used a fixed buffer size among all the routers and within the ports of the routers.

Li et al. [28] implemented a network within an area budget for CPU-GPU heterogeneous computing architecture. They proposed a 2D mesh-style on-chip heterogeneous communication infrastructure, iConn, that uses non-uniform on-chip routers with different buffer size per port. They implemented a queuing-theory based heuristic algorithm to statically reallocate the buffers to different router ports to minimize the variation of the average waiting time of each port. They also proposed to adaptively assign the buffers across all VCs at the same input port depending on the traffic. However, they only considered four different PEs' placements and a fixed number of virtual channels among all routers. Although they evaluated the proposed design in terms of power, the power was not part of the optimization process.

2.4 Summary

Although many works in the literature tackled the homogeneous NoC design problem, using homogeneous NoC is not suitable for fused CPU-GPU architecture because of the diametric network demands of these cores [27].

By comparing the related work that tackled the heterogeneous NoC design as in Table 2.1, most of them only considered one or two aspects of NoC heterogeneity, such as BS, VC, or BW. Moreover, the heterogeneity was mostly explored on the router level than on the port level; meaning all ports of the same router have the same configuration, but different routers can have a different configuration. The placement of PE either was not considered or evaluated from a predefined set of possible placements. Some of them targeted CMP architecture which is different than fused CPU-GPU architecture. Even for the works that aimed for CPU-GPU architecture, they relied on a design process of NoC that depends mostly on empirical studies. In which different predefined network configurations for a given PEs' placement were compared. This design approach limits the search space for NoC design that can be considered. Some works utilized heuristic approaches to get an optimal NoC design but mostly focus on NoC performance in their design objective neglecting the power.

Table 2.1: Comparison of Heterogeneous NoC Design Approaches in Literature

Work	Cores type	Method	PE placement	Hetro BS	Hetro VC	Hetro BW
[29]	CMP	Empirical studies	1	N	Y	Y
[43]	CMP	Empirical studies	1	Y	N	N
[8]	CMP	SA	1	N	Y	Y
[26]	CPU-GPU	Adaptive VC partitioning	5	N	N	N
[27]	CPU-GPU	Empirical studies	4	N	N	Y
[16]	CPU-GPU	Empirical studies	1	Y	N	N
[28]	CPU-GPU	Queueing theory based heuristic	4	Y	N	N

Chapter 3

Performance Model

Nowadays, most NoC performance models are based on simulations. However, the use of simulation for design optimization is not efficient. Exploring the search space of design parameters can take a long time especially when network size increases or several design parameters are considered [24]. The alternative method is to use an estimation of performance, modeled by analytical equations. With these equations, the performance of NoC designs can be obtained efficiently and can be applied easily within an optimization loop.

There are many proposed analytical models for NoC in the literature. Each varies in the degree of the complexity of their equations and the accuracy. For example, Arjomand et al. [5] proposed a power-performance model for NoCs, with arbitrary topology, buffering structure, and routing algorithm. Message generation was assumed to have Poisson distribution, and many complex equations were developed to find the effect of buffer size and virtual channels on performance and power consumption. Hu et al. [20] proposed a sophisticated model based on M/G/1/K for obtaining the average packet latency for a wormhole switching network with finite buffers. Kiasari et al. [24] proposed a performance queuing model using G/G/1. The equations modeled arbitrary topology and channel buffer size, however, the effect of virtual channels was not modeled. Ogras et al. [31] presented a formal approach for NoC performance analysis that relies on a new router description based on FIFO buffers interconnected by switches, and it was based on M/G/1 queueing model. These approaches are summarized in Table 3.1.

Table 3.1: Comparison of Four Performance Models and Their Accuracy

Related work	Queueing model	BS support	VC support	Model inaccuracy (%)	Equations complexity
[5]	M/G/1/k	Yes	Yes	$4 \sim 10$	Complex
[20]	M/G/1/k	Yes	No	10	Complex
[24]	G/G/1	Yes	No	7.5	Simple
[31]	M/G/1	Yes	No	9	Moderately complex

In this dissertation the performance model proposed by Kiasari et al. [24] is used for the following reasons:

- This model assumes a general distribution for packet arrival and service, which is suitable for the bursty nature of GPU traffic [40].
- Equations used in this model are more refined and simpler than other similar works.
- It shows more accuracy compared to other models.
- It supports having different buffer size for each channel of a router in heterogeneous NoCs.

The model, however, suffers from some inaccuracy with full system simulations, as shown after further evaluation in Section 3.5. Nevertheless, following the same approach, the inaccuracy is adjusted, and the support for arbitrary virtual channels per port is added. All the parameters' notations used in the model are described in Table 3.2.

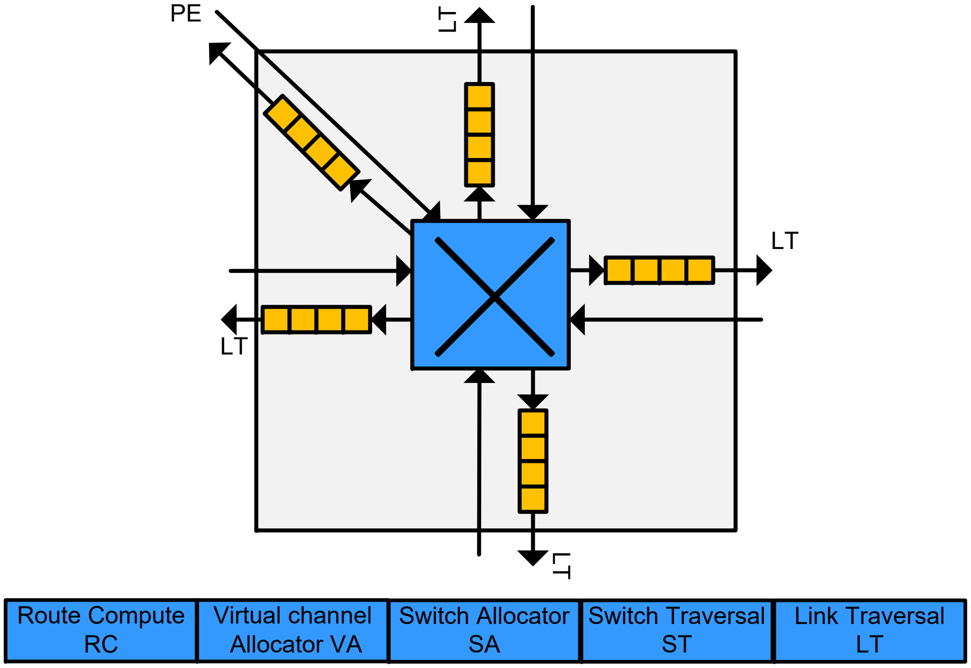


Figure 3.1: Five-ports output-buffered router model with different pipeline-stages.

3.1 Router Model

The targeted design adopts output-only buffered routers, see Figure 3.1. Two different pipeline schemes were explored. Firstly, five pipeline stages; route compute (RC), virtual channel allocation (VCA), switch allocation (SA), switch traversal (ST), and link traversal (LT). Secondly, three pipeline stages; routing and arbitration (RC + VCA), switching and crossbar traversal (SA + ST), and link traversal (LT). Each stage takes one cycle, and the flow control is implemented by monitoring the availability of buffers at each output port in the downstream router before sending.

3.2 Improving Accuracy of the Performance Model

Average message latency of a packet in the network can be given by:

$$L_{NoC} = \sum_{\forall S,D} P^{S \to D} L^{S \to D} \tag{3.1}$$

where $P^{S \to D}$ is the probability that source S sends a packet to destination D, and $L^{S \to D}$ is the packet latency between the source and destination nodes. It consists of two parts, the header latency ($L_h^{S \to D}$) and the body latency (L_b):

$$L^{S \to D} = L_h^{S \to D} + L_b \tag{3.2}$$

The header latency is the time when a packet is created in the source PE until the header flit reaches the destination PE. This includes the number of cycles to inject the flit from source PE to source router t_{inj}, obtain the routing decision t_r, switch the flit within the ports of the router t_s, traverse the link between two routers t_w, eject the flit from destination router to destination PE t_{ej}, and the waiting time W spent at the source and all intermediate routers (M).

$$\begin{aligned} L_h^{S \to D} &= (t_{inj} + t_r + W^S_{inj \to out} + t_s) \\ &+ \sum_{\forall M} (t_w + t_r + W^M_{in \to out} + t_s) \\ &+ (t_w + t_r + W^D_{in \to ej} + t_s + t_{ej}) \end{aligned} \tag{3.3}$$

The body flits will follow the same route; The latency can be found by multiplying the average message size in flits (m), excluding the header flit, by the sum of cycles of switching and link traversing:

$$L_b = (m-1)(t_s + t_w) \tag{3.4}$$

To find the queuing time (wait), [24] models the router based on non-preemptive priority queuing system where each output channel is a server:

$$W_{i\to j}^{N} = \begin{cases} \dfrac{\rho_j^N \left(C_A^2 + C_{S_j^N}^2 \right)}{2\left(\mu_j^N - \lambda_{i\to j}^N \right)} & i = 1 \\ \dfrac{\lambda_j^N \left(C_A^2 + C_{S_j^N}^2 \right)}{2\left(\mu_j^N - \sum_{k=1}^{i-1} \lambda_{k\to j}^N \right)^2} & 2 \le i \le p \end{cases} \tag{3.5}$$

where p is the total number of ports, $\lambda_{i\to j}^N$ is the arrival rate from input port i to output port j of router N, λ_j^N is the arrival rate, μ_j^N is the service rate, and ρ_j^N is the occupation rate of output channel j of router N. C_A^2 is the coefficient of variation of the arrival process to network and $C_{S_j^N}^2$ is the coefficient of variation of service time of output channel j of router N. The occupation rate can be found by:

$$\rho_j^N = \frac{\lambda_j^N}{\mu_j^N} \tag{3.6}$$

To find the arrival rate form inport IC_i to outport OC_j of router N, [24] used:

$$\lambda_{i\to j}^N = \sum_{\forall S,D} \lambda^S \, P^{S\to D} \, R(S \to D, IC_i^N \to OC_j^N) \tag{3.7}$$

where R is a routing function that returns 1 when a packet from source S to destination D passes from inport IC_i^N to outport OC_j^N, 0 otherwise. λ^S is the injection rate of the source router in packet/cycle.

However, using the same probability $P^{S\to D}$ used in (3.1) is not correct. The injected traffic from the source router will be delivered to its destinations, so basically the probability needed is the probability that there is a flow from that specific source S to a destination D among all the source flows ($F_p^{S\to D}$), as in (3.8). On the other hand, $P^{S\to D}$ is the probability that there is a flow or a communication between source S and destination D among all source-destination communications, as in (3.9).

$$F_p^{S\to D} = \frac{C_{S\to D}}{\sum_D C_{S\to D}} \tag{3.8}$$

$$P^{S\to D} = \frac{C_{S\to D}}{\sum_{\forall S,D} C_{S\to D}} \tag{3.9}$$

where $C_{S\to D}$ is the communication rate between source S and destination D. Equation (3.7) is replaced with (3.10):

$$\lambda_{i\to j}^N = \sum_{\forall S,D} \lambda^S\, F_p^{S\to D}\, R(S \to D, IC_i^N \to OC_j^N) \tag{3.10}$$

Then, the arrival rate to outport j of router N:

$$\lambda_j^N = \sum_{i=1}^{p} \lambda_{i\to j}^N \tag{3.11}$$

To compute the first and second moment of the service time of an output channel, [24] gives an index to each output channel that is equal to the maximum distance between its router and other destinations. Then, it calculates the service time for the output channel in an iterative manner starting from the channels with the smallest index (the ejection channels with index = 0) to the other channels in ascending order of their index value. The service time of the ejection channel is:

$$\overline{S}_1^N = t_s + t_w + L_b \tag{3.12}$$

Given the standard deviation of the packet size σ_m, the coefficient of variation (CV^2) of the service time of the ejection channel of router N can be found by:

$$CV^2_{\overline{S}_1^N} = \frac{(t_s + t_w)^2 \, \sigma_m^2}{(\overline{S}_1^N)^2} \tag{3.13}$$

The effect of buffers at the output channel is included in the service time. To illustrate, assume that the service time of the output channels with index x, $\overline{S}_k^N$, is already calculated. To find the service time of the output channel with index $x+1$, of the connected router M, see Figure 3.2, [24] used:

$$\overline{S}_i^M = \sum_{k=1}^{q} P_{j\to k}^N \Big(t_s + t_w + t_r + \overline{S}_k^N + W_{j\to k}^N - B_k^N (t_s + t_w) \Big) \tag{3.14}$$

where B_k^N is the output-buffer size at channel k of router N, and q is the total number of possible output ports of router N. $P_{j\to k}^N$ is the probability that a packet is sent from input port j to output port k of router N and it is given by [24] as:

$$P_{j\to k}^N = \frac{\lambda_{j\to k}^N}{\lambda_k^N} \tag{3.15}$$

where $\lambda_{j\to k}^N$ is the arrival rate from input port j to output port k of router N, and λ_k^N is the arrival rate of output port k of router N.

The problem with (3.14) is that it does not reflect the case when the average packet size is smaller than the buffer size. Therefore, the time spent in the buffer would be larger than it is supposed to be, causing the service time to be negative. Equation (3.14) is replaced by

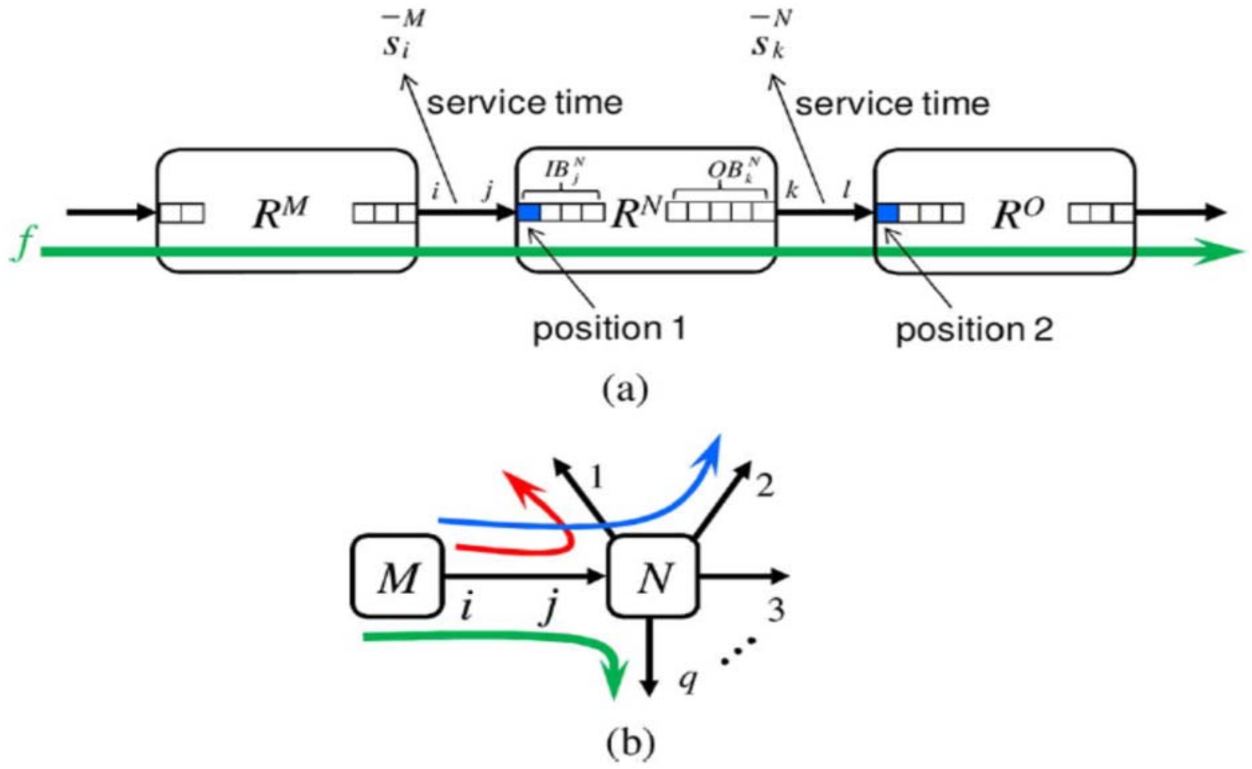


Figure 3.2: ”(a) Passing flow from R^M, R^N and R^O. (b) Some possible path for an entering flow to R^N.” [1]

(3.16), where B factor is added to ensure the effect of arbitrary buffer is correctly included when used later in the optimization.

$$\overline{S}_i^M = \sum_{k=1}^{q} P_{j\to k}^N \Big(t_s + t_w + t_r + \overline{S}_k^N + W_{j\to k}^N - B(t_s + t_w) \Big) \tag{3.16}$$

$$B = \begin{cases} B_k^N & m \geq B_k^N \\ \\ m & otherwise \end{cases} \tag{3.17}$$

Another problem with [24] is the sum of probabilities according to (3.15) would not be equal to 1. Mainly because the incoming traffic to the input port will be distributed among different output ports. When calculating the probability of a flow from input port j to output port

[1] Source: [24]

k, the arrival rate from this input port to the output port relative to all the arrival rates of the input port, not the output port, should be used. Therefore, (3.15) was corrected and replaced by (3.18):

$$P^N_{j\to k} = \frac{\lambda^N_{j\to k}}{\lambda^M_i} \tag{3.18}$$

Where λ^M_i is the arrival rate of output port i of router M (which is the same channel connected to input port j of router N). This correction ensures that the sum of the probabilities from input j to all output k is equal to 1.

Similarly, (3.19) used by [24] to compute the second moment of the output channel service time is replaced by (3.20) as:

$$\overline{(S^M_i)^2} = \sum_{k=1}^{q} P^N_{j\to k}\Big(t_s + t_w + t_r + \overline{S}^N_k + W^N_{j\to k} - B^N_k(t_s + t_w)\Big)^2 \tag{3.19}$$

$$\overline{(S^M_i)^2} = \sum_{k=1}^{q} P^N_{j\to k}\Big(t_s + t_w + t_r + \overline{S}^N_k + W^N_{j\to k} - B(t_s + t_w)\Big)^2 \tag{3.20}$$

Then, the coefficient of variation for the output channel can be calculated as:

$$C^2_{S^M_i} = \frac{\overline{(S^M_i)^2}}{(\overline{S}^M_i)^2} - 1 \tag{3.21}$$

3.3 Adding Virtual Channel to Performance Model

For the case of having virtual channels in a network, the bandwidth of the physical channel is shared among the virtual channels. The average degree of virtual channel multiplexing for every pair of source and destination need to be calculated and included in the average packet latency. If there are V virtual channels that share the bandwidth of a physical channel, all the incoming traffic rate will be uniformly distributed among them. Therefore, the incoming traffic rate into an output channel j of router N that has V virtual channels:

$$\lambda_{j,vc}^{N} = \frac{\lambda_j^N}{V_j^N} \tag{3.22}$$

For each pair of source S and destination D, the average message latency should be scaled by the average degree of virtual channel multiplexing, as in [15]. So, (3.2) is replaced by (3.23) as:

$$L^{S \to D} = (L_h^{S \to D} + L_b) \times \overline{V}_{S \to D} \tag{3.23}$$

where $\overline{V}_{S \to D}$ is the average of virtual channel multiplexing of all intermediate channels, therefore, it can be calculated as:

$$\overline{V}_{S \to D} = \frac{\sum_{i=1}^{H^{S \to D}} \overline{V}_{(a_i,b_i)}}{H^{S \to D}} \tag{3.24}$$

where $H^{S \to D}$ is the hop count of the path between source S and destination D, and $\overline{V}_{(a_i,b_i)}$ is the average virtual channel multiplexing degree of channel (a_i, b_i) at the *i-th* hop of the path between S and D.

To calculate $\overline{V}_{(a_i,b_i)}$, based on the analysis done in [15]:

$$\overline{V}_{(a_i,b_i)} = \frac{\sum_{v=1}^{V} \left(v^2 P_{(a_i,b_i)}(v) \right)}{\sum_{v=1}^{V} \left(v P_{(a_i,b_i)}(v) \right)} \tag{3.25}$$

where $P_{(a_i,b_i)}(v)$ is the probability of having v busy virtual channels at physical channel (a_i, b_i). To find this probability, all flows that use the physical channel (a_i, b_i) should be determined.

Let $F_{(a_i,b_i)} = \{F_1, F_2, \cdots, F_n\}$ denote all flows that use the physical channel (a_i, b_i) to deliver a message from any source to any destination. The probability that exactly v virtual channels are busy is the probability that v flows of set $F_{(a_i,b_i)}$ are active, and the others are not. Therefore, it can be given by:

$$P_{(a_i,b_i)}(v) = \begin{cases} \sum\limits_{\forall F^v_{(a,b)}} \Big[\prod\limits_{i \in F^v_{(a,b)}} C_{F_i:S_i \to D_i} \times \prod\limits_{i \notin F^v_{(a,b)}} (1 - C_{F_i:S_i \to D_i}) \Big] & v \leq n \\ \\ 0 & v > n \end{cases} \tag{3.26}$$

where $F^v_{(a,b)}$ is any member of the exponential set of set $F_{(a_i,b_i)}$ with v elements, and $C_{F_i:S_i \to D_i}$ is the communication rate between source S and destination D of Flow i.

Based on this analysis, (3.5) and (3.18) are replaced by (3.27) and (3.28), respectively, to support virtual channels:

$$W^N_{i \to j} = \begin{cases} \dfrac{\rho^N_j \Big(C^2_A + C^2_{S^N_j} \Big)}{2 \Big(\mu^N_j - (\lambda^N_{i \to j} / V^N_j) \Big)} & i = 1 \\ \dfrac{\lambda^N_j \Big(C^2_A + C^2_{S^N_j} \Big)}{2 \Big(\mu^N_j - (\sum_{k=1}^{i-1} \lambda^N_{k \to j} / V^N_j) \Big)^2} & 2 \leq i \leq p \end{cases} \tag{3.27}$$

$$P^N_{j \to k} = \Big(\frac{\lambda^N_{j \to k}}{\lambda^M_j} \Big) \Big/ V^M_j \tag{3.28}$$

where V^N_j is the number of virtual channels of physical channel j of router N.

3.4 Adding Heterogeneous Bandwidth Support

The link bandwidth determines the rate at which the data are transferred. It can be controlled by two parameters: the link width and the link latency. As explained later in Chapter 6, the simulator used to evaluate the proposed NoC design methods does not support heterogeneous bandwidth in terms of different link widths. Alternatively, the heterogeneous bandwidth of the links is represented as a heterogeneous links latency, that is a heterogeneous number of cycles to traverse a flit. To include this in the performance model some equations need to be changed.

Equation (3.3) need to be replaced by:

$$\begin{aligned} L_h^{S\to D} &= (t_{inj} + t_r + W_{inj\to out}^{S} + t_s) \\ &+ \sum_{\forall M}(t_{w^l} + t_r + W_{in\to out}^{M} + t_s) \\ &+ (t_{w^l} + t_r + W_{in\to ej}^{D} + t_s + t_{ej}) \end{aligned} \tag{3.29}$$

where t_{w^l} is the latency of the link attached to the input port of the respective router (M or D).

Also, (3.16) that is used to find the service time of the output channel needs to be replaced by:

$$\overline{S}_i^M = \sum_{k=1}^{q} P_{j\to k}^{N}\Big(t_s + t_{w^i} + t_r + \overline{S}_k^N + W_{j\to k}^{N} - B(t_s + t_{w^i})\Big) \tag{3.30}$$

Where t_{w^i} is the link latency of channel i. Similarly, the second moment of service time as in (3.20) is replaced by:

$$\overline{(S_i^M)^2} = \sum_{k=1}^{q} P_{j\to k}^{N}\Big(t_s + t_{w^i} + t_r + \overline{S}_k^N + W_{j\to k}^{N} - B(t_s + t_{w^i})\Big)^2 \tag{3.31}$$

Table 3.2: Performance Model Parameters' Notations

Notation	Description
B_j^N	Buffer size of outport j of router N in flits
C_A^2	Coefficient of variation of the arrival process to the network
$C_{S_j^N}^2$	Coefficient of variation of service time of outport j of router N
$C_{S\rightarrow D}$	Communication rate between source S and destination D
$C_{F_i:S_i\rightarrow D_i}$	Communication rate between source S and destination D of Flow i
$H^{S\rightarrow D}$	Hop count of the path between source S and destination D
$F_p^{S\rightarrow D}$	Flow probability from source S to destination D
$\lambda_{i\rightarrow j}^N$	Arrival rate from inport i to outport j of router N
$\lambda_{j,vc}^N$	Arrival rate to the virtual channels of outport j of router N
λ_j^N	Arrival rate of outport j of router N
λ^N	Injection rate of router N in packet/cycle
$L^{S\rightarrow D}$	Latency between source S and destination D
$L_h^{S\rightarrow D}$	Header latency from source S to destination D
L_b	Body latency
L_{NoC}	Average packet latency
m	Average packet size in flits
μ_j^N	Service rate of outport j of router N
$P_{j\rightarrow k}^N$	Probability of a packet sent from inport j to outport k of router N
$P^{S\rightarrow D}$	Probability that source S sends a packet to destination D
$P_{(a_i,b_i)}(v)$	Probability of having v busy virtual channels at physical channel (a_i, b_i)
σ_m	Standard deviation of packet size
ρ_j^N	Occupation rate of outport j of router N
$\overline{S}_j^N$	First moment of service time of outport j of router N
$\overline{(S_j^N)^2}$	Second moment of service time of outport j of router N
t_{ej}	Number of cycles to eject a flit from destination router to its PE
t_{inj}	Number of cycles to inject a flit by source PE to its router
t_r	Number of cycles to obtain routing decision
t_s	Number of cycles to switch a flit between router's ports
t_w	Number of cycles to traverse a flit between two routers
t_{w^i}	Number of cycles to traverse a flit between two routers on link i
$W_{i\rightarrow j}^N$	Waiting time from inport i to outport j of router N
$\overline{V}_{S\rightarrow D}$	Average degree of virtual channel multiplexing between S and D
$\overline{V}_{(a_i,b_i)}$	Average virtual channel multiplexing degree of channel (a_i, b_i)

3.5 Model Accuracy

The proposed model was evaluated using the Garnet Network test in gem5-gpu [32]. Six experiments were conducted. The first experiment aims to evaluate the improved model after adjusting the equations of [24]. The second experiment aims to establish the accuracy of the model after adding the support for virtual channels. The third experiment aims at establishing the accuracy of the model for handling heterogeneous buffer and virtual channels. The fourth experiment aims to validate the use of link latency to control the bandwidth instead of the link width. The purpose of the fifth experiment is to establish the accuracy of the model after adding the bandwidth support. The sixth and last experiment aims to establish the accuracy of the final model with buffer, virtual channel, and bandwidth support for heterogeneous configurations.

For the first two experiments and the fourth experiment, uniform synthetic traffic was injected into a 2D mesh of 16 CPUs with homogeneous buffer, virtual channels, and links bandwidth for a fixed number of cycles. The test was repeated for different packet injection rates (same for all the nodes). For the third experiment, real traffic trace was obtained from running workloads of a combination of Parsec and Rodinia benchmarks, see Table 6.4, on arbitrary buffer and virtual channels configurations. A similar approach was used to conduct the fifth experiment but using homogeneous buffer, virtual channels, and bandwidth. While the last experiment use the real traffic on arbitrary buffer, virtual channels, and links latencies (bandwidths) configurations.

3.5.1 Evaluating the Improved Buffer Model

By setting the number of virtual channels to one with four buffers, the average latency of the proposed model was compared with the simulator and the original unmodified model of [24] for three different average packet sizes (m). As shown in Figure 3.3, the results of the proposed model is near to the simulated result and saturate a little further than the simulator. The results of [24] on the other hand, is very far and saturate at higher injection rates. The average percentage of error in the proposed model is 12.9%, while [24] is 22.6%.

