Improving Communication in Chip Multiprocessors Using Emerging Technologies and Machine Learning

by

Parisa Khadem Hamedani

A thesis submitted in conformity with the requirements for the degree of Doctor of Philosophy
Graduate Department of Electrical and Computer Engineering
University of Toronto

© Copyright 2017 by Parisa Khadem Hamedani
Abstract

Improving Communication in Chip Multiprocessors Using Emerging Technologies and Machine Learning

Parisa Khadem Hamedani
Doctor of Philosophy
Graduate Department of Electrical and Computer Engineering
University of Toronto
2017

Technology scaling along with power and thermal limitations ushers the industry to the many-core system era. Thread level parallelism and complex applications demand a communication architecture that can provide high bandwidth communication and facilitate efficient usage of the computational units. Therefore, the communication architecture plays a major role in the performance of applications and the energy expended on data movements. The communication framework can be viewed at: 1) the interconnect infrastructure (physical) level and, 2) the inter-thread communication (logical) level. This thesis offers three different approaches to address the communication issues in many-core systems at the physical and the logical levels. First, we propose a novel low-power all-optical Networks-on-Chip (NoC) with a deterministic wavelength routing algorithm. This routing algorithm is based on wavelength division multiplexing that allows us to reduce the number of wavelengths and optimize micro-ring resonators in optical switches. The key advantages of our optical network are simplicity and lower power consumption. Second, to improve performance of on-chip interconnect infrastructure, we employ three-dimensional (3D) integration technology. To tackle the temperature issues in 3D NoCs, we model the mapping problem as an Integer Linear Programming (ILP) problem and propose three static ILP-based thermal-aware mapping algorithms. We then investigate the thermal constraints and their effects on temperature and performance. Third, we propose a new scheme to dynamically predict the critical path of a multi-threaded application. We formulate the cache statistics and the barrier arrival times as a machine learning problem to predict critical threads. We propagate criticality of these critical threads to the other threads using lock contention information. This dissertation demonstrates how opportunities at the physical and logical levels can be used to improve the communication framework.
Acknowledgements

First, I would like to express my sincere gratitude to my supervisor Prof. Natalie Enright Jerger for her guidance, knowledge, and valuable support. Her feedback and constructive criticism have enabled me to develop my professional and personal life. It has been a great honor for me to have an opportunity to work with Natalie.

I would like to thank my committee members, Prof. Andreas Moshovos, Prof. Paul Chow, Prof. Jason Anderson, and Prof. Gabriela Nicolescu for their insightful comments and feedback on the research and thesis.

I would like to thank all the administrative staff at ECE. Specifically, I thank Darlene Gorzo, Judith Levene, Dubravka Burin, Shawn Mitchell, Errin Barker, Kelly Chan and Jayne Leake, who facilitated all the school administrative work and were always available to support me.

I wish to thank all my colleagues in Natalie’s research group. I am very appreciative of constructive feedback that I have received from them. I would like to thank Myrto Papadopoulou who has always been supportive and generous in paying me attention through difficult times.

I dedicate this thesis to my mother and grandmother. I would like to thank them for unconditional love, support and confidence in me. My success would not have been possible without their support and all the sacrifices that they have made on my behalf.
## Contents

1 Introduction .................................................. 1
  1.1 Interconnect Infrastructure ................................. 2
     1.1.1 Optical NoC ........................................... 2
     1.1.2 3D NoC ................................................ 3
  1.2 Inter-thread Communication .................................. 3
  1.3 Thesis Contributions ......................................... 4
  1.4 Dissertation Organization .................................... 5

2 Background .................................................. 6
  2.1 Optical Interconnect ........................................ 6
     2.1.1 Waveguide ............................................. 7
     2.1.2 Micro-Ring Resonator .................................. 8
     2.1.3 Modulator ............................................... 9
     2.1.4 Detector ............................................... 9
     2.1.5 Routing Architecture ................................. 9
        2.1.5.1 Circuit Switching ................................. 10
        2.1.5.2 Wavelength Routing .............................. 10
     2.1.6 Concluding Remarks .................................. 13
  2.2 3D Integration ............................................... 14
     2.2.1 Concluding Remarks .................................. 17
  2.3 Critical Path ................................................ 17
     2.3.1 Concluding Remarks .................................. 21

3 Optical NoC ................................................ 22
  3.1 Introduction ................................................ 22
  3.2 Quartern Architecture ....................................... 23
     3.2.1 Data Network ........................................... 23
     3.2.2 Router Microarchitecture ............................. 25
        3.2.2.1 Wavelength Assignment .......................... 25
        3.2.2.2 All-Optical Switches ............................. 25
     3.2.3 QuT WDM Routing ....................................... 27
3.2.4 Example ................................................................. 28
3.2.5 Control Network ...................................................... 29
3.2.6 Methodology ........................................................ 30
3.3 Evaluation ............................................................... 32
3.3.1 Delay Evaluation .................................................... 32
3.3.2 Power Evaluation ................................................... 33
3.3.3 Energy Evaluation .................................................. 36
3.3.4 Throughput Evaluation ........................................... 38
3.3.5 Area Evaluation .................................................... 38
3.3.6 Discussion ........................................................... 39
3.4 Concluding remarks .................................................. 40