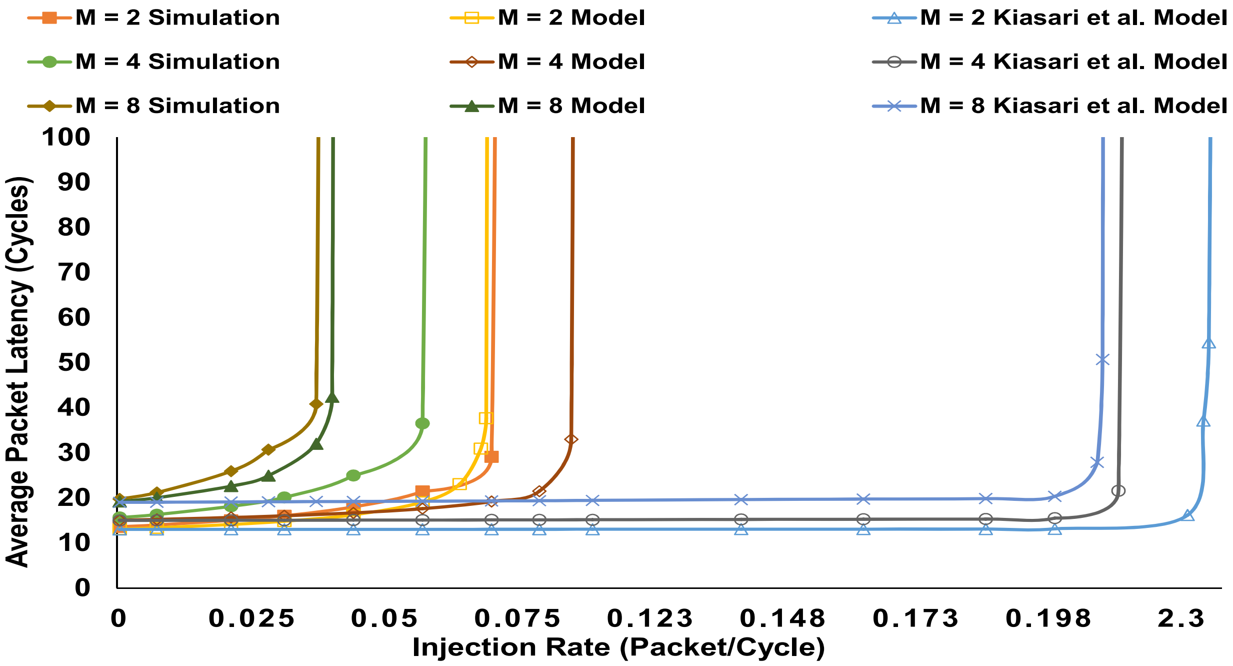


Figure 3.3: Comparison of average packet latency against simulation and [24] model for homogeneous NoC with 1 VC and 4 BS under different average message (packet) sizes (M).

3.5.2 Evaluating the Added Virtual Channels

By fixing the buffer size to four, the average latency of the proposed model was compared with the simulator using a different number of virtual channels. As seen in Figure 3.4, the accuracy of the model is improved even further with an average percentage of error of 7.7%.

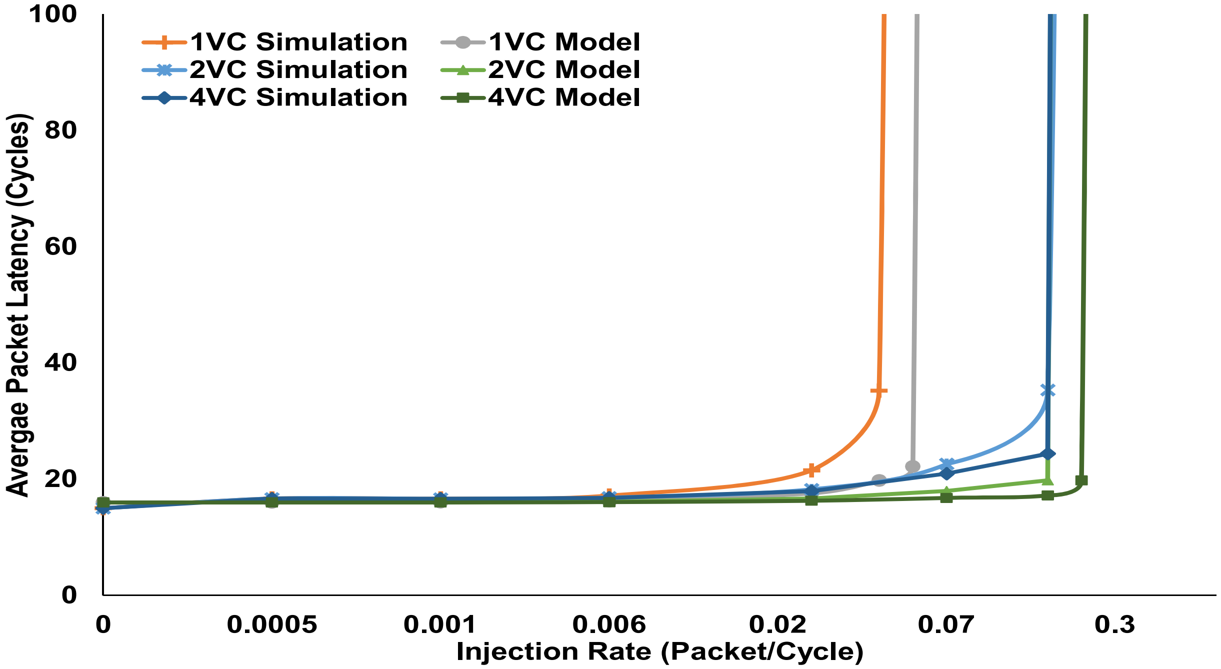


Figure 3.4: Average packet latency of proposed model against simulation for homogeneous NoC with 4 BS under different number of VCs.

3.5.3 Evaluating the Heterogeneity of the Model (BS and VC)

After running the different workloads on gem5-gpu under different buffer and virtual channels configurations, the output real traffic trace of each arbitrary configuration was fed to the model to obtain the average packet latency. Figure 3.5 compares the average packet latency obtained using the simulator and the model for 24 random NoC configurations. The average percentage of error is 5%.

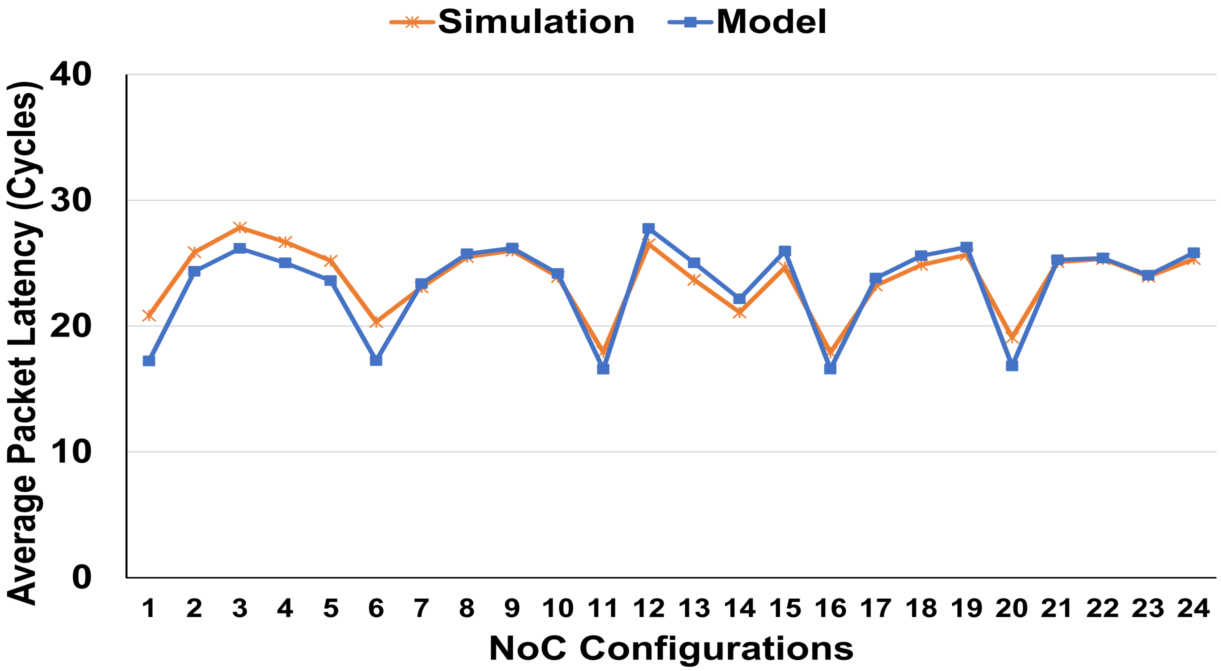


Figure 3.5: Average packet latency of proposed model (BS, VC) against simulation of real traffic for different heterogeneous NoC configurations.

3.5.4 Comparing Link Latency and Link Width

By fixing the virtual channels to 1 with 16 buffers, the network test was run on the simulator for two links settings, and the average packet latency was obtained for different injection rates. The first experiment was conducted after setting the width of the links to 16B and the latency of the links to 1 cycle. The same experiment was repeated for links' width of 32B and links' latency of 2 cycles. The results of both experiments are compared in Figure 3.6. As the injection rate increases, the difference between the two configurations increases, but the average error is about 9.92%.

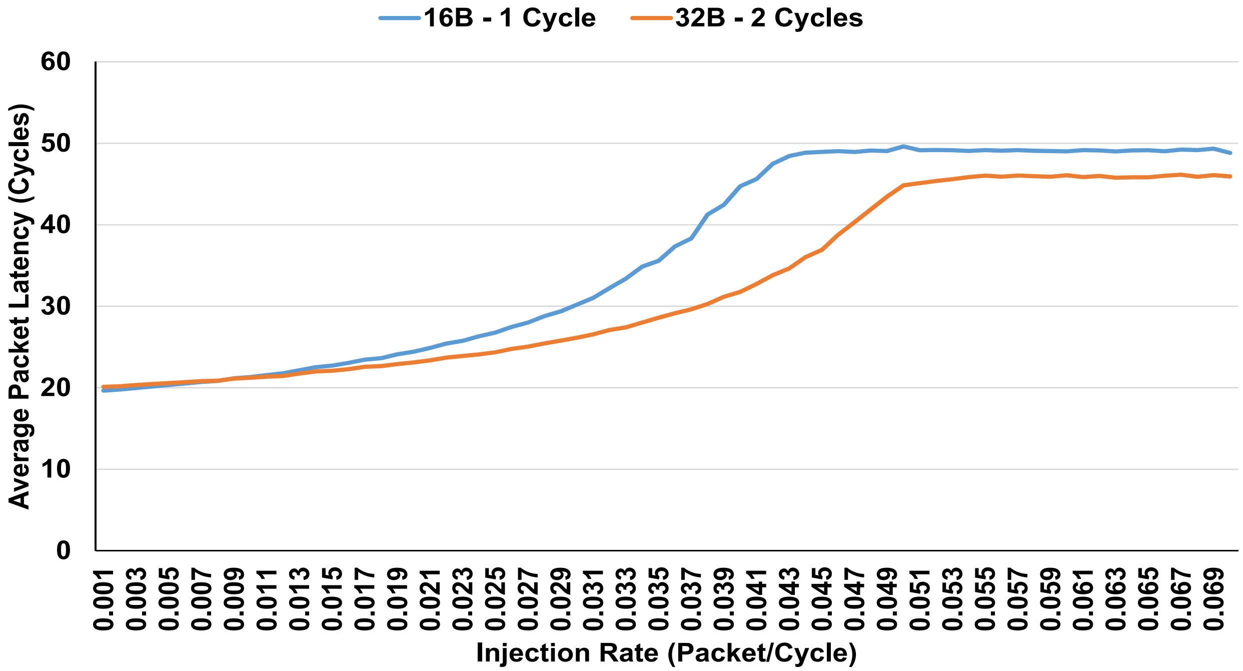


Figure 3.6: Comparison of average packet latency of model and simulation for homogeneous NoC with 1 VC and 16 BS under different links' settings.

3.5.5 Evaluating the Added Bandwidth Support

Twenty different workloads were run on gem5-gpu using homogeneous four virtual channels each with eight buffers for two different homogeneous links settings. Firstly, a link width of 16B and link latency of 1 cycle. Secondly, a link width of 32B and link latency of 2 cycles. The output real traffic trace of each experiment was fed to the model, and the average packet latency computed by the model was compared to the average packet latency obtained from the simulator as in Figure 3.7. A general notice is that the trend of the second link settings (32B, 2 Cycles) of model and simulator is more accurate than the first link setting (16B, 1 Cycle). When comparing the model results of the second link setting (32B, 2 Cycles) to the simulator results of the first link setting (16B, 1 Cycles) the average error rate is about 9%. This indicates that using link latency as a way to control the link bandwidth is justifiable since it is near to the results of changing the link width in simulation.

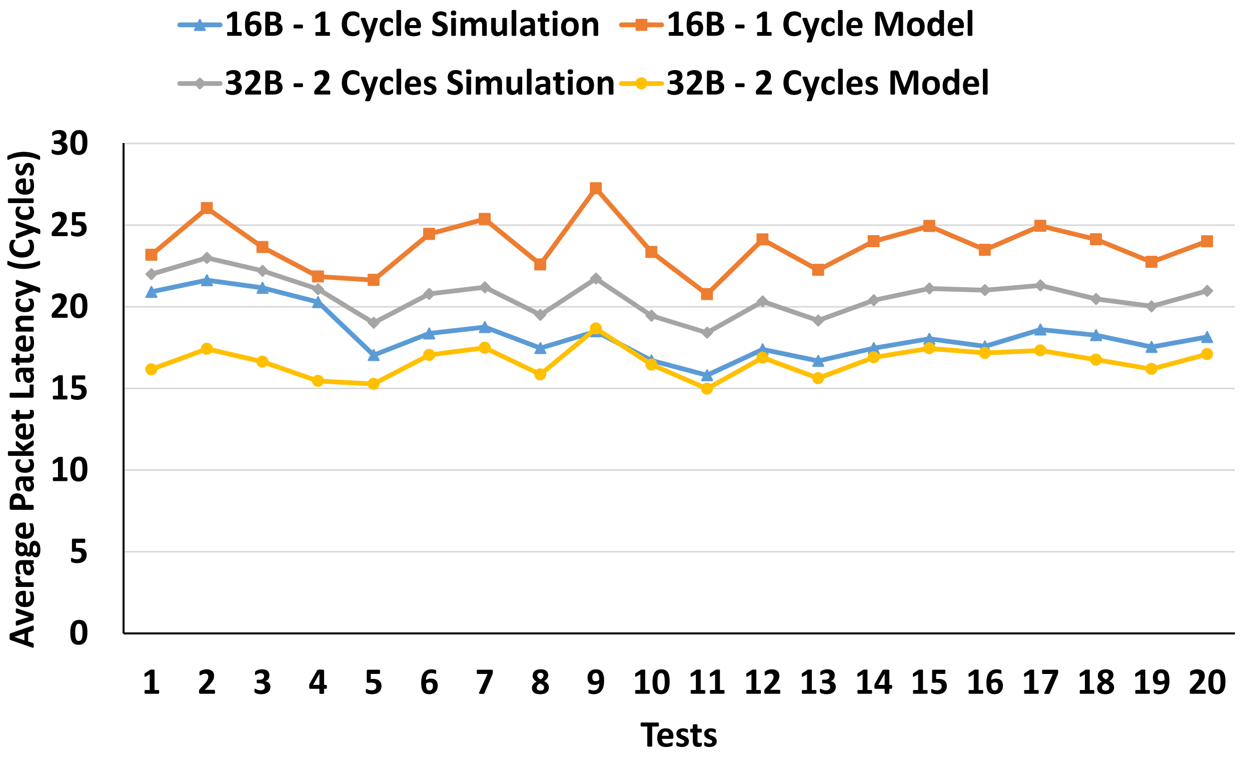


Figure 3.7: Comparison of model average packet latency against simulation of real traffic for homogeneous NoC with 4 VC and 8 BS under two different links' settings.

3.5.6 Evaluating the Heterogeneity of the Final Model

Different workloads were run on gem5-gpu under different buffer, virtual channels, and links configurations, and the output real traffic trace of each arbitrary configuration was fed to the model to obtain the average packet latency. Figure 3.8 compares the average packet latency obtained using the simulator and the model for 40 random NoC configurations. The average percentage of error is 25%.

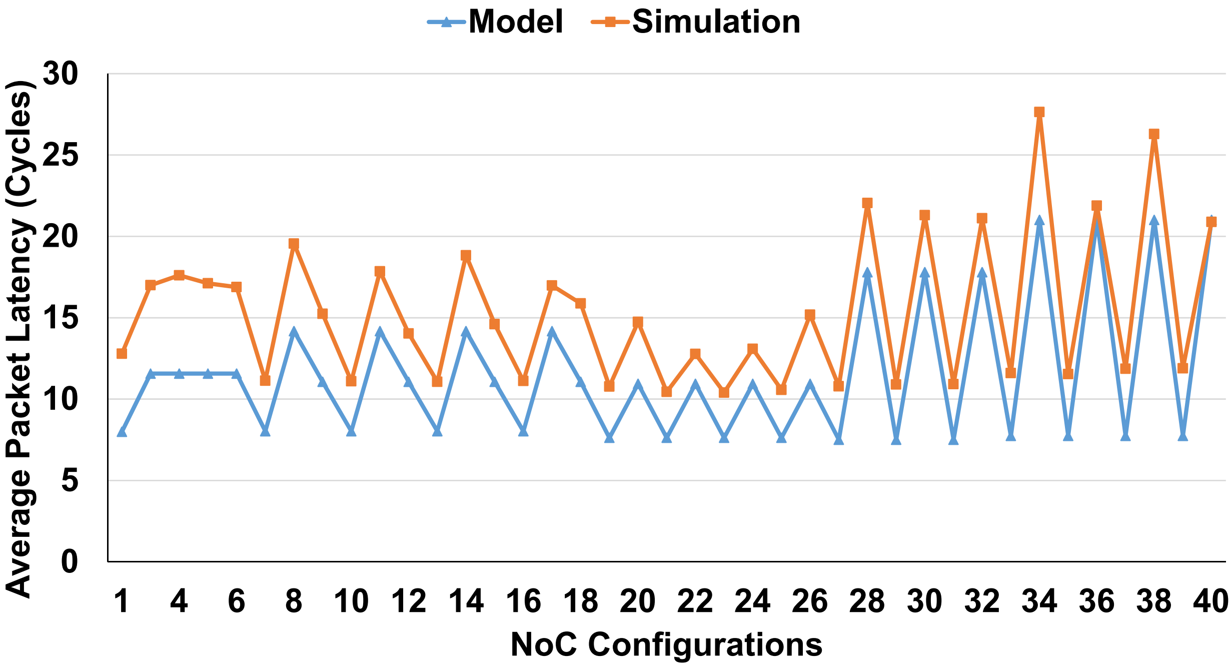


Figure 3.8: Average packet latency of the final proposed model against simulation of real traffic for different heterogeneous NoC configurations.

Chapter 4

Power Model

The model has been proposed in [4] and all model's parameters are described in Table 4.1. The total power consumption of the NoC includes the power consumed in the routers and the links. Following the analysis in [5], the power consumed in a router N consists of the power consumed in the routing and arbitration unit, the power consumed in the crossbars, and the total power consumed in the router's links as:

$$P_N^{Router} = P_N^{R\&A} + P_N^{XB} + \sum_{j=1}^{p} P_{N,j}^{TotalLink} \tag{4.1}$$

The power consumed in the arbitration and routing unit is:

$$P_N^{R\&A} = P_{Header}^{R\&A} \sum_{j=1}^{p} \lambda_j^N \tag{4.2}$$

where $P_{Header}^{R\&A}$ is the power consumed in arbitrating and routing a header flit.

The crossbar power consumption is:

$$P_N^{XB} = P_{bit}^{XB}\, K\, m \sum_{j=1}^{p} (\lambda_j^N)^2 \tag{4.3}$$

where P_{bit}^{XB} is the dynamic power consumed when a flit traverses the crossbar and $K\, m$ is the average size of the packets in bits.

The total power consumed by the router's link consists of the power consumed by the link, the dynamic, and the leakage power of the buffers:

$$P_{N,j}^{TotalLink} = P_{N,j}^{Link} + P_{N,j}^{BufferD} + P_{N,j}^{BufferL} \tag{4.4}$$

To find the link power of an L-millimeter channel j with W bits width connected to a router N that has f_{clk} frequency and V_{DD} supply voltage:

$$P_{N,j}^{Link} = \frac{1}{2}\, \lambda_j^N\, m \left(\alpha_L\, W\, C_L^0 + \alpha_C\, (W-1)\, C_C^0\right) L\, f_{clk}\, V_{DD}^2 \tag{4.5}$$

where α_L and α_C are the probabilities that different bit values cross over a single and adjacent links, respectively. C_L^0 and C_C^0 are the link and the crosstalk capacities per millimeter, respectively.

The dynamic power of the buffers is the power consumed in reading/writing a flit from/to the buffer calculated as:

$$P_{N,j}^{BufferD} = m\, k\, V_j^N\, (\lambda_{j,vc}^N\, P_{bit}^W + \mu_j^N\, P_{bit}^R + Q_j^N\, P_{bit}^{clk}) \tag{4.6}$$

where P_{bit}^W and P_{bit}^R are the power consumed in writing and reading a bit to/from the buffer, respectively. P_{bit}^{clk} is the average power consumed when a one-bit memory element receives a clock switch. Q_j^N is the average number of packets in the output buffer j of router N and calculated based on Little's Theorem as:

$$Q_j^N = \lambda_j^N \, W_j^N \tag{4.7}$$

where W_j^N is the waiting time of outport j of router N:

$$W_j^N = \sum_{i=1}^{p} W_{i \to j}^N \tag{4.8}$$

The leakage power of the output buffer j of router N:

$$P_{N,j}^{BufferL} = B_j^N \, W \, P_{bit}^L \tag{4.9}$$

where B_j^N is the buffer size in flits, and P_{bit}^L is the average leakage power of one-bit memory element. Then, the total power consumption of the NoC can be found by:

$$P_{NoC} = \sum_R P_R^{Router} \tag{4.10}$$

4.1 Adding Heterogeneous Bandwidth Support

Some equations need to be changed to include the effect of heterogeneous links bandwidth. The dynamic link power of (4.5) is replaced by:

$$P_{N,j}^{Link} = \frac{1}{2} \, \lambda_j^N \, m \left(\alpha_L \, W_j \, C_L^0 + \alpha_C \, (W_j - 1) \, C_C^0 \right) L \, f_{clk} \, V_{DD}^2 \tag{4.11}$$

Where W_j is the width in bits of channel j. The leakage power of the buffer as in (4.9) is replaced by:

$$P_{N,j}^{BufferL} = B_j^N \, W_j \, P_{bit}^L \tag{4.12}$$

Table 4.1: Power Model Parameters' Notations

Notation	Description
α_C	Probability that different bit values cross over adjacent links
α_L	Probability that different bit values cross over single links
B_j^N	Size of buffer of outport j of router N in flits
C_C^0	Crosstalk capacity per millimeter
C_L^0	Link capacity per millimeter
f_{clk}	Frequency in hertz
K	Size of a flit in bits
L	Link length in millimeters
λ_j^N	Arrival rate of outport j of router N
$\lambda_{j,vc}^N$	Arrival rate to the virtual channels of outport j of router N
m	Average packet size in flits
μ_j^N	Service rate of outport j of router N
P_{bit}^{clk}	Average power when a one-bit memory element receives a clock switch
P_{bit}^{L}	Average leakage power of one-bit memory element
P_{bit}^{R}	Power consumed in reading a bit from the buffer
P_{bit}^{W}	Power consumed in writing a bit to the buffer
P_{bit}^{XB}	Dynamic power consumed when a flit traverses the crossbar
$P_{Header}^{R\&A}$	Power consumed in arbitrating and routing a header flit
$P_{N,j}^{BufferD}$	Dynamic power consumed in buffers of outport j of router N
$P_{N,j}^{BufferL}$	Leakage power consumed in buffers of outport j of router N
$P_{N,j}^{Link}$	Dynamic power of link j of router N
$P_{N,j}^{TotalLink}$	Total power consumed in link j of router N
P_{NoC}	Total power of NoC
$P_N^{R\&A}$	Power consumed in the arbitration and routing unit of router N
P_N^{Router}	Power consumed in a router N
P_N^{XB}	Power consumed in the crossbars of router N
Q_j^N	Average number of packets in the buffer of outport j of router N
V_{DD}	Supply Voltage in volts
V_j^N	Number of virtual channels at outport j of router N
W	Link width in bits
W_j	Width of link j in bits
$W_{i\to j}^N$	Waiting time from inport i to outport j of router N
W_j^N	Waiting time of output port j of router N

Chapter 5

Optimization Methodology

The problem of designing a heterogeneous NoC involves solving different sub-problems: placing the PEs within the NoC; assigning the buffer size and number of virtual channels for each port of each router in the NoC; choosing the bandwidth for each link in the NoC. Each of these sub-problems has a large and complex design space. Combining them in one-problem enlarges the design space even further.

The inputs to the problem are: a set of PEs, routers connected in 2D mesh style NoC, a communication rate matrix between the PEs, and the injection rate of each PE. The aim is to: (1) map the PEs into the routers of the NoC, (2) configure the number of virtual channels for each router's port, (3) configure the buffer size for each router's port, and (4) configure the bandwidth for each link of the NoC. This design problem can be solved to satisfy one of many objectives, such as performance and power.

Many optimization techniques are available to solve complex problems with large design space. Some are efficient for single objective problems, and others can handle multi-objective problems. The optimization techniques can be classified into single-solution based and population-based, depending on the number of solutions that they work with [18].

Single-solution based techniques focus on modifying and improving a single solution. Different techniques vary in the way they modify the solution and accept a new one. Descent method or hill climbing is one of the simplest methods. It starts with a random solution and then either select the first feasible solution in the neighborhood that improves the objective function of the current solution or the best feasible solution of the entire neighborhood. The main drawback of this technique is that it can get stuck in a local optimum. Two other techniques that can escape local optimum are Simulated Annealing (SA) and Tabu Search (TS). SA is inspired by the annealing process, which consists of melting metal at high temperature to be then cooled to a stable condition. It starts with a random solution and then accepts a new solution based on a probability function of the temperature parameter $exp(-\Delta f/T)$. This probability makes it possible to accept a worse solution allowing exploration of the search space. The temperature is updated each iteration following a decreasing function making exploitation of the search space favorable as iterations increase. TS is based on the principle of human memory, and it memorizes previously encountered solutions by storing them in a ”tabu” list. It starts with a random solution and accepts the best solution of the neighborhood as long as it is not in the tabu list. It is possible to move to a worse solution, escaping local optimum. Moreover, prohibiting already explored best solutions avoids falling back into local optima.

Population-based techniques work with a set of solutions and improve these solutions usually using population characteristics. Population-based techniques can be further classified into evolutionary algorithms and swarm intelligence based algorithms. Evolutionary algorithms are inspired by biological evolution, which is based on the natural selection and the modification of some genetic characteristics according to a certain probability. There are many evolutionary algorithms such as Genetic Algorithm, Differential Evolution, and Bayesian approach. Swarm intelligence based techniques are inspired by natural phenomena and the behavior of a group of agents that communicate with each other and interact with their environment to survive. Examples of swarm intelligence based techniques are Particle Swarm

Optimization and Ant Colony Optimization.

Single-solution based techniques and population-based techniques differ in their way of navigating the search space. Single-solution methods focus more on exploitation, that is visiting solutions in the neighborhood, with a little exploration. Population-based methods, on the other hand, allows more exploration of the search space by working with many solutions at the same time and visiting new regions of the search space. Both exploration and exploitation are necessary to find the optimal solution. Evolutionary algorithms are known to have a good balance between exploration and exploitation [12].

This dissertation adopts the Genetic Algorithm (GA) to find a heterogeneous NoC design that optimizes a single objective; performance of the NoC. GA [19] is an evolutionary algorithm that falls under the class of guided random search techniques and works with a population of chromosomes. Each chromosome represents a possible solution to the problem and is evaluated according to a fitness function that gives the quality of the solution. To evolve, GA applies different evolutionary operators such as selection, crossover, and mutation. Since GA is based on the survival of the fittest theory, chromosomes with better fitness most probably survive and evolve to even better chromosomes. A general flow of GA is shown in Figure 5.1. The main stages of GA are:

- **Initialization:** Generate a population of chromosomes randomly.
- **Evaluation:** Assign a fitness according to the objective for each chromosome.
- **Termination:** Check the termination criteria; this can be a specific number of generations or if the solutions stopped improving.
- **Selection:** A selection method is applied to select candidate parents. There are many variations of the selection operator, such as roulette wheel selection, tournament selection, rank selection, and random selection. The selection methods select the parents

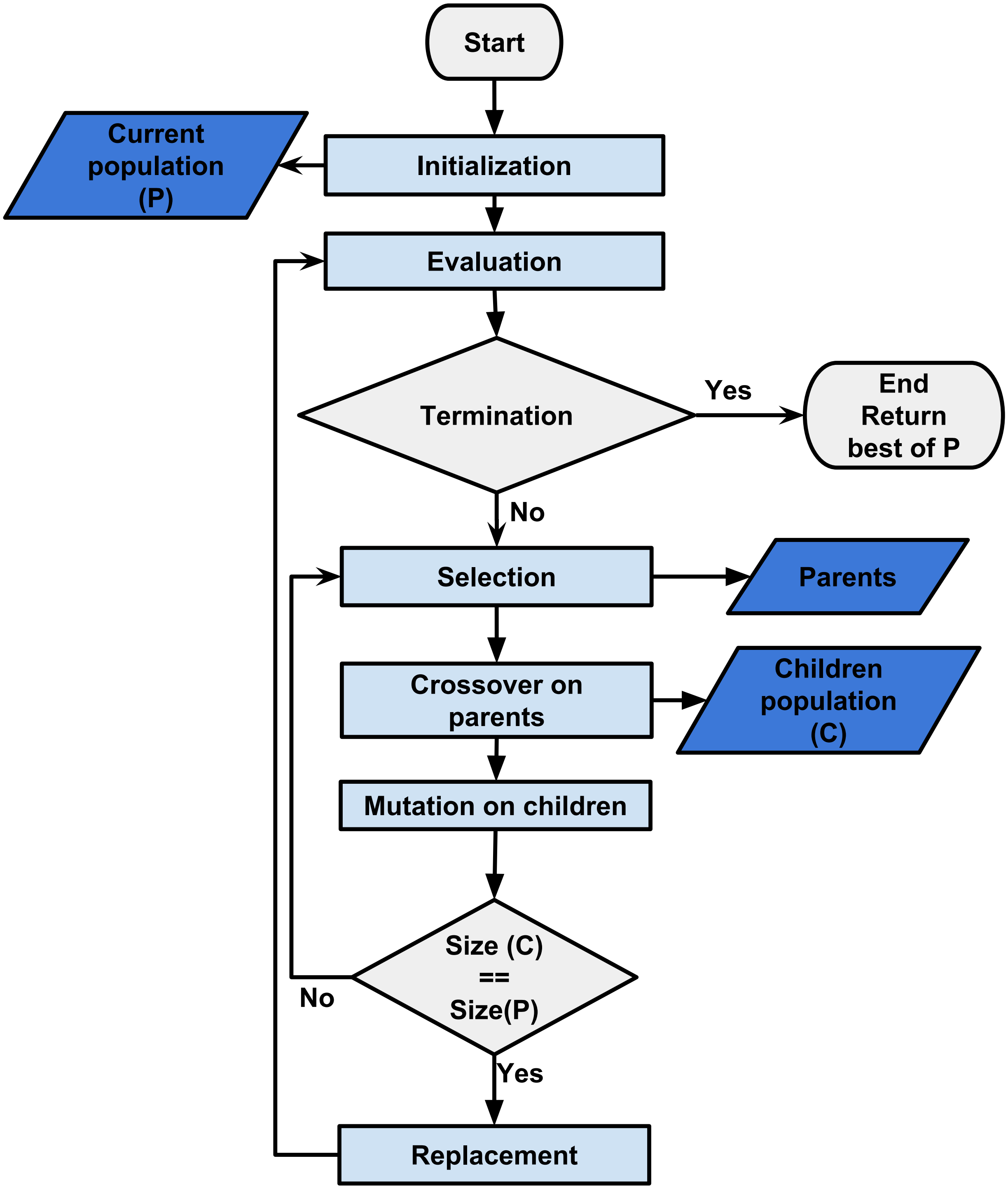


Figure 5.1: A general flow of Genetic Algorithm showing the evolution process.

according to their fitness, based on the idea that good parents most probably generate better children.

- **Crossover:** The selected parents are mate by crossing over their genes; the purpose of crossover is to exploit the good regions of the search space. There are many variations of crossover that can be applied to a traditional or ordered chromosome. Some examples of traditional crossover include one-point crossover, k-point crossover, and uniform crossover. Examples of ordered crossover methods include partially mapped crossover (PMX), cycle crossover, and position-based operator.

- **Mutation:** This operator is used to explore the search space further and avoid local optimum. The mutation works on a single solution and changes it by randomly altering one or more of its genes. Some mutation operators include random change, swap, scramble, and inversion.

- **Replacement:** This phase will generate the next generation by selecting the survivors among the parent and children populations. Two widely strategies are used. Firstly, age-based selection, where the oldest members of the populations are dropped. Secondly, fitness-based selection, where any of the selection methods can be used to select the survivors among the parent and children populations.

Multi-objective optimization methodologies that are based on genetic algorithms can be classified according to the way of handling the fitness function into three main categories [25]: weighted sum approaches, altering objective functions approaches, and Pareto-ranking approaches. The weighted sum approaches assign a weight w_i to each normalized objective function $f'_i(x)$ to convert the problem into a single objective problem as follows: $min\ f = w_1 f'_1(x) + w_2 f'_2(x) + ... + w_n f'_n(x)$. This method is simple but choosing the weights is a challenge. The altering objective approaches use only a single objective randomly chosen at the parent selection phase. This method is straightforward, but the population tends to converge to solutions that are excellent in one objective and bad at others. The third approach, explicitly utilize the concept of Pareto dominance in fitness evaluation or the selection phase. The solution is called dominate if it is better in at least one of the objective functions and is not worse in any of the objective functions. The advantage of this approach is that it provides the designer with a set of solutions ”Pareto-set” to further investigate and choose the appropriate design from it.

This dissertation adopts Strength Pareto Evolutionary Algorithm2 (SPEA2) to find a heterogeneous NoC that optimizes two objectives; performance and power of NoC. Two versions of SPEA2 are proposed. Firstly, SPEA2-based method to find an NoC design while solving three sub-problems (PE mapping, BS, and VC configurations). Secondly, a method to get an NoC design that solves four sub-problems (PE mapping, BS, VC, and BW configurations), SPEA2-BW.

SPEA2 [44] is an evolutionary algorithm that is efficient for finding the Pareto optimal set for multi-objective problems. It is based on SPEA [45] and works with two populations each with a fixed size, a regular population of solutions (chromosomes) and an archive which is an external set that keeps the non-dominated solutions. A solution is a non-dominated when there is no other feasible solution better than it in some objective function without worsening other objective functions. Environmental selection is applied to the combined regular and archive populations to select the new archive. Reproduction operators, including selection, crossover, and mutation, are applied to the new archive to generate the children population. A general flow of SPEA2 in shown in Figure 5.2. The main stages of SPEA2 are:

- **Initialization:** Generate a regular population of chromosomes randomly and an empty archive.
- **Fitness assignment:** Assign a fitness value for each chromosome taking into account both dominating and dominated solutions. The objective functions determine the

domination.

- **Environmental selection:** This stage is responsible for updating the archive. It starts by copying the non-dominated solutions from the regular population and the current archive. Since the size of the archive is constant, if the number of non-dominated solutions is less than the archive size, it will be filled with the best dominated solutions. On the other hand, if the number of non-dominated solution exceeds the archive size, a truncation operation is applied to remove the non-dominated solutions based on their k-th distance. This truncation operation prevents boundary solutions from being removed.

- **Termination:** If the termination criterion, such as a maximum number of generations, is met, the algorithm will terminate.

- **Selection:** Similar to GA, the selection is applied to choose the parents, but in SPEA2 selection is applied to the archive. That is non-dominated solutions are more probably generate better children.

- **Crossover:** Crossover is applied to mix the genetic materials of the selected parents and exploit the search space.

- **Mutation:** Mutation is applied to introduce randomness to the genes of a single solution and allow further exploration of the search space.

- **Replacement:** This phase will replace the regular population with the generated children and copy the current archive to the next generation.

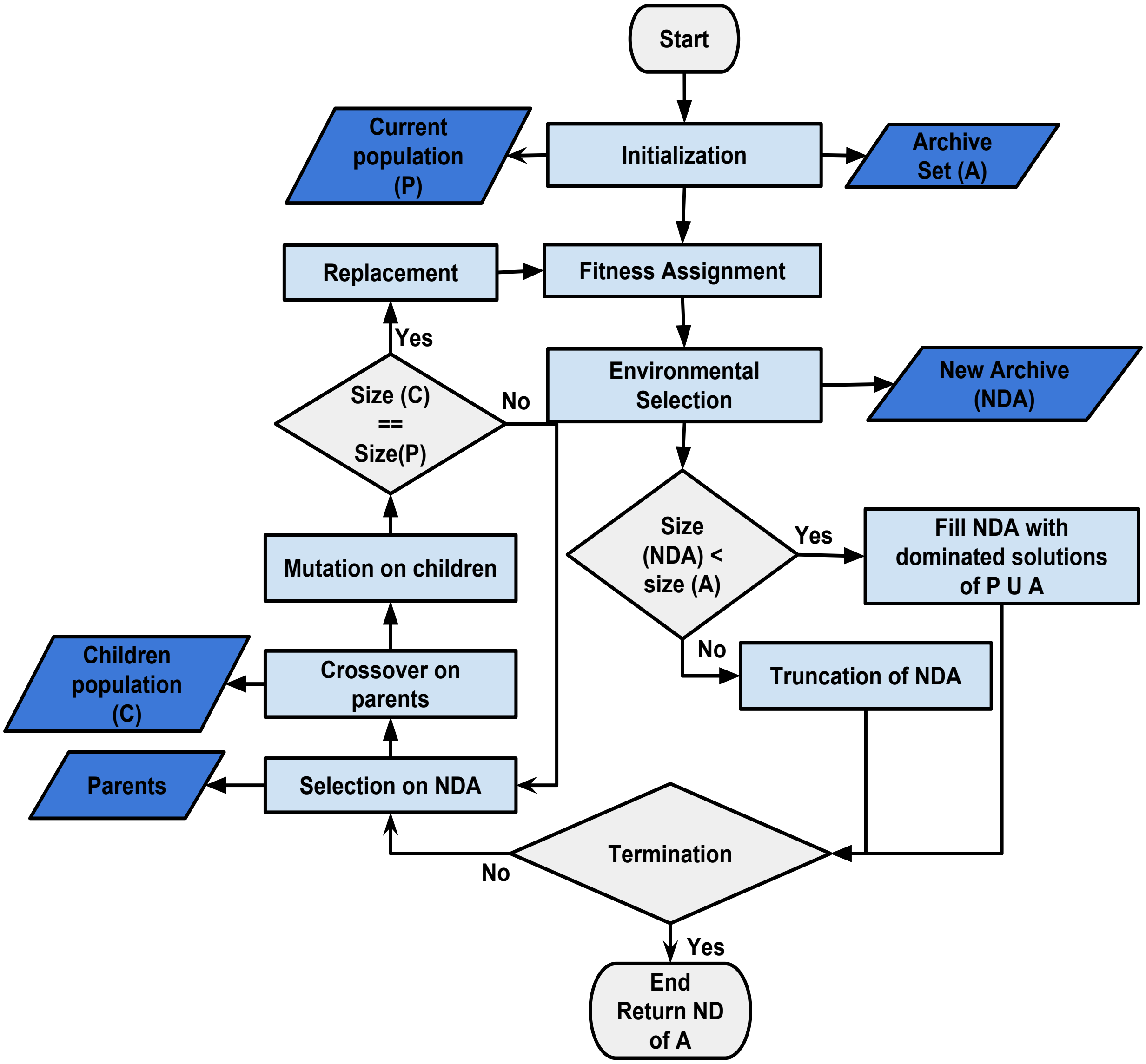


Figure 5.2: A general flow of SPEA2 showing the evolution process.

5.1 GA NoC Design for Three Sub-Problems

The objective of GA is to minimize network delay (average packet latency). Since the search space is huge, depending on simulation to get the average packet latency of the design, even though it is most accurate, is not possible. Instead, a queueing-theory-based model to estimate the average packet latency explained in Chapter 3 is used as the evaluation function where the dynamic part of the network caused by traffic is included. The buffer size and number of virtual channels are bounded:

Minimize

$$\sum_{\forall S,D} P^{S\rightarrow D}\, L^{S\rightarrow D}$$

Subject to:

$$1 \leq B_p^R \leq B_{MAX} \qquad \forall\ Router\ R, outport\ p$$

$$2 \leq V_p^R \leq V_{MAX} \qquad \forall\ Router\ R, outport\ p$$

The proposed GA to find the best NoC configuration among many generated populations is presented in Algorithm 1. The algorithm, takes the network dimensions, maximum buffer size per port, maximum virtual channels per port, and the set of PEs to be placed along with their injection rates and communication rates as inputs. The output of this algorithm is a network configuration with the best average message latency. This optimized configuration determines the position of PEs in the network and specifies the buffer size and the number of needed virtual channels for each port of every router.

Algorithm 1 Pseudo-code of the proposed heterogeneous NoC optimization based on GA

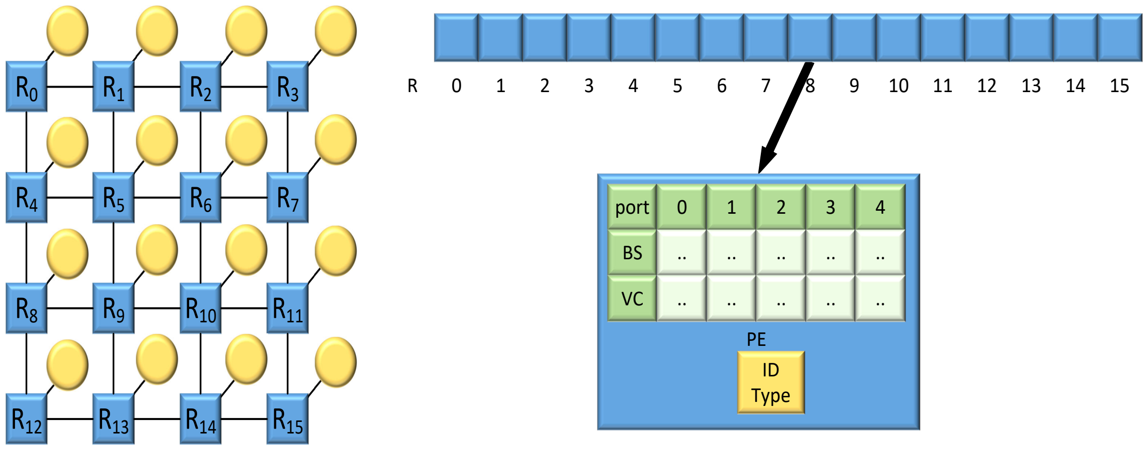
```
P = GENERATE initial population
g = 1        //generation counter
while $g \leq max\_generation$ do
    EVALUATE(P) and FIND the best solution
    $C = \{\}$
    while $size(C) < size(P)$ do
        if $rand() \leq CR$ then
            $\{Dad, Mom\}$ = SELECT two parents from P
            $\{child_1, child_2\}$ = CROSSOVER ON$\{Dad, Mom\}$
        else
            $\{child_1, child_2\} = \{Dad, Mom\}$
        end if
        $\{child_1\}$ = MUTATION ON$\{child_1\}$
        $\{child_2\}$ = MUTATION ON$\{child_2\}$
        $C = C \cup \{child_1, child_2\}$
    end while
    EVALUATE(C) and UPDATE the best solution
    P = REPLACE using TOURNAMENT SELECTION on P and C
end while
```

5.1.1 Chromosome Representation

The chromosome, as shown in Figure 5.3, is represented as an array of the 2D mesh routers, where the index of the array determines the position of the router in the mesh NoC. Each router is represented as an object that has a PE and an array of port objects. The PE has a unique id and a type. Each port has a buffer size and number of virtual channels. In this representation, the PE assignment determines the placement, and the ports array adds the heterogeneity to the NoC and specifies its configuration.

5.1.2 Initial Population and Fitness Function

The algorithm starts with a fixed-size population of chromosomes generated randomly, such that each PE is assigned to a unique router and each output port is assigned a random buffer



(a) 2D Mesh NoC

(b) Chromosome

Figure 5.3: Chromosome representation of NoC design for three sub-problems.

size and virtual channels within the boundaries. The objective is to minimize the average message latency; Equation (3.1) is used to evaluate the chromosomes. Additionally, the area of the design is calculated as the total size of buffers in NoC:

$$Area = \sum_{\forall\ Router\ \ R, outport\ \ p} V_p^R * B_p^R \tag{5.1}$$

When more than one chromosomes are equal in their fitness, the chromosome with the least area is preferred. During the evolution process, two types of populations with the same size are maintained: parents population and children population. Whenever a generation is evaluated, the global best found so far is updated.

5.1.3 Selection

A k-Tournament selection is applied to the parents' population to select the parents. First, k random chromosomes are chosen from the population to compete to be a parent. The chromosome with the best fitness among them is chosen as the first parent (Dad). The process is repeated to select the second parent (Mom).

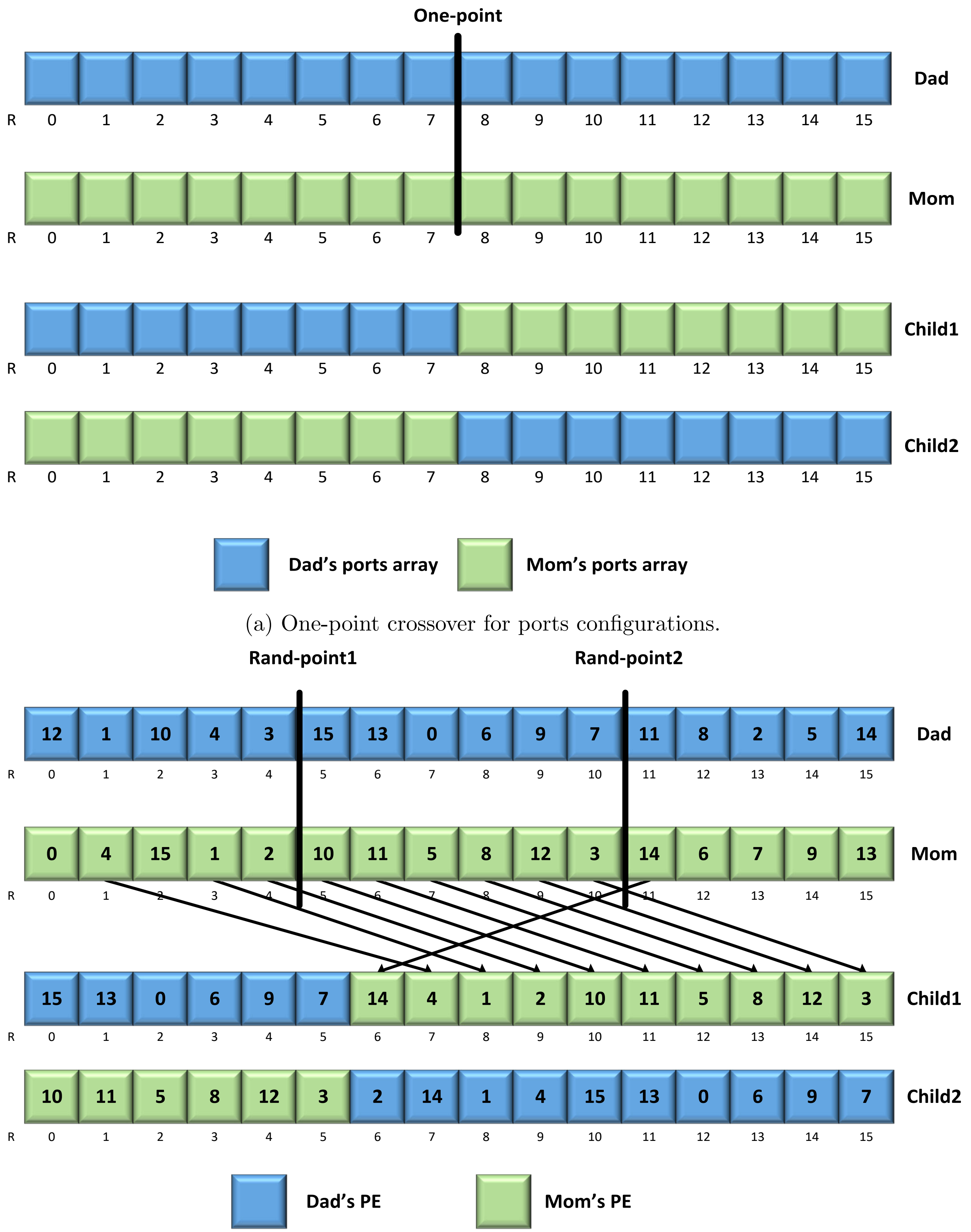
```
Algorithm 2 CROSSOVER
Input: $Dad, Mom$
Output: $Child1, Child2$
$Child1, Child2 = Port\_Crossover(Dad, Mom)$
$Child1, Child2 = Placement\_Crossover(Dad, Mom)$
```

5.1.4 Crossover Operator

Two types of crossover operators are applied to the selected parents according to a crossover rate (CR) as in Algorithm 2; otherwise, the parents are copied to the children population. One-point crossover is applied to change the buffer size and virtual channels of router's ports. For the placement of the PEs, partially mapped crossover (PMX) is applied to ensure a one-to-one mapping between the PEs and the routers.

Figure 5.4a shows an example of one-point crossover; first, a random point is chosen. Then, the port settings (BS and VC) of routers from the start of the dad chromosome to the random point are copied to child1 while the rest is copied from the mom chromosome. Similarly, the port settings of routers from the start of the mom chromosome to the random point are copied to child2 while the rest is copied from the dad chromosome.

Figure 5.4b shows one example of PMX crossover. First, two different points are chosen randomly. The PEs of the routers between the two points are copied from the dad chromosome to the beginning of child1. The rest unassigned PEs are copied in the order they appear in the mom chromosome beginning from the second random point to the end of the mom chromosome and starting over till the second point. Child2 is produced similarly.


One-point
Dad
R 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
Mom
R 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
Child1
R 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
Child2
R 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
Dad's ports array
Mom's ports array
Rand-point1
Rand-point2
12 1 10 4 3 15 13 0 6 9 7 11 8 2 5 14
Dad
R 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
0 4 15 1 2 10 11 5 8 12 3 14 6 7 9 13
Mom
R 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
15 13 0 6 9 7 14 4 1 2 10 11 5 8 12 3
Child1
R 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
10 11 5 8 12 3 2 14 1 4 15 13 0 6 9 7
Child2
R 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
Dad's PE
Mom's PE

(a) One-point crossover for ports configurations.

(b) Partially mapped crossover (PMX) for PEs placement.

Figure 5.4: Crossover operators applied on parents chromosomes to generate two children.

5.1.5 Mutation Operator

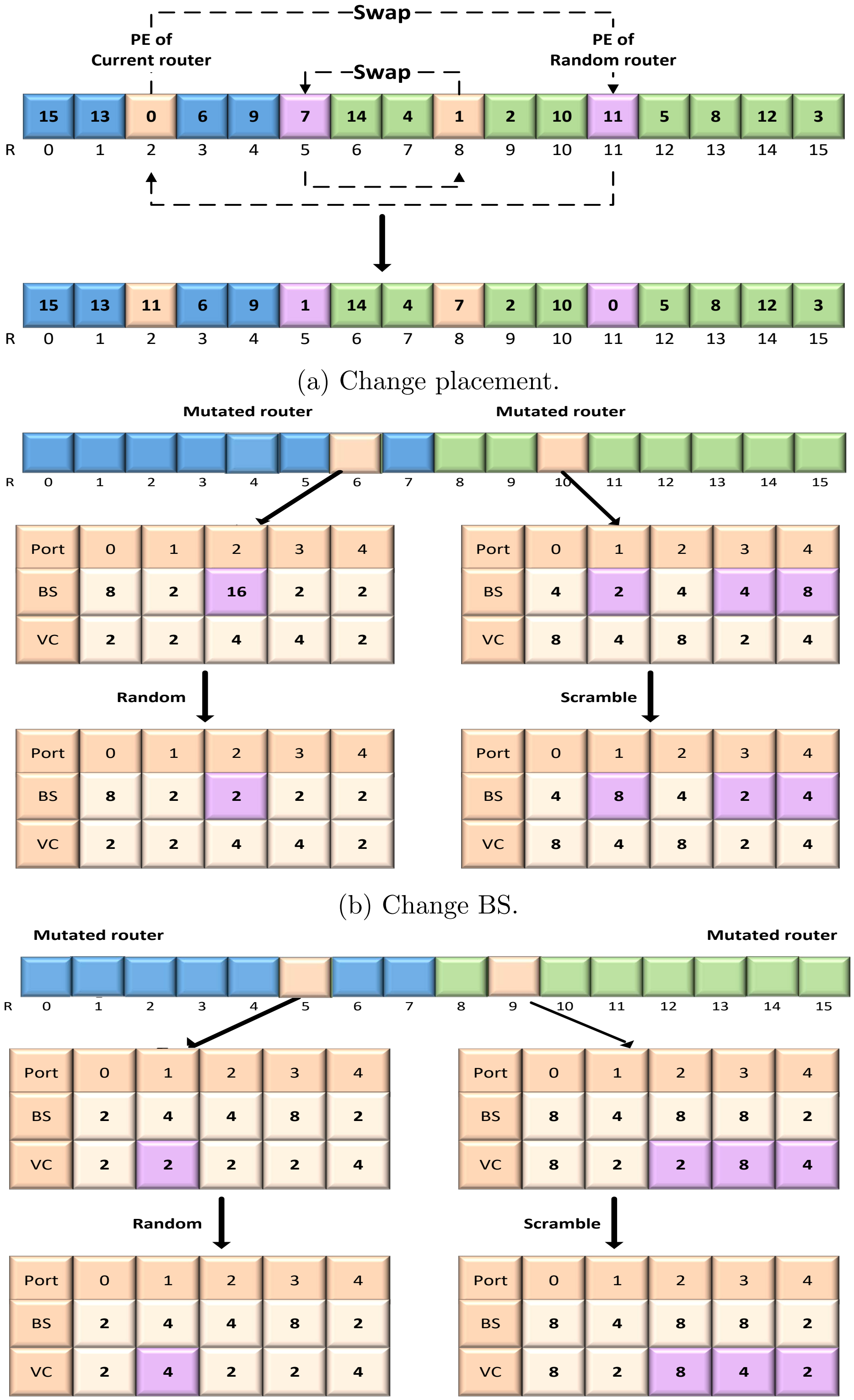
As in Algorithm 3, one of four design choices of mutation is applied randomly on routers of the child chromosome according to a mutation rate (MR): 1) change placement, 2) change buffer size, 3) change virtual channels, and 4) change all.

Figure 5.5, shows an example of mutation operators. To change the placement, the PE of the current router is swapped with the PE assigned to a random router as in Figure 5.5a.

One of three design choices is applied randomly to change the buffer size, Figure 5.5b. The first choice is to just randomly change the value of the buffer size of a randomly selected port from the current router. The second choice is to scramble the buffer sizes of three random ports of the current router. The third choice is to apply both the first and second design choices. The number of virtual channels of the current router is changed similarly, as shown in Figure 5.5c.

Algorithm 3 MUTATION

```
Input: $Child$
Output: $Child$
for $r = 0$ to NoC_SIZE $- 1$ do
  if RAND() $\leq$ MR then
    switch (RAND()%4)
    case 0:
      $Child = Placement\_Mutation(Child)$
    case 1:
      $Child = BS\_Mutation(Child)$
    case 2:
      $Child = VC\_Mutation(Child)$
    default:
      $Child = Placement\_Mutation(Child)$
      $Child = BS\_Mutation(Child)$
      $Child = VC\_Mutation(Child)$
    end switch
  end if
end for
```



R	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	15	13	0	6	9	7	14	4	1	2	10	11	5	8	12	3

R	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	15	13	11	6	9	1	14	4	7	2	10	0	5	8	12	3

(a) Change placement.

Port	0	1	2	3	4
BS	8	2	16	2	2
VC	2	2	4	4	2

Port	0	1	2	3	4
BS	8	2	2	2	2
VC	2	2	4	4	2

Port	0	1	2	3	4
BS	4	2	4	4	8
VC	8	4	8	2	4

Port	0	1	2	3	4
BS	4	8	4	2	4
VC	8	4	8	2	4

(b) Change BS.

Port	0	1	2	3	4
BS	2	4	4	8	2
VC	2	2	2	2	4

Port	0	1	2	3	4
BS	2	4	4	8	2
VC	2	4	2	2	4

Port	0	1	2	3	4
BS	8	4	8	8	2
VC	8	2	2	8	4

Port	0	1	2	3	4
BS	8	4	8	8	2
VC	8	2	8	4	2

(c) Change VC.

Figure 5.5: Mutation operators applied on the routers of child chromosome.

5.1.6 Replacement and Termination Criteria

The two populations, parents and children, are combined in one population. K-tournament selection is applied to the combined population to select the survivors for the next generation. K chromosomes are selected randomly, and the one with the best fitness is copied to the parent population of the next generation. The process is repeated until the new parent population has the same size as the original population. Elitism is used such that the global best is always copied into the new generation.

The evolution process is repeated until either a maximum number of generations is reached, or the best solution found so far stooped improving for a predefined number of generations. Then, the algorithm returns the global best with the best PEs mapping, NoC buffer size, and virtual channels configurations.

5.2 SPEA2 NoC Design for Three Sub-Problems

The objectives of the proposed SPEA2 [4] is to minimize network delay (average packet latency) and the total power consumption of the NoC. The models in Chapter 3 and Chapter 4 are used as a measure of performance and power, respectively. The optimization problem can be described as follows:

Minimize

$$\sum_{\forall S,D} P^{S\rightarrow D}\, L^{S\rightarrow D}$$

$$\sum_{R} P_R^{Router}$$

Subject to:

$$1 \leq B_p^R \leq B_{MAX} \qquad \forall\ Router\ R, outport\ p$$

$$2 \leq V_p^R \leq V_{MAX} \qquad \forall\ Router\ R, outport\ p$$

An Pseudo-code of the proposed method is shown in Algorithm 4. The algorithm, takes the network dimensions, maximum buffer size per port, maximum virtual channels per port, and the set of PEs to be placed along with their injection rates and communication rates as inputs. The output of this algorithm is a Pareto optimal set. The optimized configurations determine the position of PEs in the network, specify the buffer size, and the number of needed virtual channels for each port of every router.

5.2.1 Chromosome Representation

The chromosome representation explained in Section 5.1.1 is used to model the problem.

Algorithm 4 Pseudo-code of the proposed multi-objective heterogeneous NoC optimization based on SPEA2

```
$P$ = GENERATE initial population
$A = \{\}$
$g = 0$        //generation counter
loop
   ASSIGN FITNESS($P, A$)
   $A_{g+1}$ = Environmental_Selection($P \cup A$)
   if $g \geq$ MAX_GENERATIONS then
      $A = A_{g+1}$
      return A
   end if
   $C = \{\}$
   while SIZE($C$) < POPULATION_SIZE do
      $\{Dad, Mom\}$ = SELECTION($A_{g+1}$)
      if RAND() $\leq$ CR then
         $\{child_1, child_2\}$ = CROSSOVER($Dad, Mom$)
      else
         $\{child_1, child_2\} = \{Dad, Mom\}$
      end if
      $\{child_1\}$ = MUTATION($child_1[r]$)
      $\{child_2\}$ = MUTATION($child_2[r]$)
      $C = C \cup \{child_1, child_2\}$
   end while
   $P = C$
   $A = A_{g+1}$
   $g = g + 1$
end loop
```

5.2.2 Initial Population

The algorithm starts with a random population of solutions P, an empty archive A, and an empty children population C. In each random solution, each router is assigned to a unique PE and each output port is assigned a random buffer size and virtual channels within the boundaries.

5.2.3 Dominate Solution

A solution i dominates ($\succ$) solution j if it is better than solution j in at least one of the objective functions and is not worse than solution j in any objective function, see Algorithm 5. The first objective is the performance, calculated as in (3.1). The second objective is the power of NoC as in (4.10).

Algorithm 5 Dominate

```
Input: $S_1, S_2$
Output: $S_1 \succ S_2$?
if $\big(\textbf{AVG\_Latency}(S_1) > \textbf{AVG\_Latency}(S_2)\big) \;||\; \big(\textbf{Power}(S_1) > \textbf{Power}(S_2)\big)$ then
    return FALSE
end if
if $\big(\textbf{AVG\_Latency}(S_1) < \textbf{AVG\_Latency}(S_2)\big) \;||\; \big(\textbf{Power}(S_1) < \textbf{Power}(S_2)\big)$ then
    return TRUE
end if
return FALSE
```

5.2.4 Fitness Function

A fitness is assigned to each solution in P and A using two measures: the solution's raw fitness and its density.

$$Fitness(S) = Raw\_Fitness(S) + Density(S) \tag{5.2}$$

The raw fitness of the solution is computed based on a strength measure of the solutions. The strength of a solution represents the number of solutions it dominates, see Algorithm 6. Then, the raw fitness of a solution is calculated as the sum of the strength value of all the solutions that dominate it, see Algorithm 7.

The density of a solution is a decreasing function of the distance, in the objectives space, to the k-th nearest neighbor solution (σ^k), where k is commonly the square root of the sum of the population size and the archive size.

$$Density(S) = \frac{1}{\sigma_S^k + 2} \tag{5.3}$$

Algorithm 6 Strength

```
Input: $P \cup A, S$
Output: $Strength\ of\ S$
num_dominate $= 0$
for $s_1 = 0$ to $s_1 < SIZE(P \cup A) - 1$ do
    if Dominate($S, s_1$) then
        num_dominate++
    end if
end for
return num_dominate
```

Algorithm 7 Raw_Fitness

```
Input: $P \cup A, S$
Output: $Raw\_Fitness\ of\ S$
strength_dominate $= 0$
for $s_1 = 0$ to $s_1 < SIZE(P \cup A) - 1$ do
    if Dominate($s_1, S$) then
        strength_dominate += Strength($s_1$)
    end if
end for
return strength_dominate
```

5.2.5 Environmental Selection

The non-dominated solutions (with fitness < 1) of the combined regular population P and archive A of the current generation are copied to a new archive A_{g+1}. If the non-dominated solutions fit exactly in the fixed size of the archive, the environmental selection step is done. Otherwise, there are two possibilities, the size of the new archive is less than the fixed size, or it exceeds the fixed size, see Algorithm 8. In the first case, the dominated solutions of the combined populations with the best fitness are added to the next archive until the new archive size reaches the fixed size. In the second case, a truncation operation is applied to the new archive, where solutions are removed iteratively from the new archive until its size is equal to the fixed archive size. In each iteration, the solution with the minimum distance to another solution is chosen for removal. In case of a tie, the second smallest distances are considered and so forth.

Algorithm 8 Environmental Selection

```
Input: $P \cup A, S$
Output: $\textit{new archive } A_{g+1}$
$T = $ SORT_Ascending$(P \cup A, Fitness)$
$A_{g+1} = \{\}$
while Fitness$(T[i]) < 1$ do
    $A_{g+1} = A_{g+1} \cup T[i]$
end while
if SIZE$(A_{g+1}) <$ ARCHIVE_SIZE then
    repeat
        if Fitness$(T[i]) \geq 1$ then
            $A_{g+1} = A_{g+1} \cup T[i]$
        end if
    until SIZE$(A_{g+1}) ==$ ARCHIVE_SIZE
else if SIZE$(A_{g+1}) >$ ARCHIVE_SIZE then
    TRUNCATE$(A_{g+1})$
end if
return $A_{g+1}$
```

5.2.6 Selection

Binary tournament selection is applied to the new archive to select the parents. Two solutions are selected randomly from the new archive, and the solution that dominates the other is chosen as a parent. The process is repeated to select the second parent.

5.2.7 Crossover Operator

Two types of crossover are applied to the selected parents according to a crossover rate (CR); otherwise, the parents are copied to the children population, see Algorithm 2. The placement of the PEs and the port's configurations (BS and VC) are changed using PMX and one-point crossover, respectively, as explained in Section 5.1.4.

5.2.8 Mutation Operator

Three types of mutation are applied according to a mutation rate, as in Algorithm 3, to change 1) placements of PE, 2) buffer size, and 3) virtual channels. These three types of mutation are applied randomly to each router of the child chromosome, as explained in Section 5.1.5.

5.2.9 Replacement and Termination Criteria

After the reproduction process, the regular population is replaced by the generated children population and the archive is replaced by the new archive. The process is repeated for a maximum number of generations. Then, the algorithm returns the archive as the optimal Pareto set with the best PEs mapping, NoC buffer size, and virtual channels configurations.

5.3 SPEA2-BW NoC Design for Four Sub-Problems

The objectives of the proposed SPEA2-BW are the same as the proposed SPEA2-based method, explained in the previous Section; minimize network delay (average packet latency) and the total power consumption of the NoC. The models in Chapter 3 and Chapter 4 are used as a measure of performance and power, respectively. The heterogeneous bandwidth is added as a target for the final NoC design. The optimization problem can be described as follows:

Minimize

$$\sum_{\forall S,D} P^{S\to D}\, L^{S\to D}$$

$$\sum_{R} P_R^{Router}$$

Subject to:

$$1 \leq B_p^R \leq B_{MAX} \qquad \forall\ Router\ R, outport\ p$$

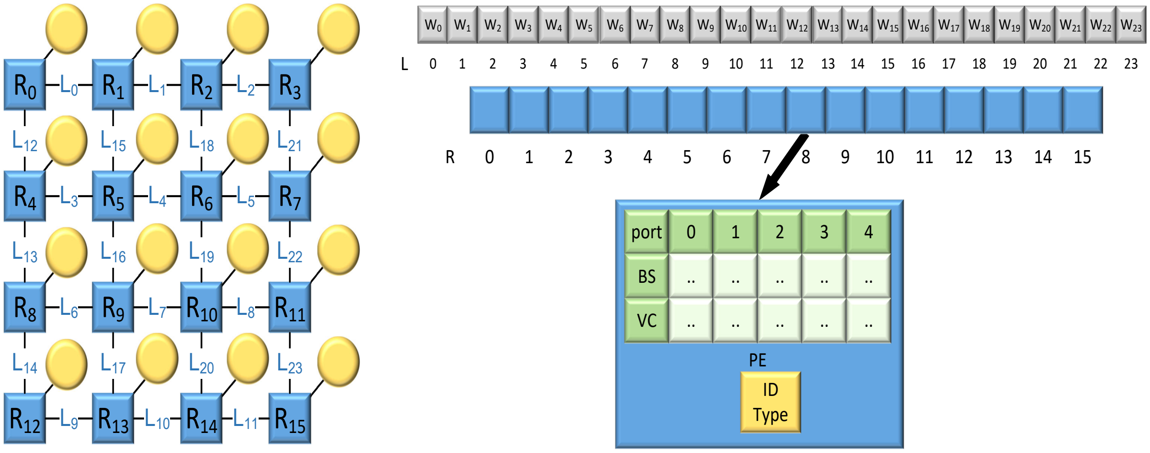
$$2 \leq V_p^R \leq V_{MAX} \qquad \forall\ Router\ R, outport\ p$$

$$W^L \in \{W_1, W_2, ..., W_n\} \qquad \forall\ Link\ L$$

The same Pseudo-code shown in Algorithm 4 is used. In addition to the previous explained inputs, the algorithm takes the set of available link bandwidths. The output of this algorithm is a Pareto optimal set. The optimized configurations determine the position of PEs in the network, specify the buffer size and the number of needed virtual channels for each port of every router, and specify the bandwidth of each link in NoC.

5.3.1 Chromosome Representation

The chromosome representation explained in Section 5.1.1 is used to model the problem. Moreover, an array of link's bandwidth is used to represent the bandwidth of the NoC . The index of the array is the link id in the NoC, and its value is the bandwidth assigned to it, see Figure 5.6.


R0 L0 R1 L1 R2 L2 R3
L12 L15 L18 L21
R4 L3 R5 L4 R6 L5 R7
L13 L16 L19 L22
R8 L6 R9 L7 R10 L8 R11
L14 L17 L20 L23
R12 L9 R13 L10 R14 L11 R15
W0 W1 W2 W3 W4 W5 W6 W7 W8 W9 W10 W11 W12 W13 W14 W15 W16 W17 W18 W19 W20 W21 W22 W23
L 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
R 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
port 0 1 2 3 4
BS .. ..
VC .. ..
PE
ID
Type

(a) 2D Mesh NoC

(b) Chromosome

Figure 5.6: Chromosome representation of NoC design for four sub-problems.

5.3.2 Initial Population

The algorithm starts with a random population of solutions P, an empty archive A, and an empty children population C. In each random solution, each router is assigned to a different PE and each output port is assigned a random buffer size and virtual channels within the boundaries. Moreover, each link is assigned a random bandwidth from the set of available bandwidths.

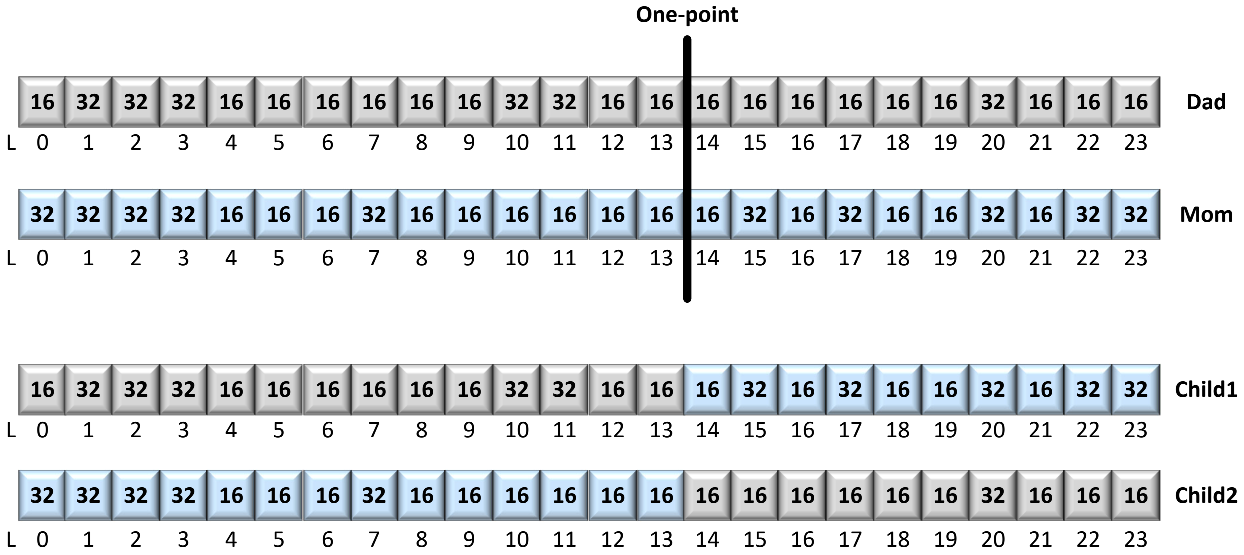
5.3.3 Crossover Operator

Three types of crossover, as in Algorithm 9, are applied to the selected parents according to a crossover rate (CR); otherwise, the parents are copied to the children population. The placement of the PEs and the port's configurations (BS and VC) are changed using PMX and one-point crossover, respectively, as explained in Section 5.1.4.

The bandwidth of the links is changed using one-point crossover as in Figure 5.7. One point is chosen randomly, and the bandwidth of the two parents are swapped after this point to generate two children.

Algorithm 9 CROSSOVER

```
Input: $Dad, Mom$
Output: $Child1, Child2$
$Child1, Child2 = Port\_Crossover(Dad, Mom)$
$Child1, Child2 = Placement\_Crossover(Dad, Mom)$
$Child1, Child2 = BW\_Crossover(Dad, Mom)$
```



L	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
Dad	16	32	32	32	16	16	16	16	16	16	32	32	16	16	16	16	16	16	16	16	32	16	16	16
Mom	32	32	32	32	16	16	16	32	16	16	16	16	16	16	16	32	16	32	16	16	32	16	32	32
Child1	16	32	32	32	16	16	16	16	16	16	32	32	16	16	16	32	16	32	16	16	32	16	32	32
Child2	32	32	32	32	16	16	16	32	16	16	16	16	16	16	16	16	16	16	16	16	32	16	16	16

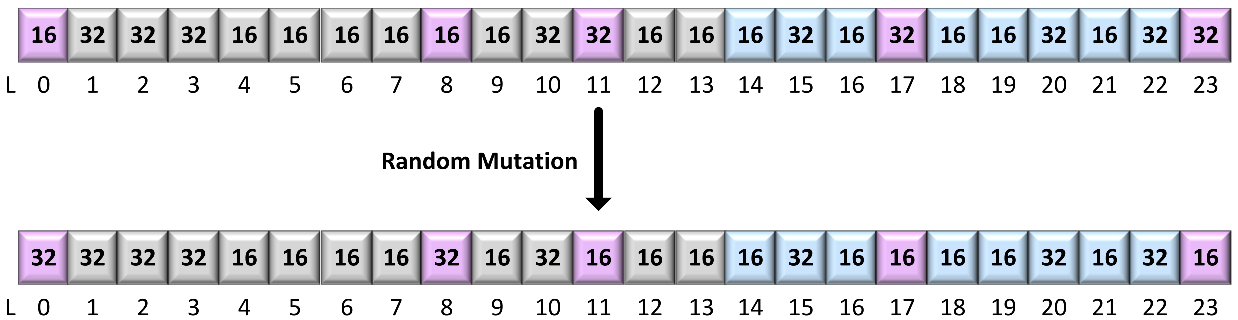
Figure 5.7: One-point crossover to change the links' bandwidth.

5.3.4 Mutation Operator

Four types of mutations are applied according to a mutation rate, as in Algorithm 10, to change 1) placements of PE, 2) buffer size, 3) virtual channels, and 4) links' bandwidth. The first three types of mutations are applied randomly to each router of the child chromosome, as explained in Section 5.1.5. The last type is applied to each link of the child chromosome to change the bandwidth randomly, as in Figure 5.8.

Algorithm 10 MUTATION

```
Input: $Child$
Output: $Child$
for $r = 0$ to NoC_SIZE $- 1$ do
  if RAND() $\leq$ MR then
    switch (RAND%4)
    case 0:
      $Child = Placement\_Mutation(Child)$
    case 1:
      $Child = BS\_Mutation(Child)$
    case 2:
      $Child = VC\_Mutation(Child)$
    default:
      $Child = Placement\_Mutation(Child)$
      $Child = BS\_Mutation(Child)$
      $Child = VC\_Mutation(Child)$
    end switch
  end if
end for
for $l = 0$ to LINKS_SIZE $- 1$ do
  if RAND() $\leq$ MR then
    $Child = BW\_Mutation(Child)$
  end if
end for
```



L	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
	16	32	32	32	16	16	16	16	16	16	32	32	16	16	16	32	16	32	16	16	32	16	32	32

L	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
	32	32	32	32	16	16	16	16	32	16	32	16	16	16	16	32	16	16	16	16	32	16	32	16

Figure 5.8: Mutation operator to change the links' bandwidth.

5.3.5 Replacement and Termination Criteria

After the reproduction process, the regular population is replaced by the generated children population and the archive is replaced by the new archive. The process is repeated for a maximum number of generations. Then, the algorithm returns the optimal Pareto set with the best PEs mapping, NoC buffer size and virtual channels configurations, and NoC bandwidth.

Chapter 6

Results

A full-system CPU-GPU simulator gem5-gpu [32] was used to obtain processor and network-level information. This simulator is based on gem5 [10] and gpgpu-sim [7]. It can model tightly integrated CPU-GPU systems under different interconnections and coherency protocols. The interconnection network is modeled using GARNET [2], a flit-level NoC model.

For simulating heterogeneous NoCs for different hardware configurations, some modifications to gem5-gpu simulator were needed. Firstly, adding support to connect CPU cores and GPU cores in a 2D mesh style NoC. Moreover, adding support for different buffer sizes and virtual channels, not only for each router but also for each port of each router.

This chapter shows the evaluation of three NoC design methodologies:

- **GA** for **performance** optimal NoC design considering three sub-problems simultaneously; **PE mapping, BS, VC**.
- **SPEA2** for **performance** and **power** optimal NoC design considering three sub-problems simultaneously; **PE mapping, BS, VC**.
- **SPEA2** for **performance** and **power** optimal NoC design considering four sub-

problems simultaneously; **PE mapping, BS, VC, BW**.

The parameters used for GA and SPEA2 are shown in Table 6.1, and were chosen after extensive parameter tuning experiments.

Table 6.1: GA and SPEA2 Parameters Used in NoC Optimization

Parameter	GA	SPEA2
Population size	32	32
Archive size	NA	32
Crossover rate (CR)	0.7	0.7
Mutation rate (MR)	0.5	0.5
Tournament Selection size (k)	8	2
Max Generations	10000	10000

The proposed NoC design methodologies were evaluated following three steps, as shown in Figure 6.1. The first step is to gather the traffic trace by running different workloads on the baseline architecture using gem5-gpu simulator. The second step is to feed the traffic trace as an input to the NoC design optimizer to get near optimal design. The last step is to run the optimal design on gem5-gpu and compare it with other NoC design methodologies. The NoC power was obtained by feeding the output of the simulator to DSENT [38]; a Design Space Exploration for Network Tool that supports Garnet Network within gem5-gpu. After modifying it to support heterogeneous buffers and virtual channels per port, a 22nm technology node was used to obtain NoC power. The criteria that were used to evaluate the different NoC designs are:

- **Total area**: The area was obtained from DSENT tool using 22nm technology node.
- **Average network latency**: The average network latency was obtained from gem5-gpu and in packets/cycle.
- **Percentage of non-blocking**: The average percentage of non-blocking for buffers was computed by finding the number of times the buffers of the whole NoC are not full out of the total number of times they are needed.

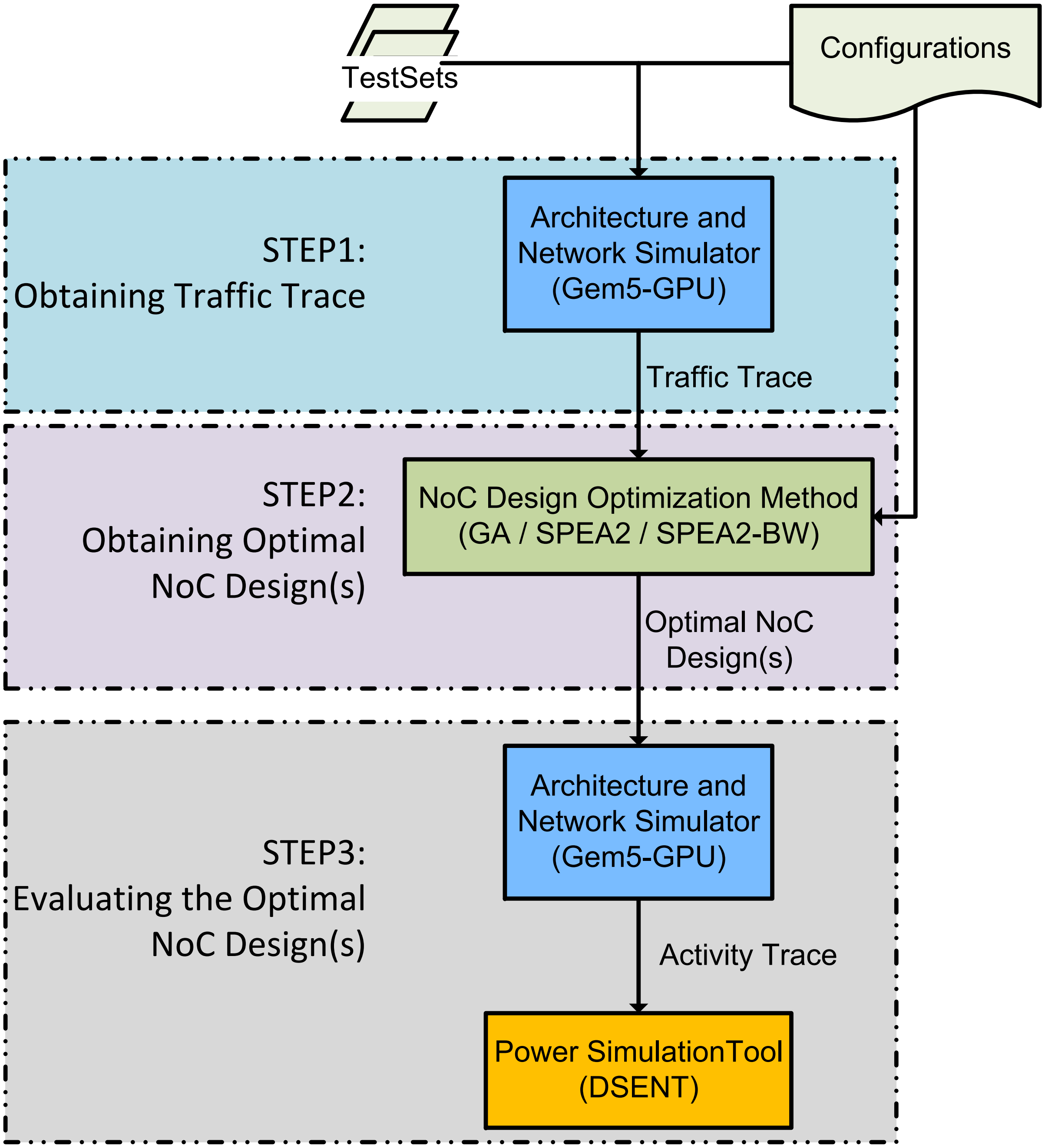

TestSets
Configurations
STEP1:
Obtaining Traffic Trace
Architecture and Network Simulator (Gem5-GPU)
Traffic Trace
STEP2:
Obtaining Optimal NoC Design(s)
NoC Design Optimization Method (GA / SPEA2 / SPEA2-BW)
Optimal NoC Design(s)
STEP3:
Evaluating the Optimal NoC Design(s)
Architecture and Network Simulator (Gem5-GPU)
Activity Trace
Power SimulationTool (DSENT)

Figure 6.1: A 3-steps evaluation methodology of the proposed NoC design methods.

- **NoC power**: The NoC power was obtained from DSENT tool using 22nm technology node.
- **NoC throughput**: The throughput of the NoC was measured as the average packets injected per cycle.
- **Average speedup**: The speedup in instructions per cycle (IPC) is calculated by finding the speedup of each benchmark with a configuration over the baseline configuration as in (6.1). Then, the geometric mean speedup for all CPU cores and SMs was computed as in (6.2) and (6.3), respectively. The overall system speedup was computed by taking the geometric mean of CPU and GPU speedups as in (6.4).

$$Speedup_i = IPC_i / IPC_i^{Baseline} \tag{6.1}$$

$$Speedup_{CPU} = geomean(Speedup_i);\ i \text{ is a CPU core} \tag{6.2}$$

$$Speedup_{GPU} = geomean(Speedup_i);\ i \text{ is a GPU core} \tag{6.3}$$

$$Speedup_{system} = geomean(Speedup_{CPU}, Speedup_{GPU}); \tag{6.4}$$

6.1 Baseline Architecture

For the purpose of testing, the system and network configurations shown in Table 6.2 and Table 6.3 were adopted, respectively. The architecture of the system consists of many x86 CPU cores fused with GPU on the same chip and connected through a 2D mesh NoC. Each CPU core has a private L1 cache. The GPU consists of multiple streaming multiprocessor cores (SM) each one with a private L1 cache. Moreover, to support 100s lanes of address translation [33], gem5-gpu provides the option of using a shared page walk cache (PW) that is accessed upon a miss in the SM's L1 TLBs to decrease the number of accesses to L2 cache and DRAM. The CPU cores and the SMs share the L2 cache. Both CPU and GPU share a virtual address space where MESI-Two-Level cache coherence protocol is used to ensure coherency. A baseline homogeneous architecture is shown in Figure 6.2. Based on the observations in [28], simple placement of the PEs was adopted by grouping the CPU cores, grouping the SMs, and placing the shared caches and MCs in the middle. This architecture was just chosen for testing, and the use of PW is optional, and the validation of the proposed methods does not depend on it.

Table 6.2: System Configuration for Gem5-gpu Simulation

PE type	Parameter	Value
GPU	Number of cores	6
	Core Clock	1.4 GHz
	Private L1 cache	4-way 32 kB
CPU	Number of cores	4
	Core Clock	2 GHz
	Private L1 I cache	2-way 32 kB
	Private L1 D cache	2-way 32 kB
Memory	Shared L2 cache	8-way 2 MB
	MC	4 (each 8 banks, 4 channels) 3.006 GHz, 1kB row-buffer FR-FCFS scheduler
	DRAM	DDR3-1600 16GB

Table 6.3: Baseline NoC Configurations

Configuration	Value
Topology	4 x 4 2D Mesh
Pipeline	5-stage (GA)/ 3-stage (SPEA2)
Routing	x-y Routing
Link width	16 B
Link latency	1 cycle
VC(Homog)	4 per port (8-flit buffer)
VC(Dual)	Big: 7 per port (8-flit buffer) Small: 3 per port (8-flit buffer)

For comparison with other buffer and virtual channel allocation schemes, the homogeneous baseline configuration (Homog) and DUAL approach, proposed by [29] for homogeneous CMPs, were considered, see Table 6.3. Both configurations use the PEs' placements shown in Fig 6.2. The baseline uses a homogeneous number of buffers and virtual channels for all ports of all routers. DUAL is based on using two types of routers, big and small. The big router has more VCs than the small router, but the number of VCs is homogeneous within all ports of the same router. Also, the buffer size is homogeneous through all ports of all routers. Their concept was applied based on the traffic generated using the homogeneous baseline; four out of the sixteen routers with higher injection rate were set to be big, and the rest were small. Seven virtual channels were used for the big router and three virtual channels for the small router to keep the total number of virtual channels, hence the area, less than or equal to the baseline.

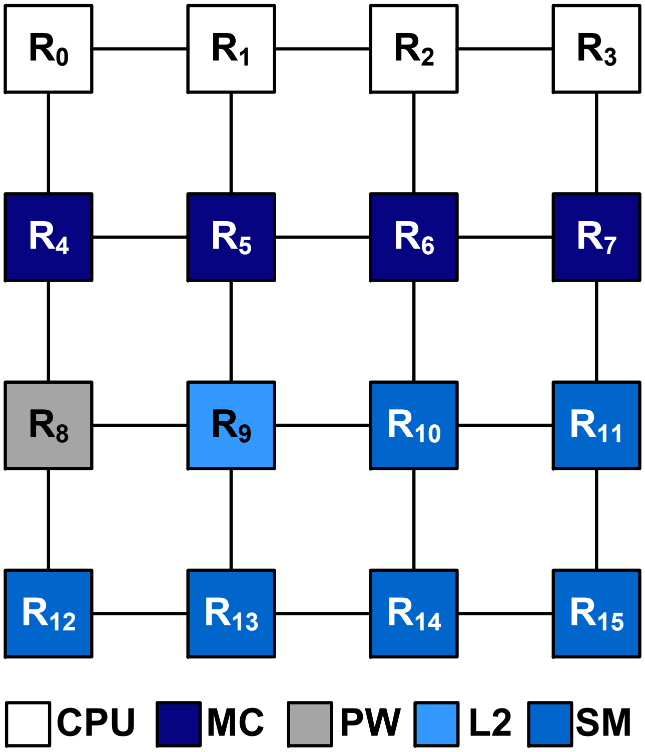


Figure 6.2: PEs' placement in the baseline architecture.

6.2 Benchmarks

Benchmarks from Rodinia [11] for GPU and Parsec [9] for CPU, as shown in Table 6.4, were used to obtain the traffic trace. The benchmarks were grouped into seven TestSets, see Table 6.5, each one is a workload composed of one GPU benchmark and three independent CPU benchmarks. Each benchmark is pinned to one CPU.

Using the baseline configurations, each TestSet was run on gem5-gpu until the GPU benchmark finished. Then, the benchmarks within the workload were rotated to different CPUs and reran, repeating this process for two more times. Finally, the four-generated injection rates and traffic traces were averaged and fed into the optimizer to obtain the optimal NoC design.

Table 6.4: Rodinia GPU Benchmarks and Parsec CPU Benchmarks

PE	Benchmark	Configurations
GPU	Backprop (BC)	1,048,576 layers
	Gaussian (G)	208 × 208 matrix
	HotSpot (HS)	1,024 rows, 2 height, 2 iterations, 1024 input
	LU Decomposition (LUD)	512 × 512 matrix
	Nearest Neighbor (NN)	5120k input, 5 records, 30 latitude, 90 longitude
	Needleman-Wunsch (NW)	16,384 maximum rows, 10 penalty
	Path Finder (PF)	100,000 rows, 100 columns, 20 pyramid height
CPU	Blackscholes (BS)	65,536 options
	Bodytrack (BT)	2 frames, 2,000 particles
	Canneal (C)	200,000 elements
	Dedup (D)	32.2 MB data
	Fluidanimate (FA)	5 frames, 300,000 particles
	Freqmine (FM)	990,000 transactions
	Streamcluster (SC)	16,384 points per block, 1 block
	Swaption (S)	16 swaptions, 20,000 simulations
	X264	128 frames, 640 × 360 pixels

Table 6.5: Workloads Combination of GPU and CPU Benchmarks

TestSet	Workload
1	BC, BT, C, D
2	G, C, S, SC
3	PF, x264, D, SC
4	HS, BS, S, C
5	NN, S, BT, FM
6	NW, SC, BS, FA
7	LUD, C, FM, x264

6.3 GA NoC Design for Three Sub-Problems

The generated traffic trace was fed to the proposed GA-based optimizer, and the optimal NoC design, that specifies the PEs placement and the buffer size and virtual channels for each port of each router, was obtained. The TestSets were rerun on gem5-gpu using the optimal design configuration and compared with Homog and DUAL configurations.

6.3.1 Total Area

The improvements in the area compared to the baseline is shown in Figure 6.3. There are no improvements in the area in the Dual configurations, which is expected since the concept is to choose big and small routers such that the total number of virtual channels is the same as the baseline. On the other hand, GA provides 34% improvements on average in the area.

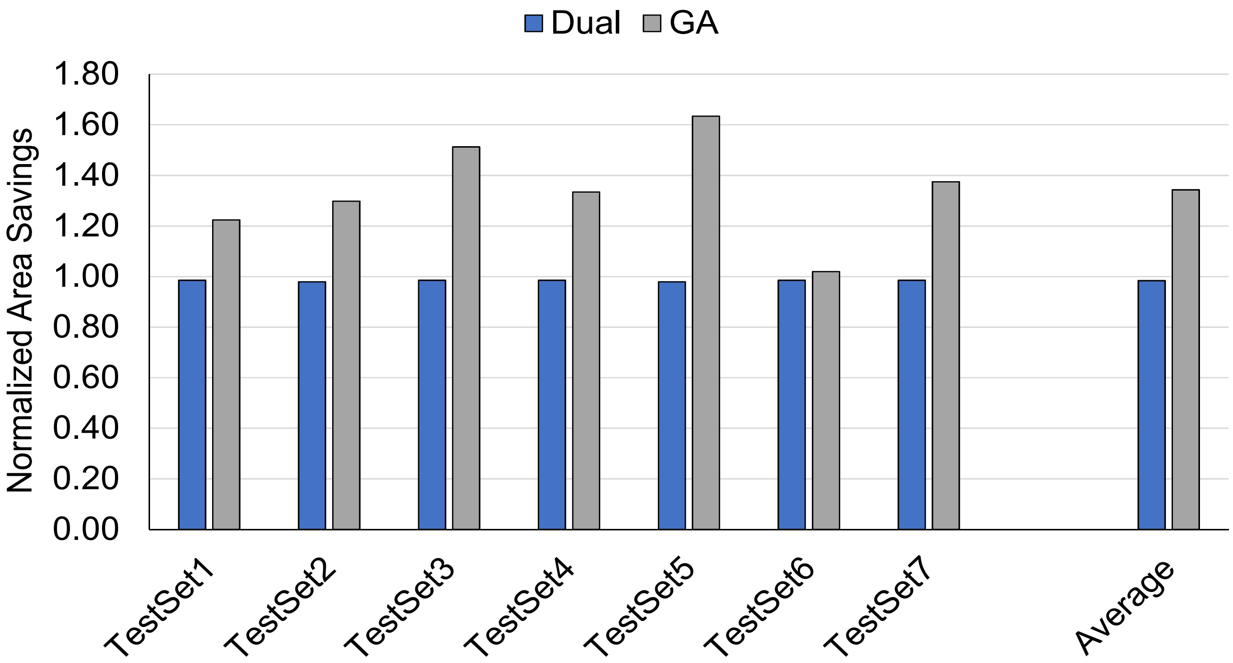


Figure 6.3: Improvement of NoC area of Dual and GA configurations normalized to the homogeneous baseline configuration.

6.3.2 Average Network Latency

Figure 6.4 shows the improvements in the average network latency normalized to the baseline. GA is better than the homogeneous baseline in all TestSets except for TestSet2. Moreover, GA gives better improvement than the Dual configurations for most of the TestSets, except TestSet2 and TestSet6. This is because the GPU benchmark in both TestSet2 and TestSet6 has a higher injection rate compared to the other. On average, GA provides about 19% improvement in the average network latency whereas Dual shows only 1%. In general, GA can reduce the average network latency while decreasing the area.

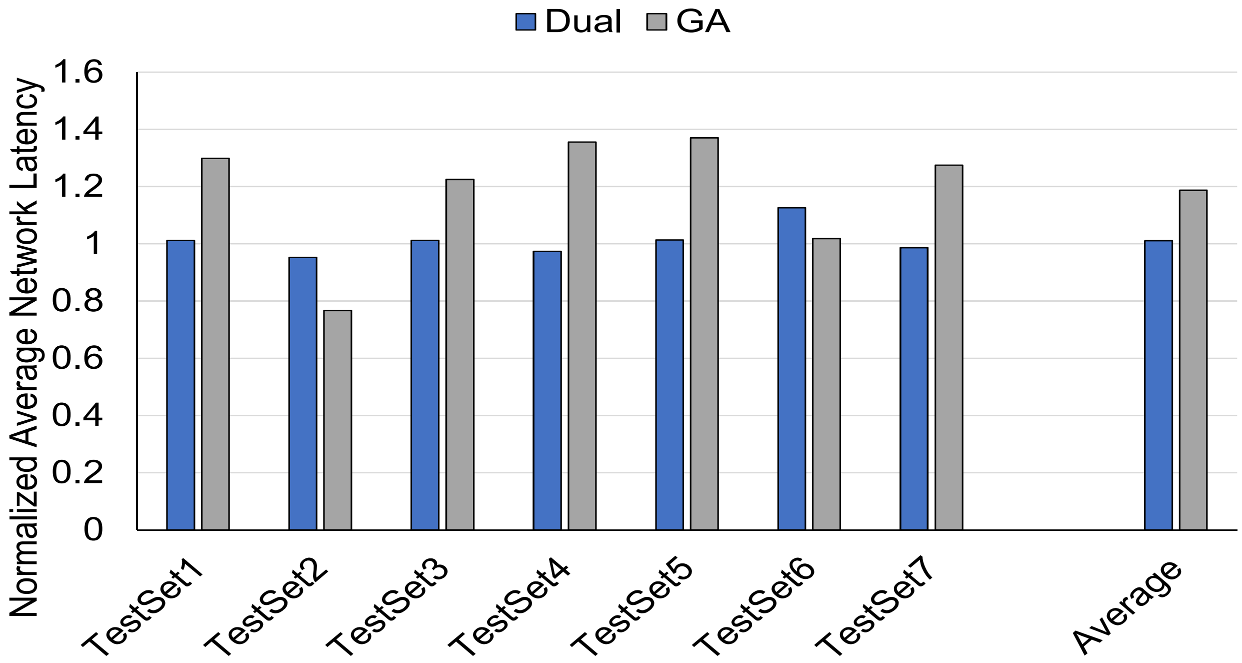


Figure 6.4: Improvement of average network latency of Dual and GA configurations normalized to the homogeneous baseline configuration.

6.3.3 Percentage of Non-Blocking

According to Figure 6.5, both the homogeneous and dual configurations have almost 100% of average non-blocking percentage; this can either means there are just enough buffers for the traffic, or there are extra buffers. Since GA has better average network latency while having an average percentage of non-blocking buffers of about 77%, this indicates that there are excess buffers in the baseline and Dual configurations.

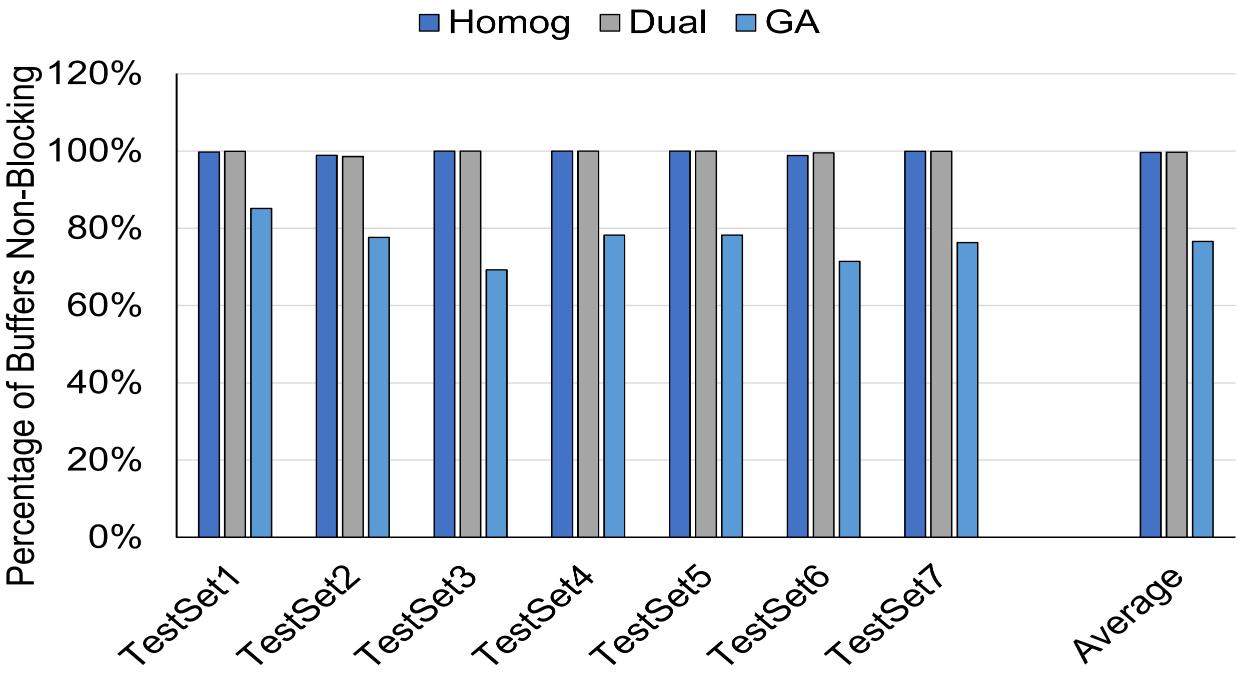


Figure 6.5: Comparison of average percentage of buffers' non-blocking for Homog, Dual, and GA configurations.

6.3.4 NoC Power

As shown in Figure 6.6, contrary to Dual, GA provides power savings in all of the TestSets compared to the homogeneous baseline configuration, and on average the savings reaches 37%. The NoC power consumption can be broken down into different components: buffer, clock, crossbar, switch, and link power consumption. Figure 6.7, shows the NoC power consumption break down under different configurations for the average of the seven TestSets. In all the configurations, the buffer is the component that contributes the most to the NoC power consumption. It contributes 91.95%, 92.02%, and 89.04% using Homog, Dual, and GA configurations respectively. Since the area, represented by the total buffer size, is considered in the proposed optimizer, the contribution of the buffer to the NoC power is decreased compared to other configurations.

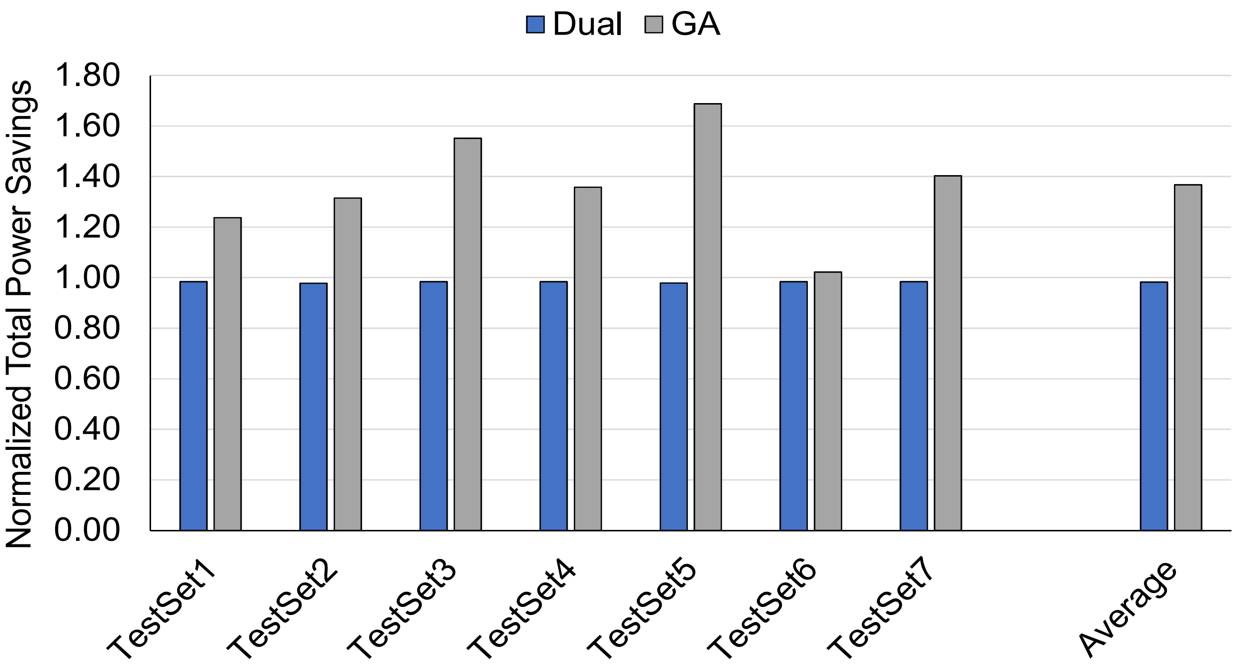


Figure 6.6: NoC power savings of Dual and GA configurations normalized to the homogeneous baseline configuration.

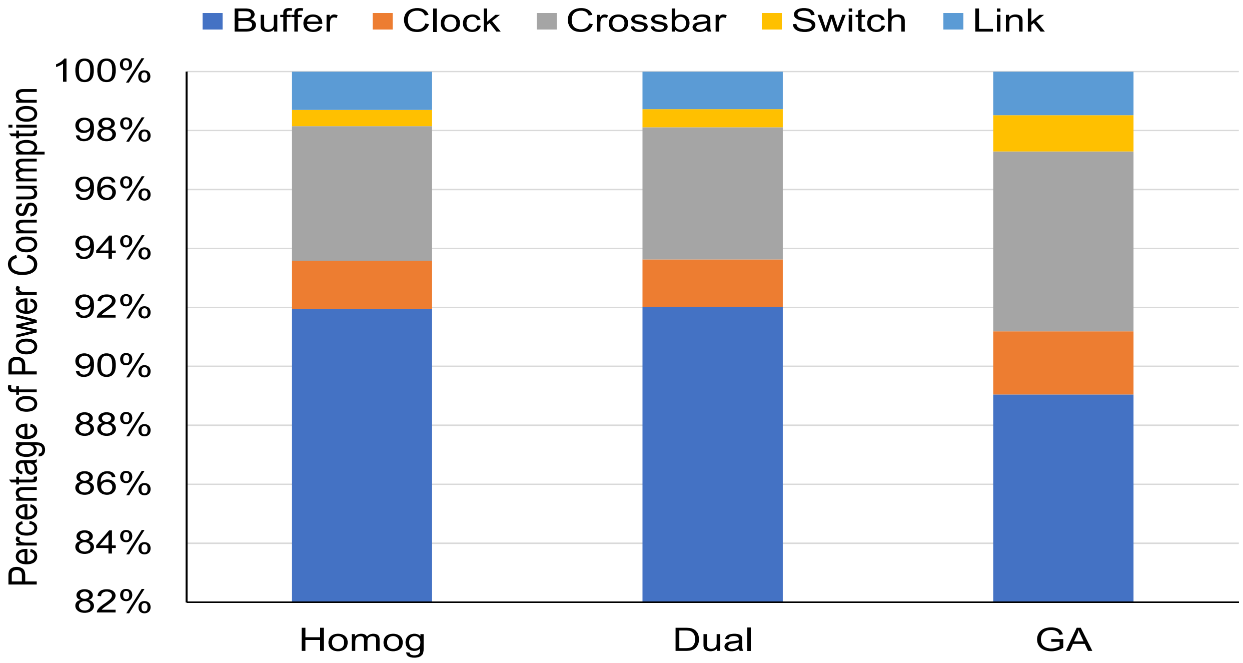


Figure 6.7: Comparison of NoC power consumption break-down for the average of TestSets under different NoC configurations.

6.3.5 NoC Throughput

Figure 6.8 shows the improvement of NoC throughput provided by both Dual and GA compared to the homogeneous baseline configuration. GA improves the NoC throughput compared to the baseline in all the TestSets, except TestSets 2 and 7, and on average has 2% improvement. Dual, on the other hand, degrades the NoC throughput or provides slight improvement and on average degrades the NoC throughput by 0.7%.

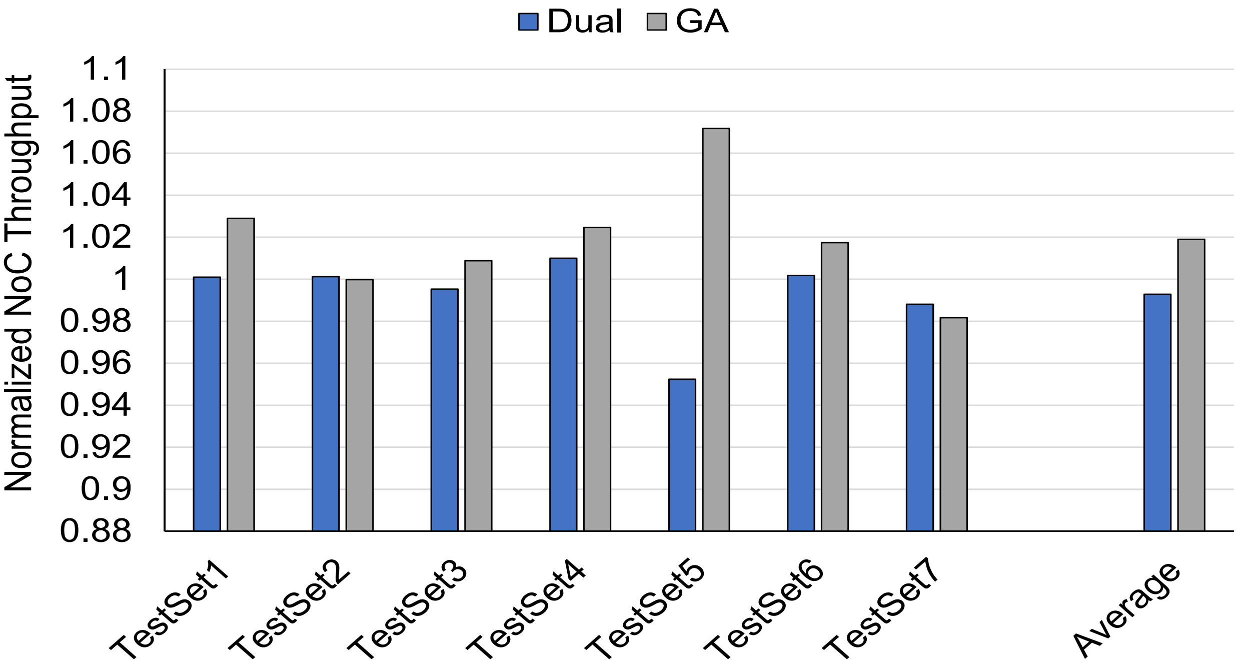


Figure 6.8: Improvement of NoC throughput of Dual and GA configurations normalized to the homogeneous baseline configuration.

6.3.6 Average Speedup

As shown in Figure 6.9, generally Dual and GA maintain the CPU speedup among TestSets, and GA slightly improves it by 1% on average while Dual slightly decreases it. However, in all TestSets GA provides better CPU speedup than Dual. On the other hand, there is a variation in the GPU speedup, Figure 6.10. While Dual improves the GPU speedup compared to the baseline in all TestSets and on average can reach up to 13%, GA provides slightly better improvement than the Dual for TestSets 1, 2, and 4 and on average the improvement over the baseline can reach up to 5.15%. On average, GA provides 3% overall system speedup and Dual provides 6%, as shown in Figure 6.11.

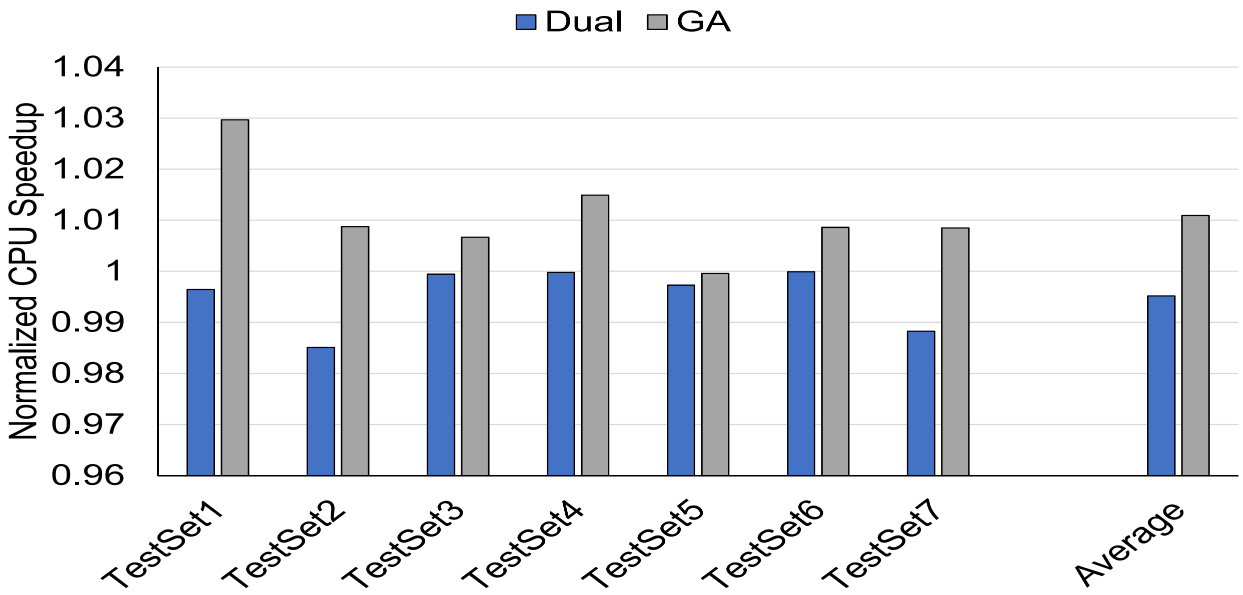


Figure 6.9: Average CPU speedup of Dual and GA configurations normalized to the homogeneous baseline configuration.

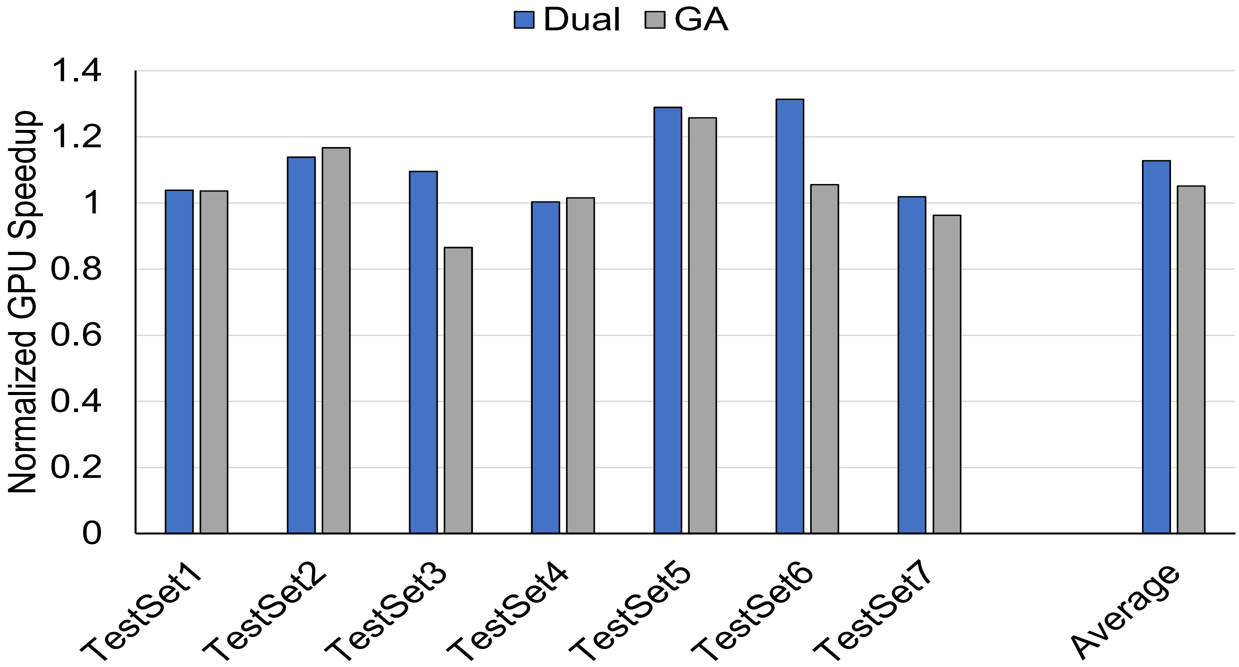


Figure 6.10: Average GPU speedup of Dual and GA configurations normalized to the homogeneous baseline configuration.

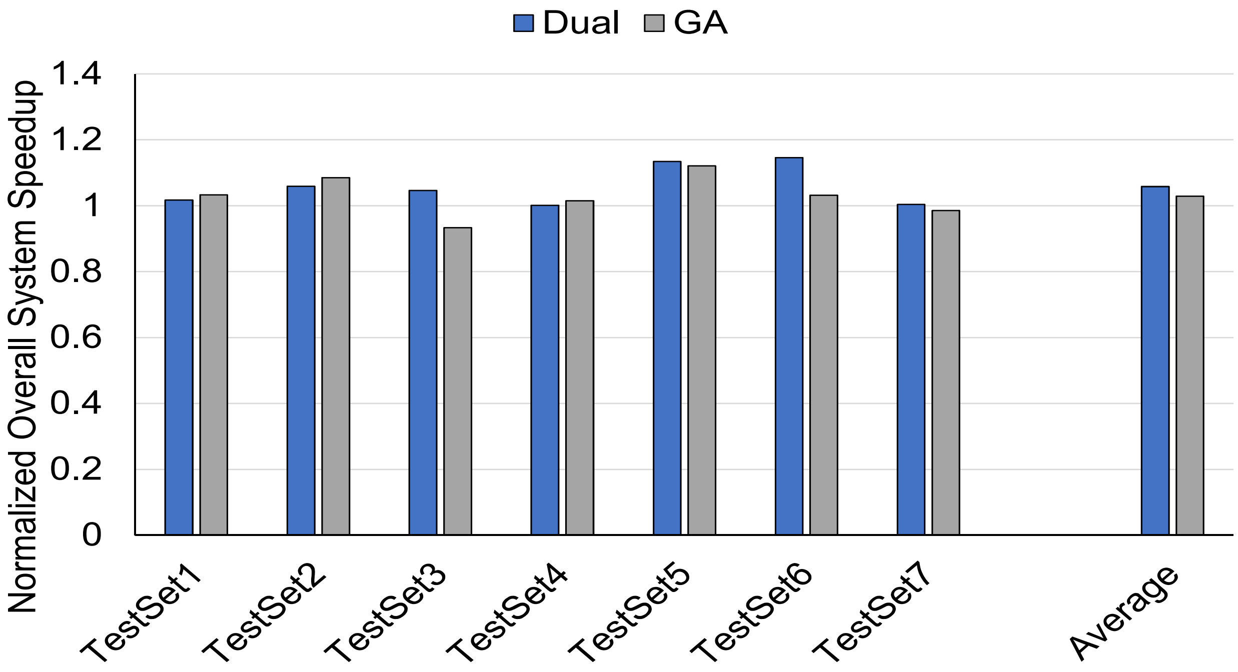


Figure 6.11: Overall speedup of the system of Dual and GA configurations normalized to the homogeneous baseline configuration.

6.3.7 Placement

The optimal configurations obtained from GA provide the placement of the PEs in the NoC. The TestSets were rerun using this placement for both the homogeneous configuration and the Dual configuration, then compared based on the average buffer occupation of the NoC. The average buffer occupation of the NoC is calculated as the total number of writes to all the buffers in NoC per cycle divided by the total number of buffers in NoC. Figure 6.12 shows the average buffer occupation of GA and Dual normalized to the homogeneous configuration. While Dual does not provide any improvement in the average buffer occupation, GA provides about 38% improvement on average. This indicates that GA improves the utilization of the buffers in the NoC.

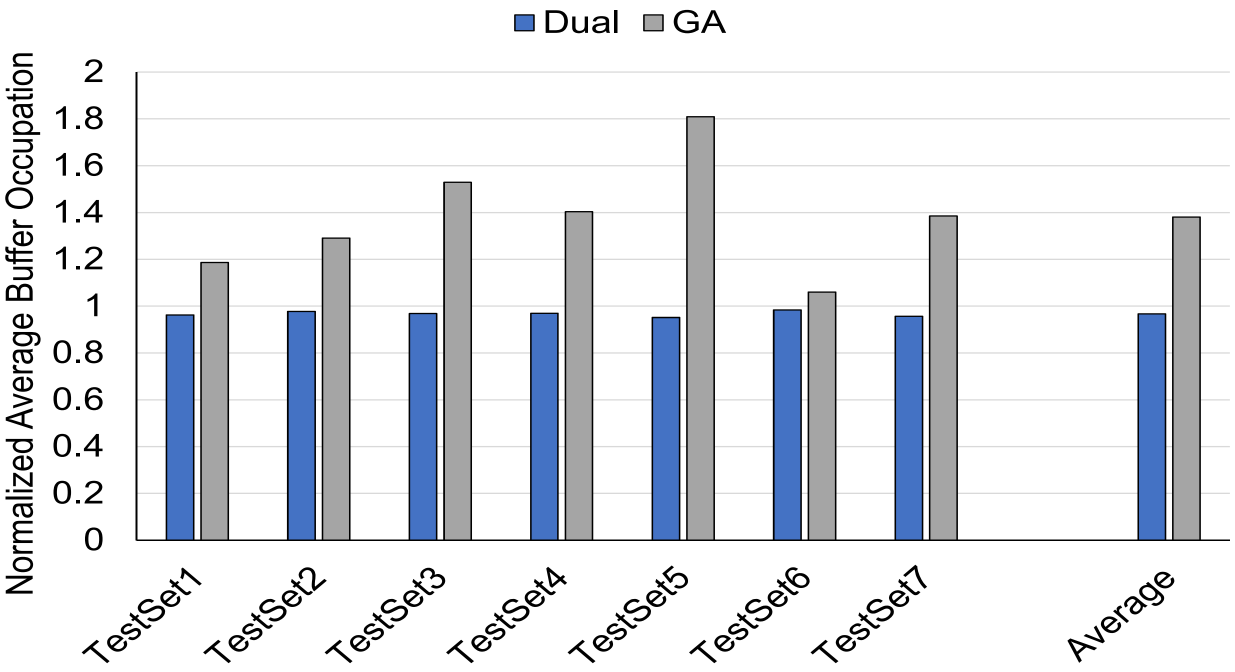


Figure 6.12: Average buffers occupation obtained by running the homogeneous and Dual configurations using GA optimal PEs' placement, normalized to the homogeneous configuration.

6.4 SPEA2 NoC Design for Three Sub-Problems

By running the proposed optimizer based on SPEA2 [4] on the traffic trace, a non-dominated Pareto optimal set of solutions is generated; each represents an NoC design with optimal mapping of PE and optimal assignment of buffer size and virtual channels per outport. Three solutions out of the Pareto-optimal set are considered for comparison: 1) The solution with the best performance (SPEA2-Latency), 2) The solution with the best power consumption (SPEA2-Power), and 3) The solution with the best fitness as in (5.2) (SPEA2-Fitness).

6.4.1 Total Area

Figure 6.13 shows the improvements in the area of all configurations compared to the baseline. As expected, Dual does not provide any improvement in the area. All SPEA2 optimal configurations improve the area, but they vary in the amount of improvement. Generally, power and fitness optimal configurations provide better area improvement than latency optimal configuration, except for TestSet5. In this TestSet, the fitness optimal solution happens to be the latency optimal solution. On average the improvement in the area are 4x, 2x, 4.33x, for the fitness, latency, and power optimal configurations, respectively.

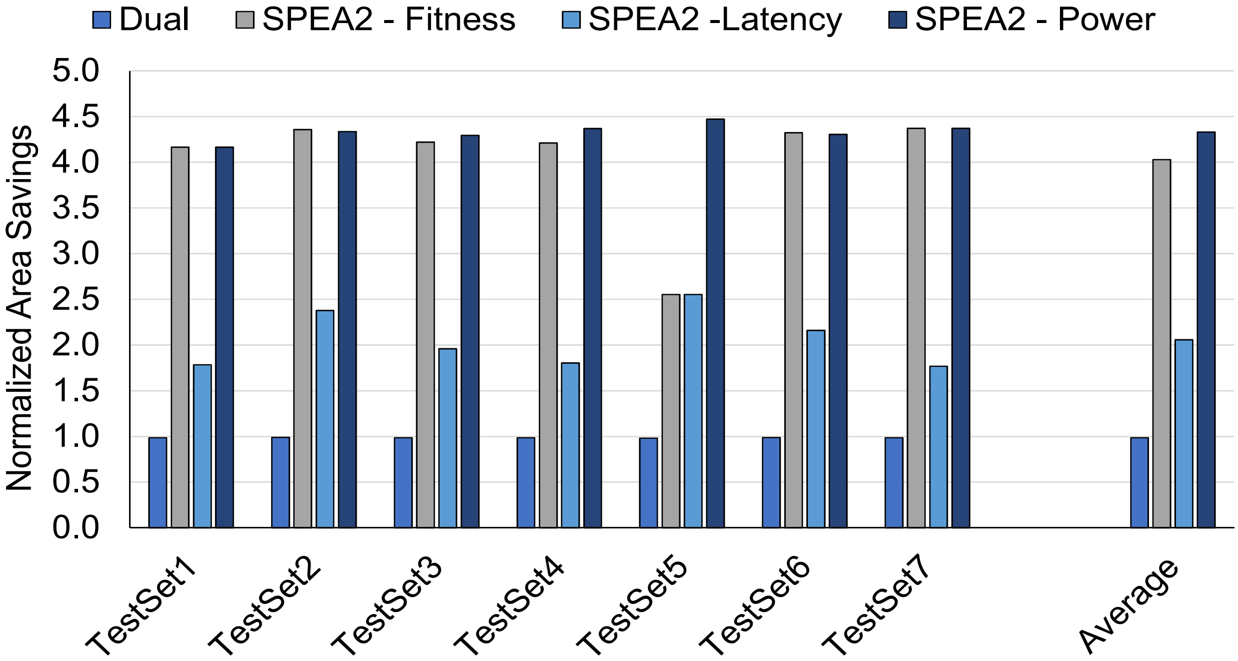


Figure 6.13: Improvement of NoC area of Dual and SPEA2 optimal configurations normalized to the homogeneous baseline configuration.

6.4.2 Average Network Latency

Figure 6.14 shows the improvements in NoC average packet latency of all the configurations normalized to the homogeneous baseline configuration. Regarding the optimal configurations obtained from SPEA2, the configuration with the best latency provides better improvement than other configurations, except for TestSet2, and on average provides 18% improvement. Generally, the configuration with optimal power does not provide any improvement, while the configuration with the best fitness improves the performance of the NoC, except for

TestSet2, TestSet6, and TestSet7. In both TestSet2 and TestSet6, the GPU benchmark has a higher injection rate compared to the other. For TestSet7, the optimal fitness configuration is the same as the optimal power configuration; hence it does not improve the latency. On the other hand, Dual configuration slightly improves the latency for all except three TestSets, 2, 4, and 7, but has less improvement than the SPEA2 optimal fitness and optimal latency configurations.

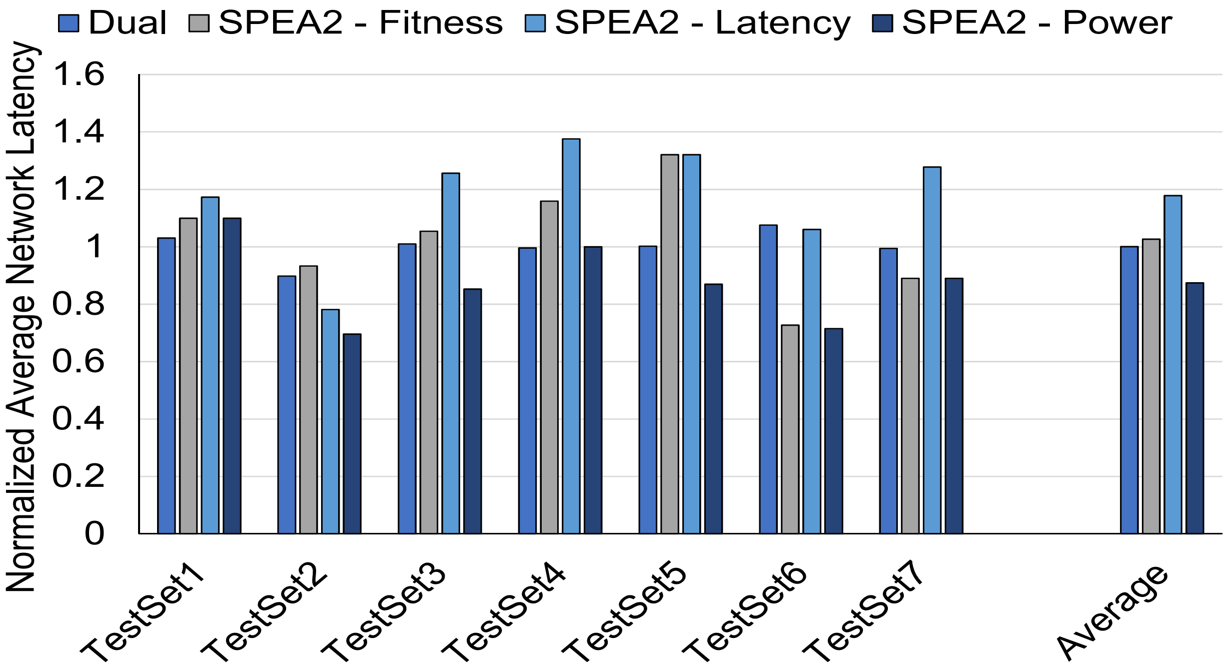


Figure 6.14: Improvements in NoC latency using Dual and SPEA2 optimal configurations normalized to the baseline.

6.4.3 Percentage of Non-Blocking

According to Figure 6.15, both Homog and Dual configurations have 100% of average buffer non-blocking, while SPEA2 optimal configurations vary. Generally, the SPEA2 optimal latency has a higher percentage than other SPEA2 optimal configurations. On average, the percentage of buffer non-blocking is 72%, 84%, and 70% under SPEA2 optimal fitness, latency, and power configurations, respectively. This again indicates that there are excess buffers in both Homog and Dual configurations since SPEA2 configurations improve the NoC latency while having less percentage of buffer non-blocking.

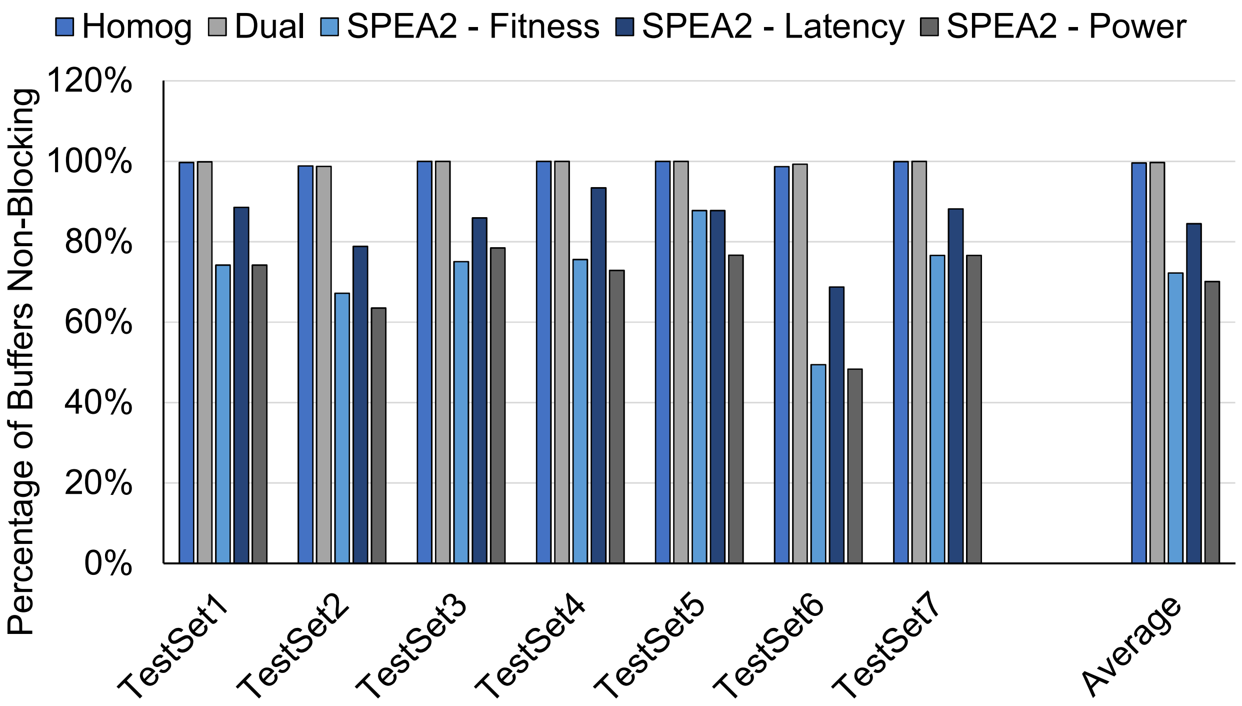


Figure 6.15: Comparison of average percentage of buffers' non-blocking for Homog, Dual, and SPEA2 optimal configurations.

6.4.4 NoC Power

For the NoC power savings, as in Figure 6.16, all the SPEA2 configurations save more power than the baseline. The savings is up to 4.64x, 2.17x, and 5.04x on average using fitness, latency, and power optimal SPEA2 configuration, respectively. Generally, fitness and power optimal configurations save more power than the latency optimal configuration, except for TestSte5 where the optimal fitness solution is the same as the optimal latency. However, Dual configuration does not provide any power savings. This is mainly due to the considerable reduction in NoC area obtained from the proposed method, represented as the buffers of the NoC. As shown in Figure 6.17, the percentage of power consumed in the buffer reaches 92% in Homog and Dual configurations while it is decreased to 66%, 83%, and 61% using SPEA2 optimal fitness, latency, and power configurations, respectively.

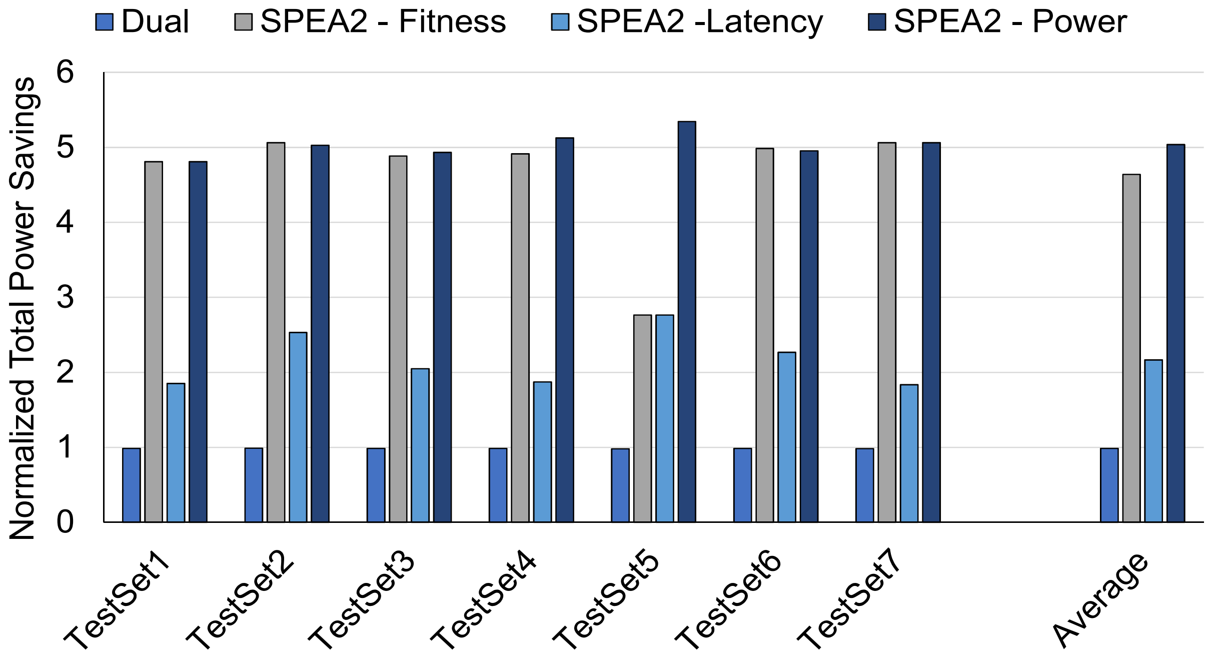


Figure 6.16: NoC power consumption savings using Dual and SPEA2 optimal configurations normalized to the baseline.

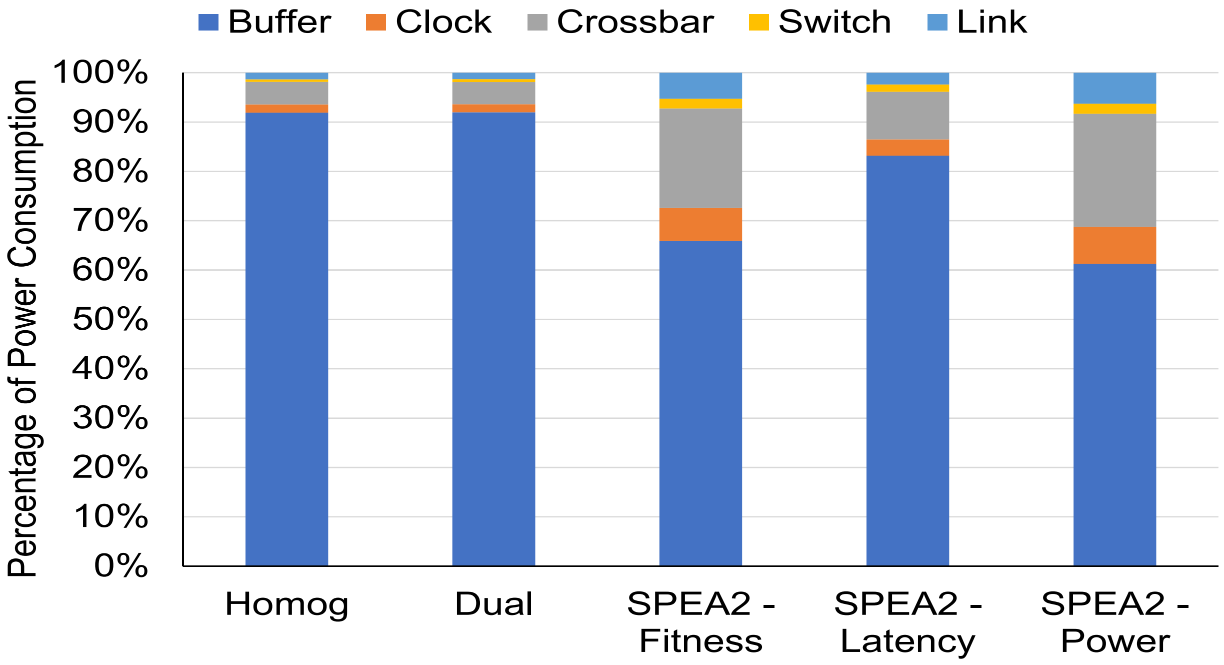


Figure 6.17: Comparison of NoC power consumption break-down for the average of TestSets under different NoC configurations.

6.4.5 NoC Throughput

The improvement in NoC throughput provided by the different configurations compared to the baseline is shown in Figure 6.18. Generally, SPEA2 optimal latency configuration has better throughput than other SPEA2 optimal configurations. It is, also, better than the homogeneous baseline configuration except for TestSet1 and TestSet2, where Dual outperforms it. On average, SPEA2 optimal latency provides 2.7% improvement in NoC throughput compared to the baseline, and Dual provides 1.5% improvement. SPEA2 optimal power generally degrades the throughput except for TestSet7. The effect of SPEA2 optimal fitness on the NoC throughput varies across the TestSets.

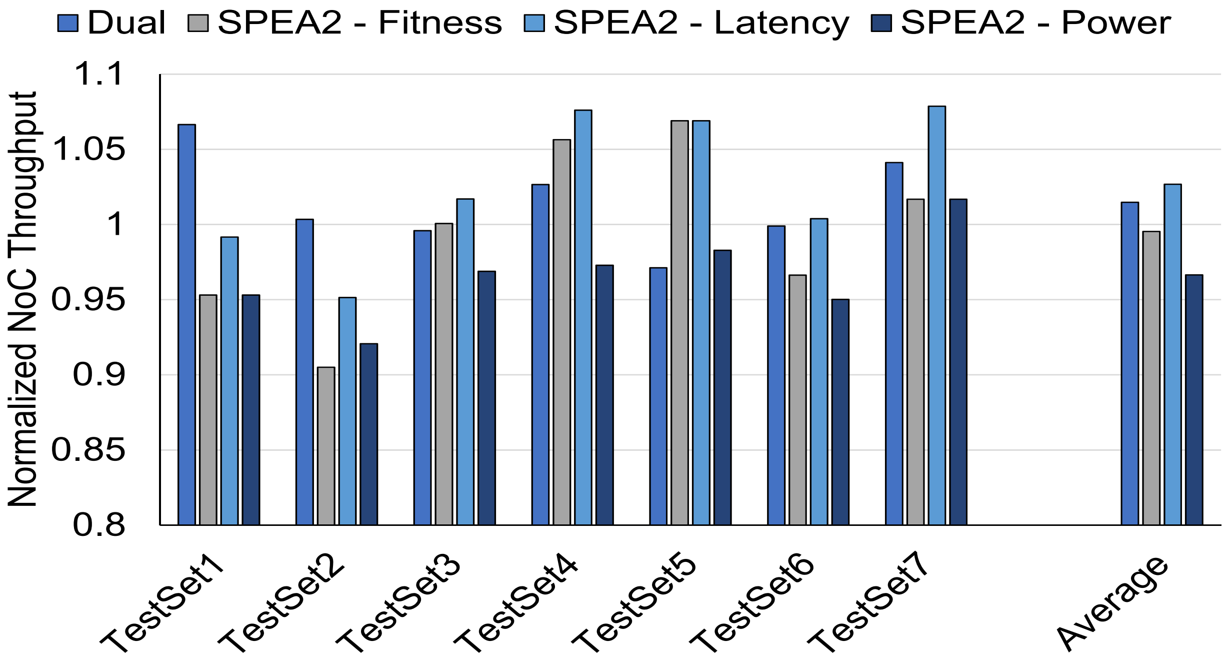


Figure 6.18: Improvement of NoC throughput of Dual and SPEA2 optimal configurations normalized to the homogeneous baseline configuration.

6.4.6 Average Speedup

As shown in Figure 6.19, both Dual and SPEA2 optimal latency configurations slightly improve the CPU speedup for all TestSets, except TestSet2 for the optimal latency, and on average has 0.3% and 1% improvement compared to the baseline, respectively. Similarly, SPEA2 optimal fitness slightly improves the CPU speedup compared to the baseline except for TestSets 2 and 6. SPEA2 optimal power, on the other hand, degrades the CPU speedup compared to the baseline except for TestSets 1, and 7. In these TestSets, the optimal fitness configuration happens to be the same as the optimal power configuration. For the GPU speedup, shown in Figure 6.20, Dual slightly improves the GPU speedup by an average of 3.9%. Both, SPEA2 optimal fitness and power degrades the GPU speedup for all the TestSets, except TestSet5 where the fitness and the latency optimal configurations are the same. SPEA2 optimal latency configuration, on the other hand, improves the GPU speedup for all except two TestSets, 6 and 7, and on average has 2.34% improvement. As shown in Figure 6.21, Dual configuration slightly improves the overall system speedup with an average of 2%. While SPEA2 optimal latency configuration has an average improvement of 1.6%, both the SPEA2 optimal fitness and power configurations degrade the overall system speedup.

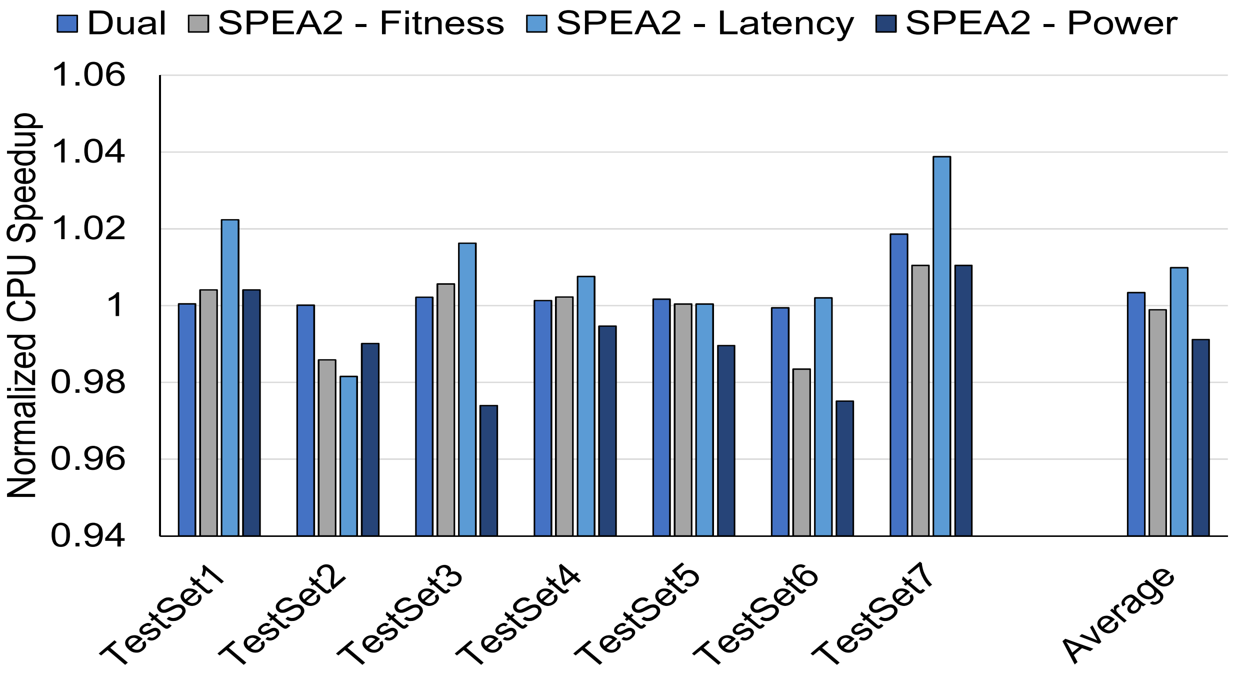


Figure 6.19: Average CPU speedup of Dual and SPEA2 optimal configurations normalized to the homogeneous baseline configuration.

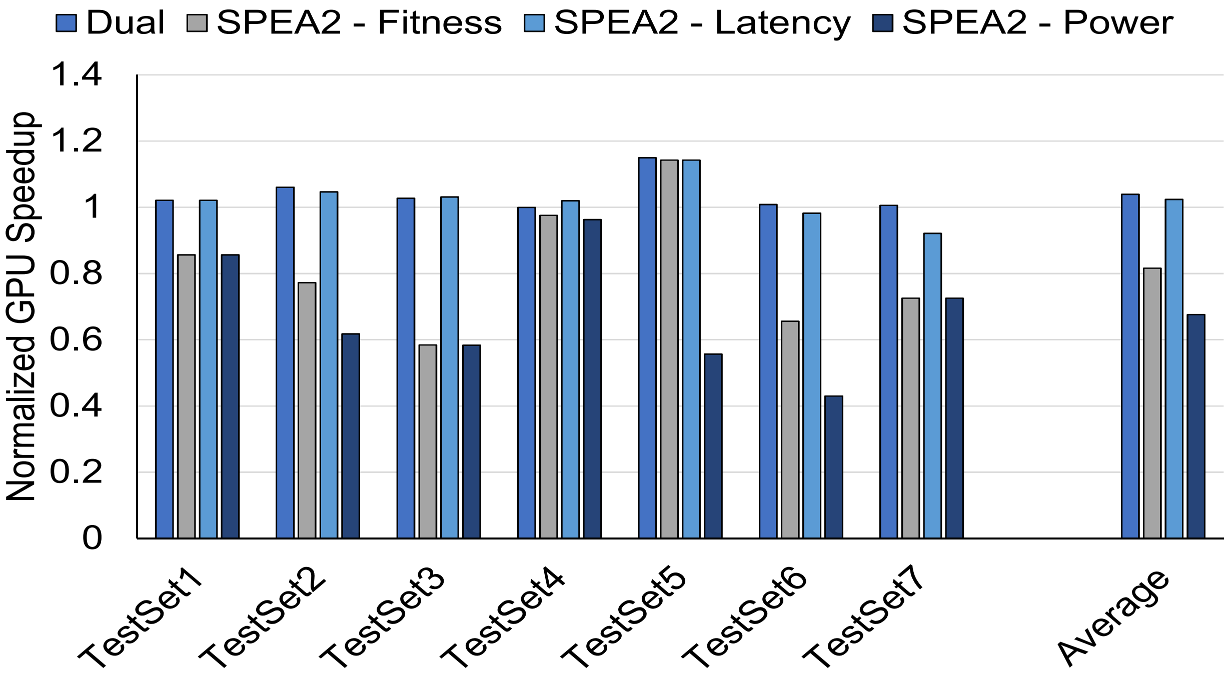


Figure 6.20: Average GPU speedup of Dual and SPEA2 optimal configurations normalized to the homogeneous baseline configuration.

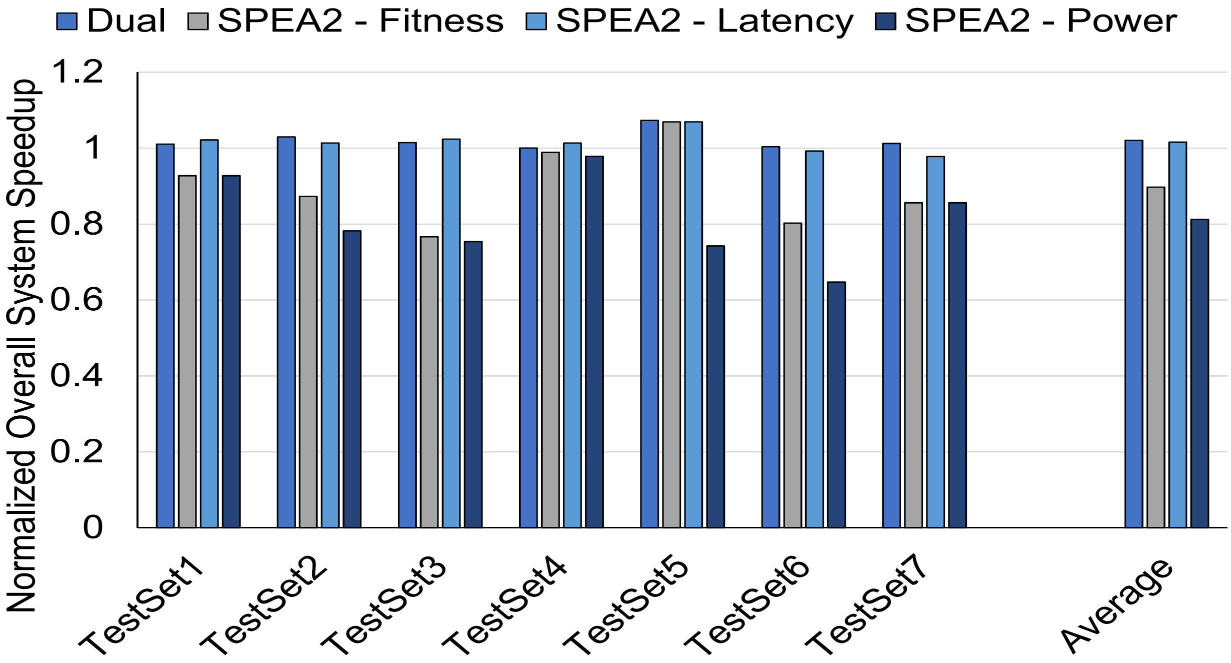


Figure 6.21: Overall speedup of the system gained by using Dual and SPEA2 optimal configurations normalized to the baseline.

6.5 SPEA2-BW NoC Design for Four Sub-Problems

The gem5-gpu simulator does not support heterogeneous bandwidth NoC. Alternatively, the latency of the links was adjusted to reflect the different bandwidths and simulate an NoC with heterogeneous bandwidths. The initial traffic trace was obtained as explained in Section 6.2 using the baseline network configurations in Table 6.3. After feeding the traffic trace to the proposed optimizer based on SPEA2 that supports bandwidth optimization (SPEA2-BW), a non-dominated Pareto optimal set of solutions is generated. Each of these solutions represents an NoC design with optimal mapping of PE, optimal assignment of buffer size and virtual channels per outport, and optimal bandwidth for each link. Three solutions out of the Pareto-optimal set were considered for comparison: 1) The solution with the best performance (S-BW - Latency), 2) The solution with the best power consumption (S-BW - Power), and 3) The solution with the best fitness as in (5.2) (S-BW - Fitness).

The chosen SPEA2-BW solutions were rerun on gem5-gpu while fixing the bandwidth of the links to 32B and setting the latency of each link to reflect the intended bandwidth. For example, if the link bandwidth according to the configuration should be 16B, then the link latency is set to 2 cycles. Similarly, if the link bandwidth is supposed to be 32B, the latency is set to 1 cycle. This approach was also used for the DSENT power tool to get the power of the simulated NoC.

A distribution of the heterogeneous bandwidth in the different SPEA2-BW optimal configurations of the different TestSets is shown in Figure 6.22. Generally, the optimal latency configuration has a higher percentage of the higher bandwidth links than the other optimal configurations. Similarly, the optimal power configuration has the least percentage of the higher bandwidth links compared to the other optimal configurations.

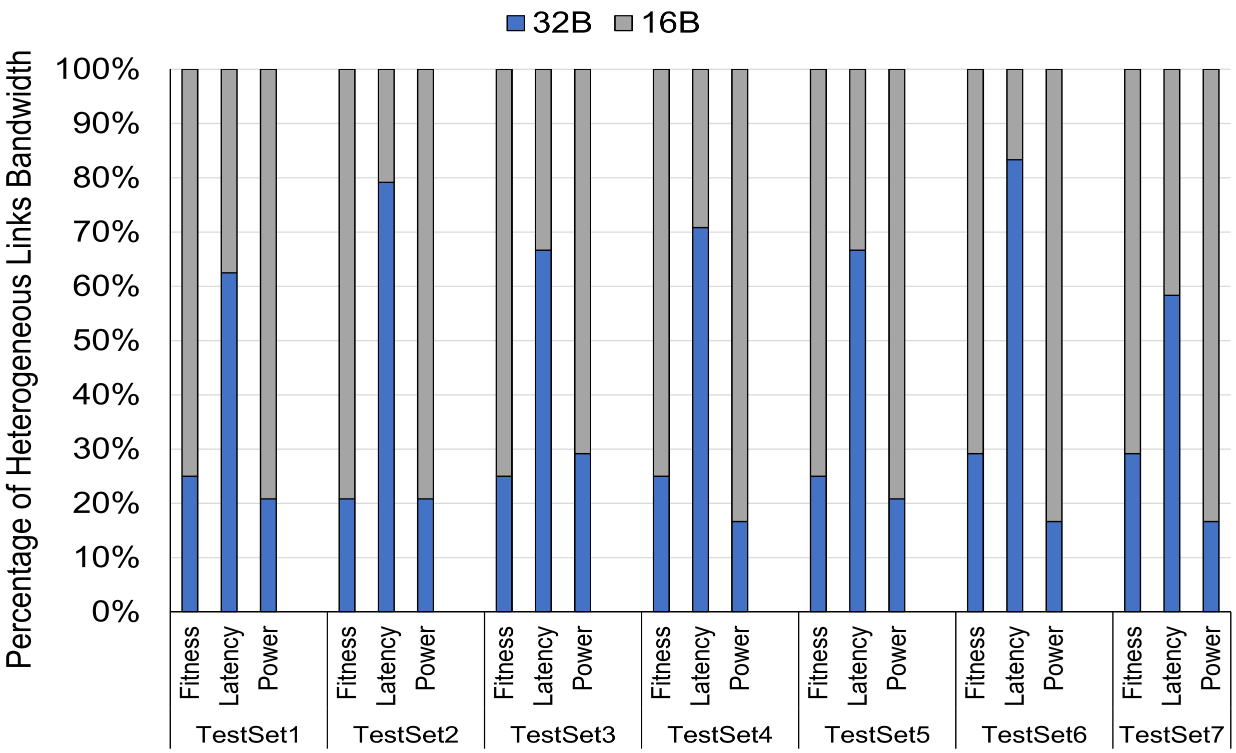


Figure 6.22: Links' bandwidth distribution using different SPEA2-BW optimal configurations for the different TestSets.

6.5.1 Total Area

As shown in Figure 6.23, Dual does not provide any improvement in the area, as expected. Fitness and power optimal configurations improve the area in all TestSets and on average provide 53% and 55% area savings, respectively. On the other hand, the latency optimal configuration vary, it provides improvement for some TestSets while degrades the others, and on average provides only 4% improvement in the area.

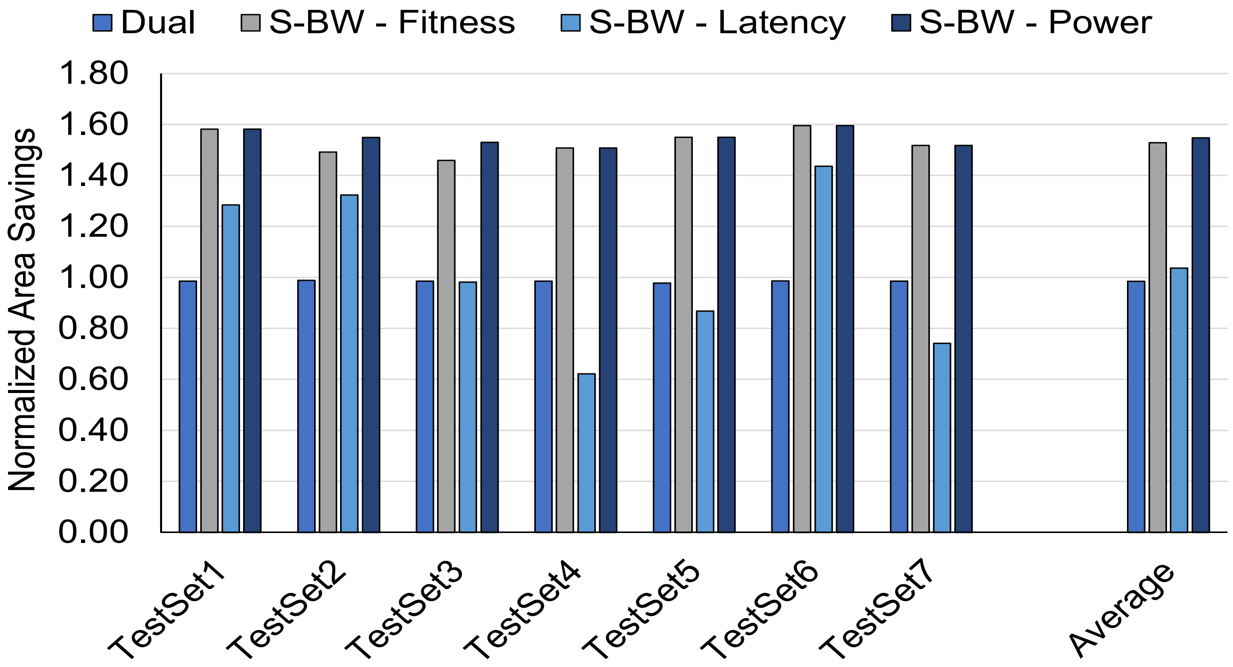


Figure 6.23: Improvement of NoC area of Dual and SPEA2-BW optimal configurations normalized to the homogeneous baseline configuration.

6.5.2 Average Network Latency

The improvements in NoC average packet latency of all the configurations normalized to the homogeneous baseline configuration is shown in Figure 6.24. Dual improves all but three TestSets, 2, 4, and 7, with an average improvement of 0.08% only. SPEA2 optimal latency configuration improves the NoC latency in all the TestSets and on average has 54% improvement. SPEA2 optimal fitness configuration improves all except three TestSets 5, 6, and 7, but on average degrades the NoC performance by 2.7%. SPEA2 optimal power configuration degrades all except TestSets 1 and 4. For all the TestSets except TestSet2 and TestSet3, the SPEA2 optimal fitness and power configurations happen to be the same.

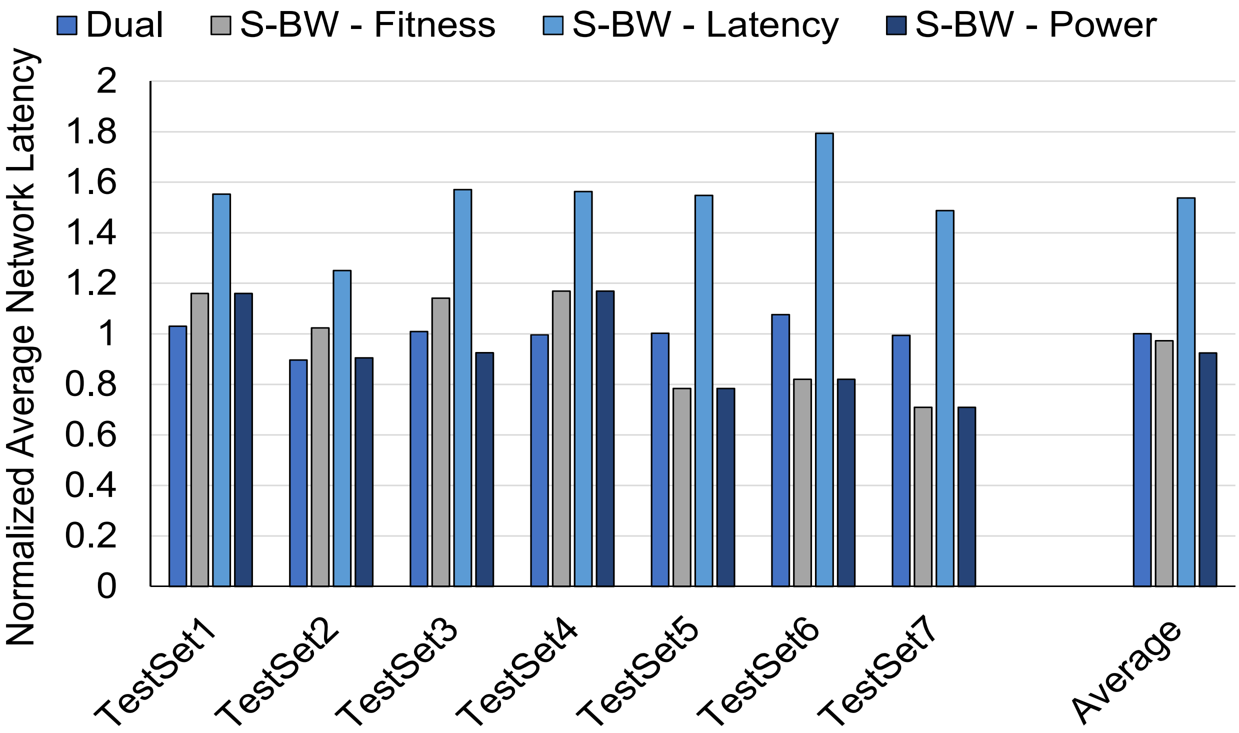


Figure 6.24: Improvements in NoC latency using Dual and SPEA2-BW optimal configurations normalized to the baseline.

6.5.3 Percentage of Non-Blocking

According to Figure 6.25, both Homog and Dual have almost 100% of average buffers non-blocking. SPEA2 optimal configurations vary with an average of 80%, 86%, 80% for the optimal fitness, latency, and power, respectively. While SPEA2 optimal latency manages to improve the network latency while decreasing the percentage of buffers non-blocking, this indicates there are excess buffers in the Homog and Dual configurations.

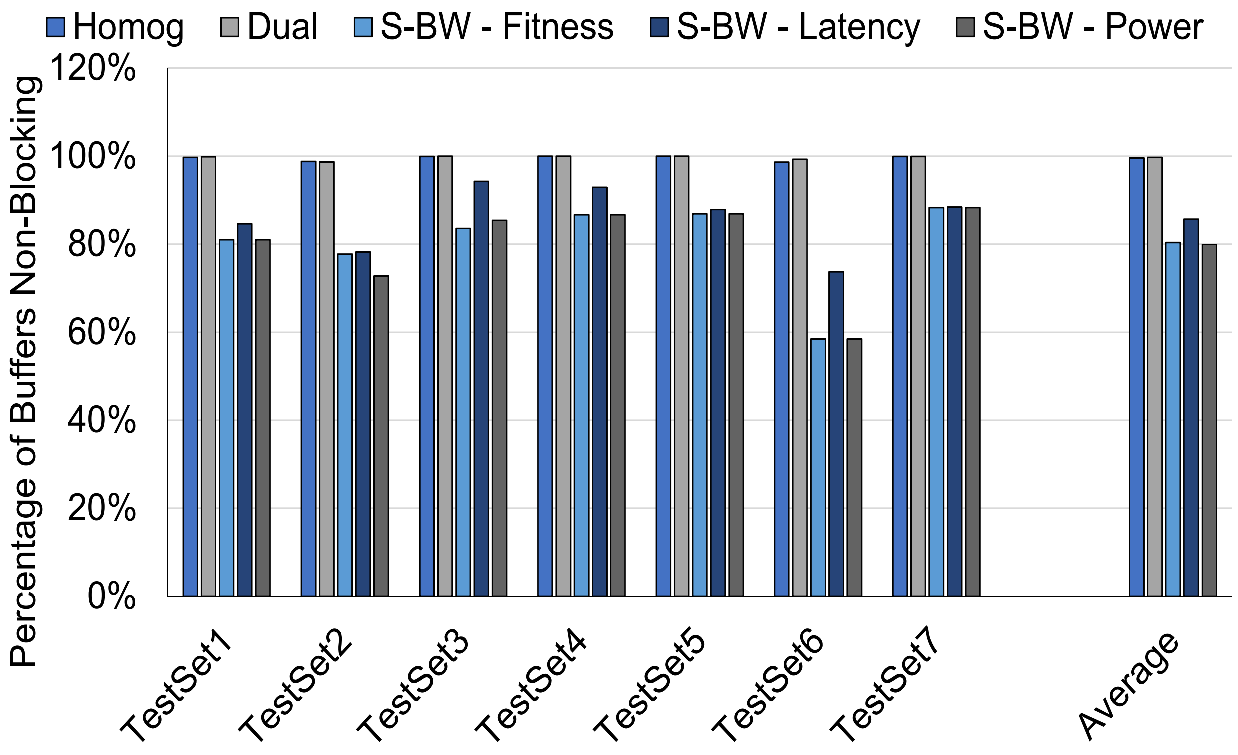


Figure 6.25: Comparison of average percentage of buffers' non-blocking for Homog, Dual, and SPEA2-BW optimal configurations.

6.5.4 NoC Power

For the NoC power savings, as in Figure 6.26, Dual provides no power savings. Both SPEA2 optimal fitness and power provides power savings for all the TestSets compared to the homogeneous baseline configuration with an average of 2.5x and 2.55x, respectively. SPEA2 optimal latency improves the power savings for all except TestSet 4 and 7 and has an average improvement of 45%. As shown in Figure 6.27, the contribution of the buffer to the total NoC power is reduced from 92% in Homog and Dual configurations to 63%, 82%, and 63% in SPEA2 optimal fitness, latency and power configurations.

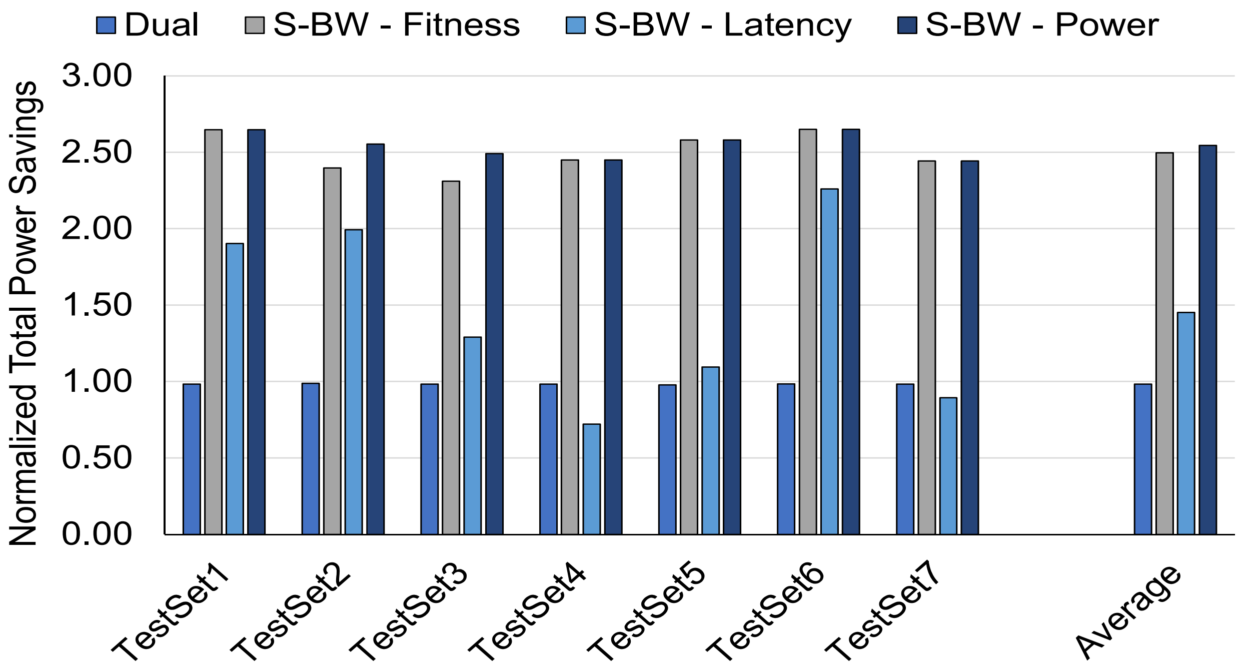


Figure 6.26: NoC power consumption savings using Dual and SPEA2-BW optimal configurations normalized to the baseline.

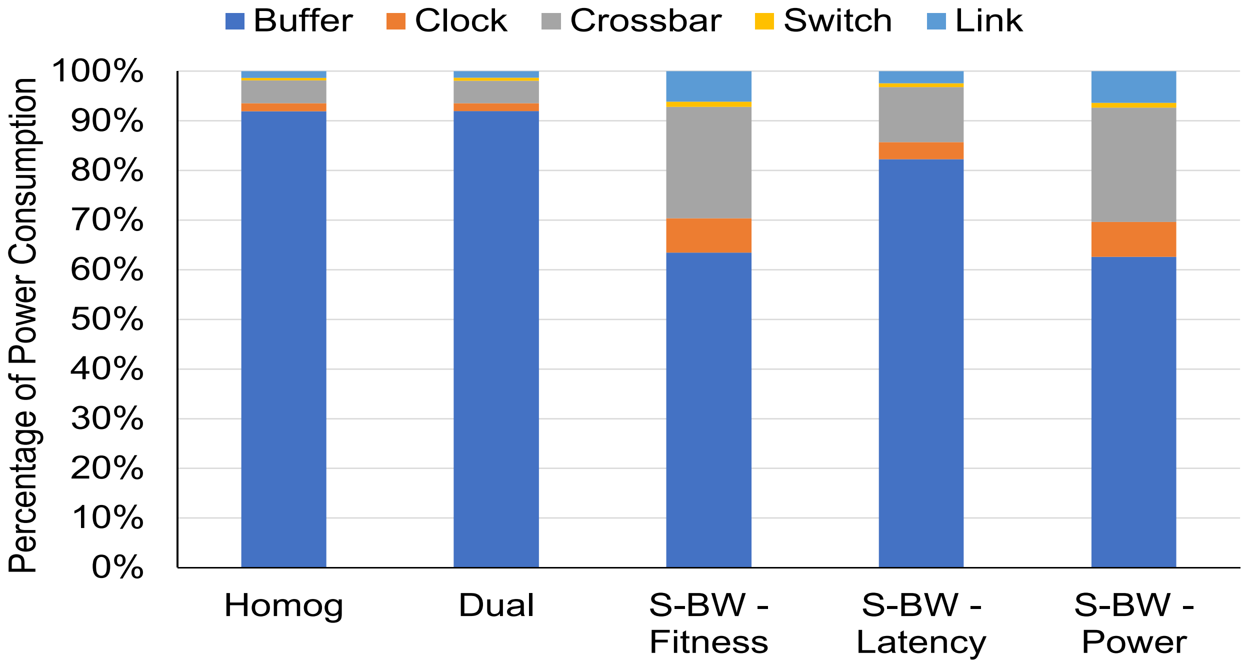


Figure 6.27: Comparison of NoC power consumption break-down for the average of TestSets under different NoC configurations.

6.5.5 NoC Throughput

The improvement in NoC throughput provided by the different configurations compared to the baseline is shown in Figure 6.28. Generally, SPEA2 optimal latency improves the NoC throughput in all TestSets except TestSet2 with an average improvement of 6%. On the other hand, SPEA2 optimal power slightly degrades the NoC throughput in all TestSets, except TestSet1 and 4, with an average degradation of 1%. Similarly, SPEA2 optimal fitness degrades all except TestSets 1, 3, and 4 with average NoC throughput degradation of 0.5%. Dual configuration effect on NoC throughput varies through the TestSets and has an average improvement of 0.43%.

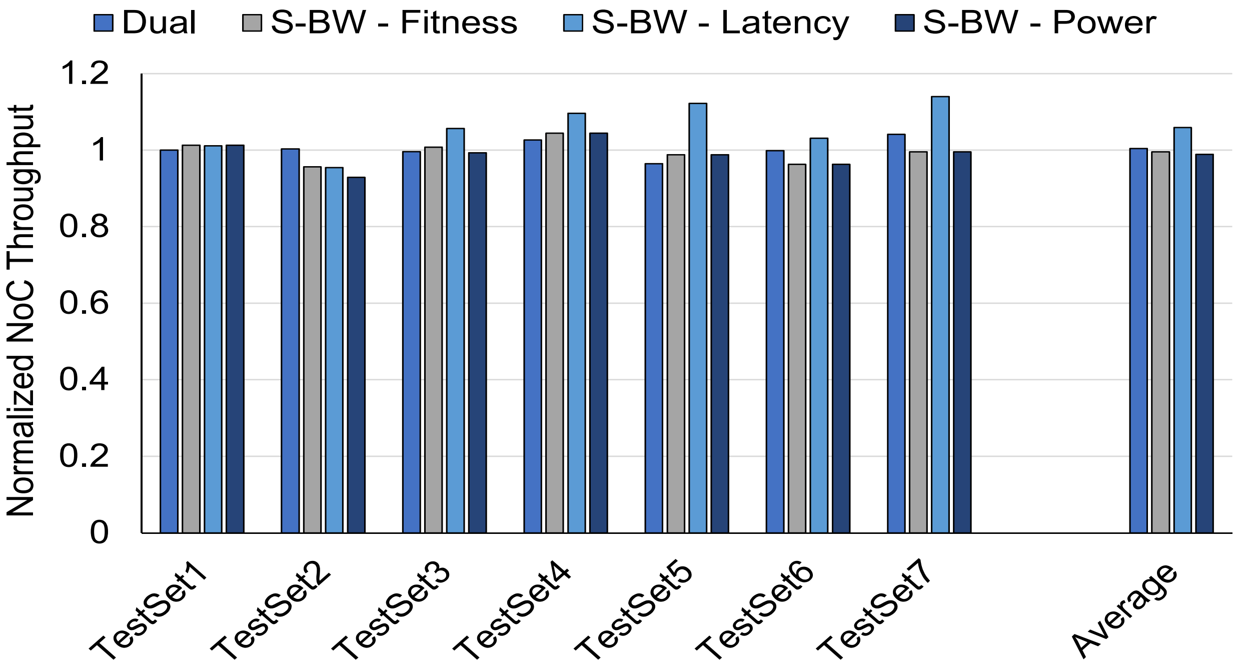


Figure 6.28: Improvement of NoC throughput of Dual and SPEA2-BW optimal configurations normalized to the homogeneous baseline configuration.

6.5.6 Average Speedup

As shown in Figure 6.29, Dual maintains the CPU speedup with an average improvement of 0.35%. SPEA2 optimal latency improves the CPU speedup compared to the homogeneous baseline for all except TestSet2 and has an average improvement of 1.7%. SPEA2 optimal fitness and power only improve the CPU speedup of TestSets 1,3, and 4, with an average degradation of 1%. Similarly, For the GPU speedup, shown in Figure 6.30, Dual maintains the GPU speedup and slightly improves it on average by 4%. SPEA2 optimal latency improves the GPU speedup for all the TestSets with an average improvement of 25%. Both SPEA2 optimal fitness and power generally degrade the GPU speedup compared to the homogeneous baseline with an average degradation of 9.6% and 21%, respectively. On average, both Dual and SPEA2 optimal latency improve the overall system speedup, as shown in Figure 6.31, by 2.1% and 12.2%, respectively. On the other hand, SPEA2 optimal fitness and power degrade the overall system speedup by an average of 6.2% and 12.2%, respectively.

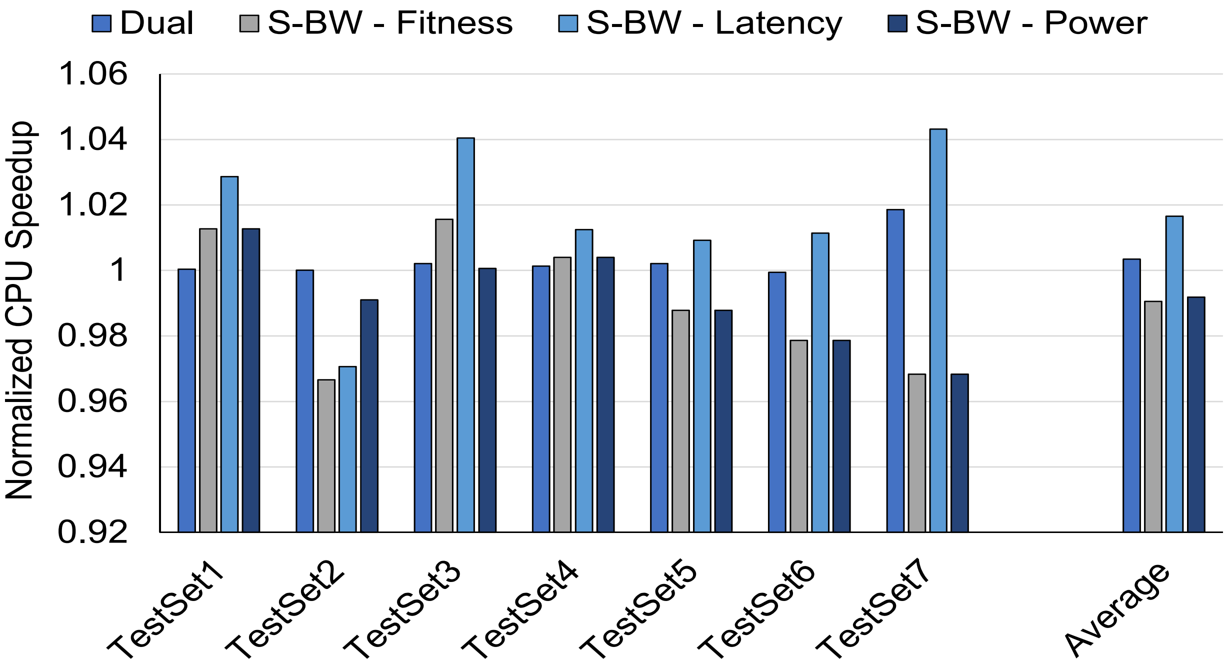


Figure 6.29: Average CPU speedup of Dual and SPEA2-BW optimal configurations normalized to the homogeneous baseline configuration.

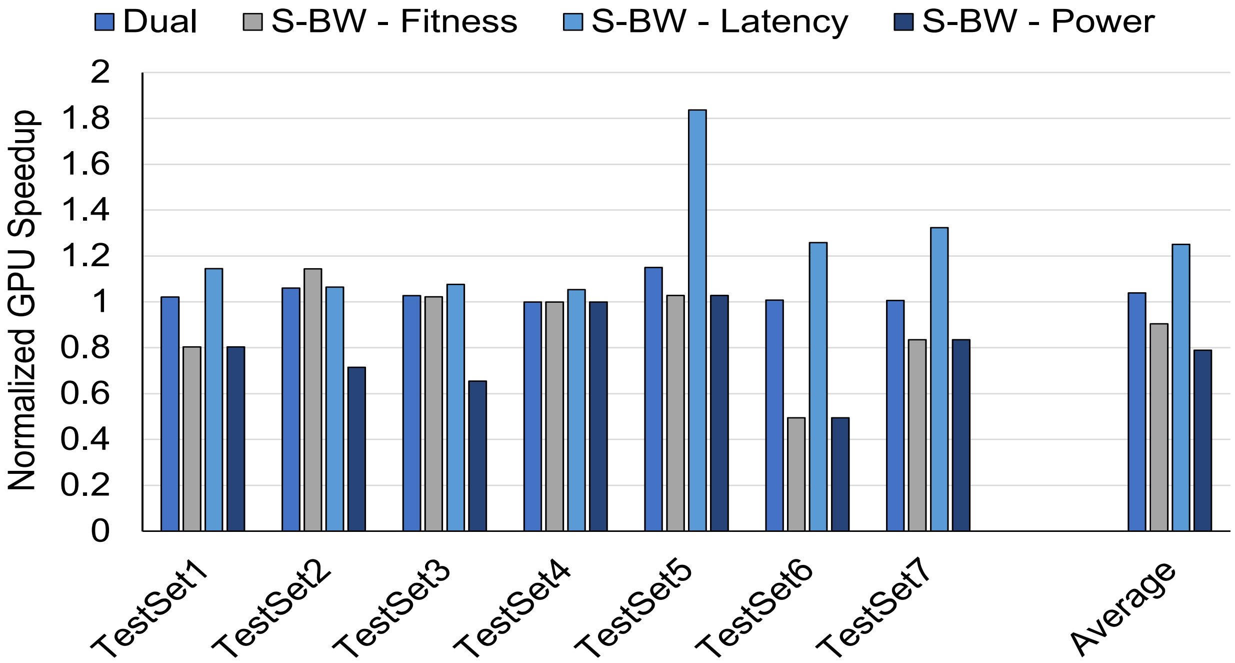


Figure 6.30: Average GPU speedup of Dual and SPEA-BW optimal configurations normalized to the homogeneous baseline configuration.

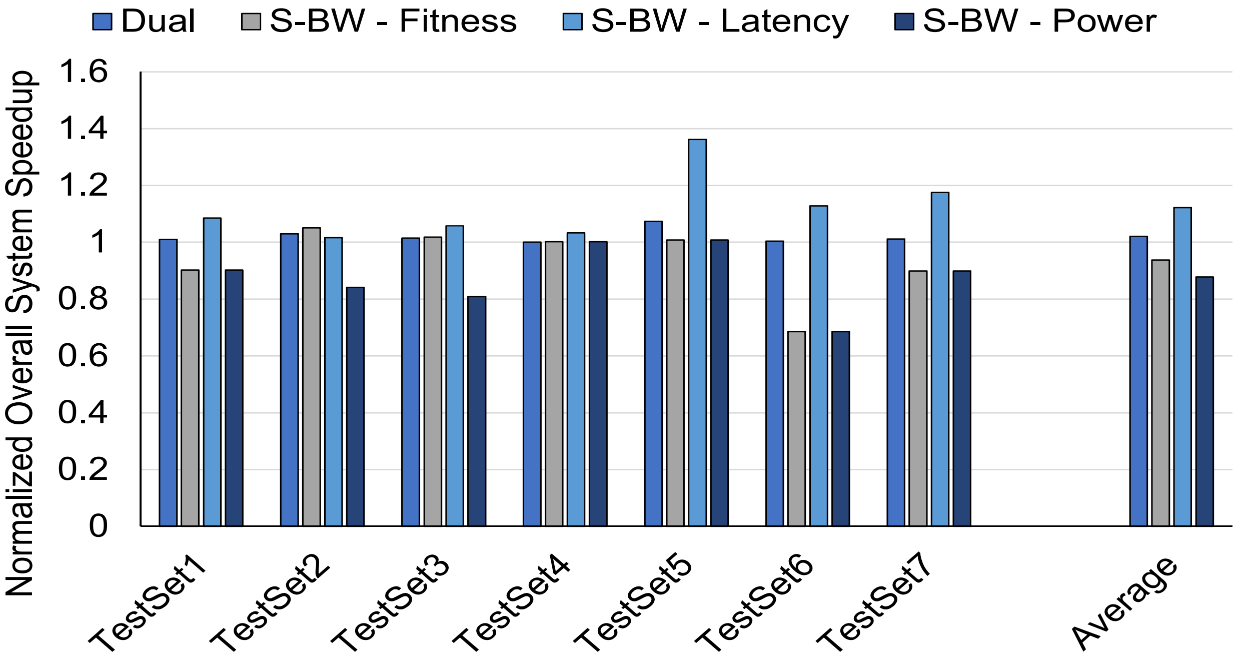


Figure 6.31: Overall speedup of the system gained by using Dual and SPEA2-BW optimal configurations normalized to the baseline.

6.6 Summary

The improvement gained by the different proposed methods, GA, SPEA2, and SPEA2-BW, is shown in Figure 6.32, and summarized in Table 6.6. Since the proposed GA provides NoC design that optimizes NoC performance, the optimal latency configurations obtained from SPEA2 and SPEA2-BW are used for comparison. The comparison includes the effect of each method on different criteria averaged over the seven TestSets.

The first observation is that when power is included in the optimization as in SPEA2 and SPEA-BW, the power and area are improved more than GA. When comparing GA and SPEA2, the general notice is that GA outperforms SPEA2 in other criteria like network latency and speedups. While including the heterogeneous bandwidth in NoC design as in SPEA2-BW further improves all other criteria compared to GA. The GPU in particular benefits from the heterogeneous BW, hence implicitly enhances the overall system speedup.

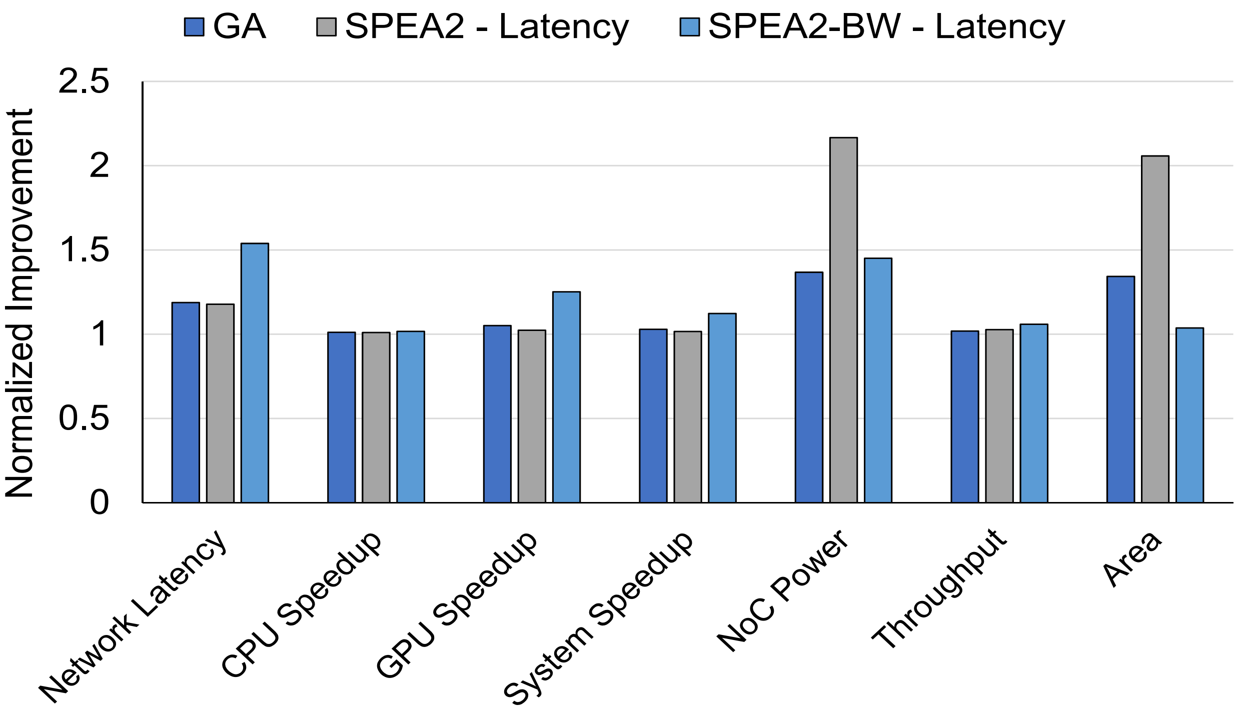


Figure 6.32: Comparison of the average normalized improvement for different criteria gained by GA and the latency optimal configuration of SPEA2 and SPEA2-BW.

Similarly, SPEA2-BW outperforms SPEA2 when comparing the obtained power optimal configurations, as in Figure 6.33, in all criteria except the power and area.

Table 6.6: Comparison of the Proposed Methods

Improvement	GA	SPEA2			SPEA2-BW		
		Fitness	Latency	Power	Fitness	Latency	Power
NoC Latency	19%	2.6%	18%	-12.5%	-2.7%	54%	-7.5%
NoC Throughput	2%	-0.5%	2.7%	-3.4%	-0.5%	6%	-1%
CPU Speedup	1%	-0.1%	1%	-0.9%	-0.9%	1.7%	-0.8%
GPU Speedup	5.15%	-18.4%	2.34%	-32%	-9.6 %	25%	-21%
System Speedup	3%	-10.2%	1.6%	-19%	-6.2%	12.2%	-12.2%
NoC Area	34%	4.03x	2.06x	4.33x	53%	4%	55%
NoC Power	37%	4.64x	2.17x	5.04x	2.5x	45%	2.55x

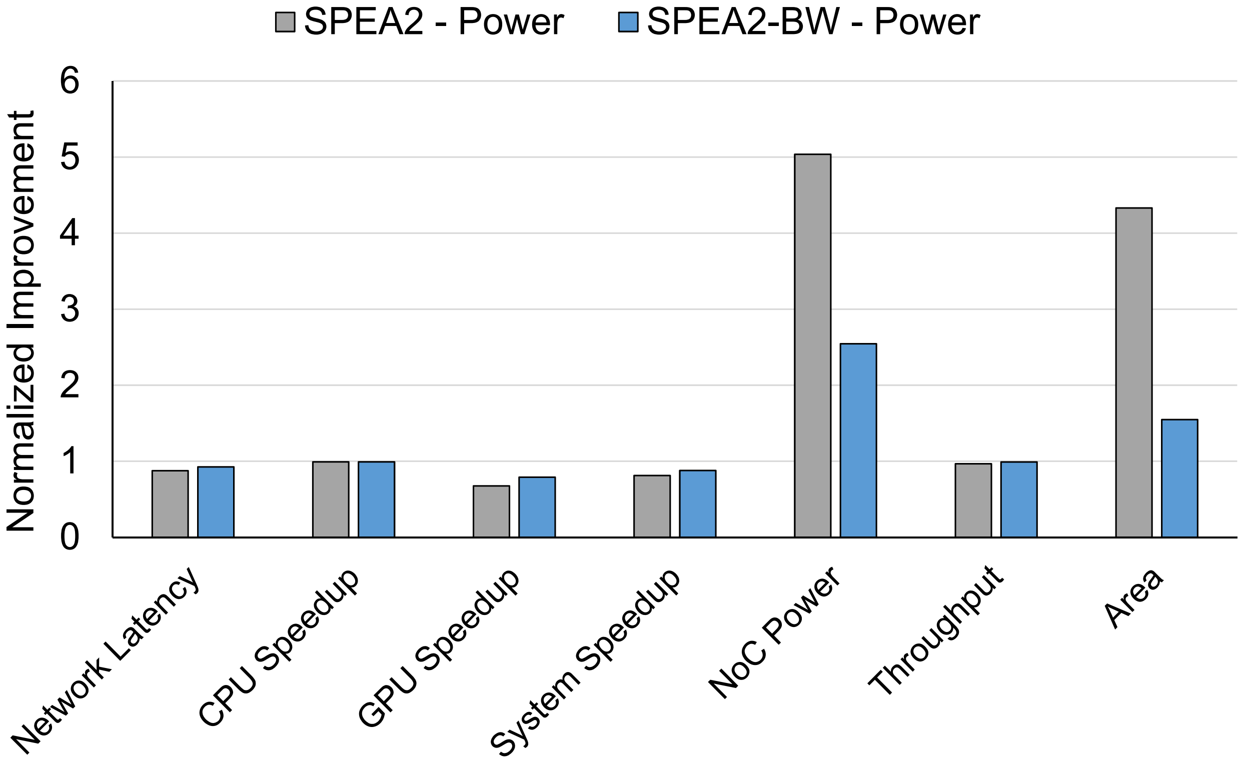


Figure 6.33: Comparison of the average normalized improvement for different criteria gained by the power optimal configuration of SPEA2 and SPEA2-BW.

Chapter 7

Conclusion and Future Work

As GPU becomes a powerful processor that can be used for scientific and general computations, the movement from CPU vs. GPU era to combining the powerful features of both processors becomes a necessity. Many HSA, fused the CPU and GPU on the same chip to utilize both processors. However, combining different processors with diametric network demands places a burden on the common interconnection network along with other different shared resources problem.

This dissertation focused on designing a heterogeneous 2D mesh style NoC for fused CPU-GPU architecture. In this regards, heterogeneity was explored on routers and links of the NoC. The heterogeneity was investigated on the port level of the NoC's routers, where arbitrary virtual channels and arbitrary buffer sizes were considered for each port of each router. Also, different bandwidth of each link of the NoC was considered. Moreover, the placement or the mapping of the heterogeneous processing elements (CPU cores, GPU cores, MC, and shared caches) to the mesh NoC was explored. All these sub-problems of heterogeneous NoC design were considered simultaneously as one optimization problem.

A performance model which supports arbitrary buffers based on G/G/1 queuing theory model was presented. This model was extended to support different virtual channels per port. Also, an approximation of link bandwidth in terms of link latency was added. Moreover, an activity-based power model was proposed using the same queueing model. These analytical models were used to obtain a measure of the NoC performance and power within the proposed optimization methods.

First, this multi-dimensional heterogeneous NoC design was tackled with a method based on GA. The objective was to get a design with optimal performance (average packet latency) that determines the placement of the PEs within the mesh, the buffer size, and virtual channels configurations. The results demonstrate that this method can increase network performance by 19% on average and reduce the area by 34% on average while enhancing the overall speedup of the system on average by 3%.

Second, an optimization method based on SPEA2 to explore the design space of PEs placements and NoC configurations (the buffer size and the number of virtual channels) was proposed. This method produces Pareto-optimal designs that satisfy two objectives; performance and power of NoC. When simulating the optimal configurations, results show that the NoC performance can be improved by 18% while minimizing the power consumption by at least 2.17x and maintaining the overall system performance.

Finally, the optimization method based on SPEA2 was extended to include the heterogeneous link bandwidth and obtaining a Pareto-optimal set. The results show that including heterogeneous bandwidth enhances the performance of the NoC and the overall system speedup in particular the GPU speedup. On average, the improvement in NoC performance reaches up to 54%, GPU speedup 25%, and overall system speedup 12.2%. Compared to the former SPEA2 based method, the improvement in power savings and area was less. Nevertheless, the power savings reached at least 45% on average.

Instead of relying on simulation to explore a limited set of designs, the proposed optimization methods help explore the large design space of heterogeneous NoC, solving different sub-problems simultaneously. Also, the SPEA2 based optimization methods give the NoC designer a set of design choices to choose from depending on the target architecture goals; power or performance.

7.1 Future Work

The work in this dissertation can be extended in different directions. First, 3D NoCs become popular for their performance, flexibility, and throughput. This work can be extended to design a 3D style mesh NoC. Thermal considerations should be added as a third objective of the optimization methods. Moreover, different multi-objective optimization methods can be considered. Machine learning can be used to evaluate NoC design as an alternative for the analytical models.

Utilizing the CPU and GPU core can be further improved by proper mapping of different benchmarks to the appropriate core type. This mapping is dependent on the nature of the benchmarks along with the capabilities of the processing cores. Also, the heterogeneous NoC design plays a key factor in benchmarks mapping that can be further investigated. Furthermore, partitioning of the tasks within the benchmark to the appropriate core type can be considered. Including the benchmarks or application mapping and partitioning in the NoC design means adding the performance and power of the processing cores as measures to the heterogeneous NoC design.

The focus of this dissertation was on the design time of the NoC. Adaptivity of the design during run-time is another dimension that can be pursued. This can include a virtual channels partitioning technique between the CPU and GPU traffic, buffer re-distribution between the ports according to the traffic. Even considering different routing paths between the CPU and GPU traffic is worth exploring.

This dissertation concentrated on the common interconnection network in the fused CPU-GPU architecture. Additional shared resources can be considered. For example, a heterogeneous memory architecture can be investigated.

Bibliography

[1] ORNL Titan Supercomputer. `https://www.olcf.ornl.gov/olcf-resources/compute-systems/titan/`.

[2] N. Agarwal, T. Krishna, L.-S. Peh, and N. K. Jha. Garnet: A detailed on-chip network model inside a full-system simulator. In *Performance Analysis of Systems and Software, 2009. ISPASS 2009. IEEE International Symposium on*, pages 33–42. IEEE, 2009.

[3] J. Alben. Nvidia brings kepler, worlds most advanced graphics architecture, to mobile devices. `https://blogs.nvidia.com/blog/2013/07/24/kepler-to-mobile/`, July 2013.

[4] L. Alhubail and N. Bagherzadeh. Power and performance optimal noc design for cpu-gpu architecture using formal models. In *2019 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 634–637, March 2019.

[5] M. Arjomand and H. Sarbazi-Azad. Power-performance analysis of networks-on-chip with arbitrary buffer allocation schemes. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 29(10):1558–1571, 2010.

[6] G. Ascia, V. Catania, and M. Palesi. A multi-objective genetic approach to mapping problem on network-on-chip. *J. UCS*, 12(4):370–394, 2006.

[7] A. Bakhoda, G. L. Yuan, W. W. Fung, H. Wong, and T. M. Aamodt. Analyzing cuda workloads using a detailed gpu simulator. In *Performance Analysis of Systems and Software, 2009. ISPASS 2009. IEEE International Symposium on*, pages 163–174. IEEE, 2009.

[8] Y. Ben-Itzhak, I. Cidon, and A. Kolodny. Optimizing heterogeneous noc design. In *Proceedings of the International Workshop on System Level Interconnect Prediction*, pages 32–39. ACM, 2012.

[9] C. Bienia, S. Kumar, J. P. Singh, and K. Li. The parsec benchmark suite: Characterization and architectural implications. In *Proceedings of the 17th international conference on Parallel architectures and compilation techniques*, pages 72–81. ACM, 2008.

[10] 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.

[11] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S.-H. Lee, and K. Skadron. Rodinia: A benchmark suite for heterogeneous computing. In *Workload Characterization, 2009. IISWC 2009. IEEE International Symposium on*, pages 44–54. Ieee, 2009.

[12] M. Črepinšek, S.-H. Liu, and M. Mernik. Exploration and exploitation in evolutionary algorithms: A survey. *ACM Computing Surveys (CSUR)*, 45(3):35, 2013.

[13] M. Daga, A. M. Aji, and W.-c. Feng. On the efficacy of a fused cpu+ gpu processor (or apu) for parallel computing. In *2011 Symposium on Application Accelerators in High-Performance Computing*, pages 141–149. IEEE, 2011.

[14] B. Dally. Project denver processor to usher in new era of computing. `https://blogs.nvidia.com/blog/2011/01/05/project-denver-processor-to-usher-in-new-era-of-computing/`, January 2011.

[15] W. J. Dally. Virtual-channel flow control. *IEEE Transactions on Parallel and Distributed systems*, 3(2):194–205, 1992.

[16] J. Fang, Z.-Y. Leng, S.-T. Liu, Z.-C. Yao, and X.-F. Sui. Exploring heterogeneous noc design space in heterogeneous gpu-cpu architectures. *Journal of Computer Science and Technology*, 30(1):74–83, 2015.

[17] M. Gulati and N. Bagherzadeh. Performance study of a multithreaded superscalar microprocessor. In *High-Performance Computer Architecture, 1996. Proceedings., Second International Symposium on*, pages 291–301. IEEE, 1996.

[18] F. Héliodore, A. Nakib, B. Ismail, S. Ouchraa, and L. Schmitt. *Metaheuristics for Intelligent Electrical Networks*, volume 10. John Wiley & Sons, 2017.

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

[20] P.-C. Hu and L. Kleinrock. An analytical model for wormhole routing with finite size input buffers. In *Teletraffic Science and Engineering*, volume 2, pages 549–560. Elsevier, 1997.

[21] R. K. Jena and G. K. Sharma. A multiobjective evolutionary algorithm-based optimisation model for network on chip synthesis. *International Journal of Innovative Computing and Applications*, 1(2):121–127, 2007.

[22] D. Kanter. Intel's sandy bridge microarchitecture. `https://www.realworldtech.com/sandy-bridge/`, September 2010.

[23] D. Kanter. Intels ivy bridge graphics architecture. `https://www.realworldtech.com/ivy-bridge-gpu/`, April 2012.

[24] A. E. Kiasari, Z. Lu, and A. Jantsch. An analytical latency model for networks-on-chip. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 21(1):113–123, 2013.

[25] 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.

[26] J. Lee, S. Li, H. Kim, and S. Yalamanchili. Adaptive virtual channel partitioning for network-on-chip in heterogeneous architectures. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 18(4):48, 2013.

[27] J. Lee, S. Li, H. Kim, and S. Yalamanchili. Design space exploration of on-chip ring interconnection for a cpu–gpu heterogeneous architecture. *Journal of Parallel and Distributed Computing*, 73(12):1525–1538, 2013.

[28] Z. Li, N. Goswami, and T. Li. Iconn: A communication infrastructure for heterogeneous computing architectures. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, 11(4):42, 2015.

[29] A. K. Mishra, N. Vijaykrishnan, and C. R. Das. A case for heterogeneous on-chip interconnects for cmps. In *ACM SIGARCH Computer Architecture News*, volume 39, pages 389–400. ACM, 2011.

[30] S. Mittal and J. S. Vetter. A survey of cpu-gpu heterogeneous computing techniques. *ACM Computing Surveys (CSUR)*, 47(4):69, 2015.

[31] U. Y. Ogras, P. Bogdan, and R. Marculescu. An analytical approach for network-on-chip performance analysis. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 29(12):2001–2013, 2010.

[32] J. Power, J. Hestness, M. S. Orr, M. D. Hill, and D. A. Wood. gem5-gpu: A heterogeneous cpu-gpu simulator. *IEEE Computer Architecture Letters*, 14(1):34–36, 2015.

[33] J. Power, M. D. Hill, and D. A. Wood. Supporting x86-64 address translation for 100s of gpu lanes. In *High Performance Computer Architecture (HPCA), 2014 IEEE 20th International Symposium on*, pages 568–578. IEEE, 2014.

[34] M. J. Schulte, M. Ignatowski, G. H. Loh, B. M. Beckmann, W. C. Brantley, S. Gurumurthi, N. Jayasena, I. Paul, S. K. Reinhardt, and G. Rodgers. Achieving exascale capabilities through heterogeneous computing. *IEEE Micro*, 35(4):26–36, 2015.

[35] D. Shin and J. Kim. Power-aware communication optimization for networks-on-chips with voltage scalable links. In *Proceedings of the 2nd IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, pages 170–175. ACM, 2004.

[36] H. Singh, M.-H. Lee, G. Lu, F. J. Kurdahi, N. Bagherzadeh, and E. M. Chaves Filho. Morphosys: an integrated reconfigurable system for data-parallel and computation-intensive applications. *IEEE transactions on computers*, 49(5):465–481, 2000.

[37] Stoner. HSA Foundation Presented Deeper Detail on HSA and HSAIL. `http://www.hsafoundation.com/hot-chips-2013-hsa-foundation-presented-deeper-detail-hsa-hsail/`, August 2013.

[38] C. Sun, C.-H. O. Chen, G. Kurian, L. Wei, J. Miller, A. Agarwal, L.-S. Peh, and V. Stojanovic. Dsent-a tool connecting emerging photonics with electronics for opto-electronic networks-on-chip modeling. In *Networks on Chip (NoCS), 2012 Sixth IEEE/ACM International Symposium on*, pages 201–210. IEEE, 2012.

[39] Y. Z. Tei, M. N. Marsono, N. Shaikh-Husin, and Y. W. Hau. Network partitioning and ga heuristic crossover for noc application mapping. In *Circuits and Systems (ISCAS), 2013 IEEE International Symposium on*, pages 1228–1231. IEEE, 2013.

[40] S. E. Van Winkle. *Dynamic Bandwidth and Laser Scaling for CPU-GPU Heterogenous Network-on-Chip Architectures.* PhD thesis, Ohio University, 2017.

[41] O. Villa, D. R. Johnson, M. Oconnor, E. Bolotin, D. Nellans, J. Luitjens, N. Sakharnykh, P. Wang, P. Micikevicius, A. Scudiero, et al. Scaling the power wall: a path to exascale. In *High Performance Computing, Networking, Storage and Analysis, SC14: International Conference for*, pages 830–841. IEEE, 2014.

[42] W. V. Winkle. Amd fusion: How it started, where it's going, and what it means. `http://www.tomshardware.com/reviews/fusion-hsa-opencl-history,3262.html`, August 2012.

[43] H. Zhao, M. Kandemir, W. Ding, and M. J. Irwin. Exploring heterogeneous noc design space. In *Proceedings of the International Conference on Computer-Aided Design*, pages 787–793. IEEE Press, 2011.

[44] E. Zitzler, M. Laumanns, and L. Thiele. Spea2: Improving the strength pareto evolutionary algorithm. *TIK-report*, 103, 2001.

[45] E. Zitzler and L. Thiele. Multiobjective evolutionary algorithms: a comparative case study and the strength pareto approach. *IEEE transactions on Evolutionary Computation*, 3(4):257–271, 1999.