4 Exploration of Thermal Constraints for 3D Network-on-Chip 41
4.1 Introduction ............................................................ 41
4.2 Static Thermal-aware Mapping .................................... 42
4.3 OptHeraMap: Optimal thermal-aware mapping ................. 43
4.3.1 Objective function .................................................. 44
4.3.2 Constraints .......................................................... 45
4.3.3 Parameters and variables ....................................... 47
4.3.4 Linearization ....................................................... 47
4.3.5 Algorithm complexity ............................................. 48
4.4 PartiMap: Partitioning-based mapping ............................ 48
4.4.1 Objective function .................................................. 50
4.4.2 Constraints .......................................................... 50
4.4.3 Parameters and Variables ....................................... 52
4.4.4 Linearization ....................................................... 52
4.4.5 Algorithm complexity ............................................. 53
4.5 ApproxiMap: approximation mapping ............................. 55
4.5.1 Algorithm complexity ............................................. 55
4.6 Methodology ........................................................... 55
4.7 Evaluation ............................................................... 57
4.7.1 Effect of task power balancing on temperature ............. 57
4.7.2 Effect of performance and explicit thermal constraints on temperature .......................................................... 59
4.7.3 Effect of different combinations for LP objective on performance and temperature .......................................................... 61
4.7.4 Discussion ........................................................... 63
4.8 Concluding remarks .................................................. 64
List of Tables

3.1 Insertion Loss Parameters ................................................. 32
3.2 Power dissipation for optical networks with 64 Nodes .................. 35
3.3 Power dissipation for optical networks with 128 Nodes ................. 35
3.4 Characteristics of Optical Topologies .................................. 35
3.5 Area overhead normalized to QuT ....................................... 38

4.1 Technology Information (45 nm) [4] ..................................... 56
4.2 Benchmarks’ information .................................................. 56
4.3 Number of variables in the mapping algorithms .......................... 57

5.1 Sample of data inputs, features and sort based normalization ............ 70
5.2 Full System Configuration ................................................ 74
5.3 Synchronization characteristics of the used benchmarks (16-threads, simmedium). *Lock Contention* is the ratio of the number of lock acquire failures to the total number of lock acquires ....................... 75
5.4 Prediction ACP example (total number of blocks = 400, blocks on offline critical path = 100). ......................................... 77
# List of Figures

2.1 High-level structure of optical network ............................................. 6  
2.2 Optical Transmitter ................................................................. 7  
2.3 Micro-Ring Resonator ............................................................... 8  
2.4 Optical modulator and detector .................................................... 9  
2.5 4-node MWSR structure. In this figure, only 2 of four waveguides are shown [12]. 11  
2.6 16-node Spidergon structure and wavelength assignment ..................... 12  
2.7 N-node λ-router architecture [20]. In this figure $I_k$ is a source and $T_m$ is a destination. Si is silicon. .......................................................... 13  
2.8 3D bonding: face-2-face and face-2-back [139] .................................... 15  
2.9 Criticality Stacks: hardware for online criticality calculation [38] ............... 19  
3.1 16-node QuT structure and wavelength assignment ............................ 24  
3.2 16-node QuT layout .................................................................... 24  
3.3 Structure of QuT Switches ............................................................ 26  
3.4 Routing Example: N4 to N12, N2 to N8 .......................................... 29  
3.5 Control Network Structure ............................................................ 29  
3.6 Average Packet Latency for 64 nodes .............................................. 33  
3.7 Average Packet Latency for 128 nodes ............................................. 34  
3.8 Average Energy-per-bit. Traffic Patterns (excluding hotspot) use $\alpha = 0.5$ 37  
3.9 Average Throughput-per-watt normalized to QuT. Traffic Patterns (excluding hotspot) use $\alpha = 0.5$ ............................................ 39  
4.1 The CTG graph of VOPD benchmark [14] ....................................... 43  
4.2 Optimal thermal-aware mapping ..................................................... 43  
4.3 A 3-layer 3D IC with heat sink connected to the first layer ..................... 45  
4.4 PartiMap: walk-through example. In each figure, the left part represents the NoC structure and the right part represents the task partitions. ................. 49  
4.5 ApproxiMap: walk-through example .............................................. 54  
4.6 Effect of task power balancing ...................................................... 58  
4.7 Effect of performance and explicit thermal constraints .......................... 60  
4.8 Effect of different LP objective combinations ..................................... 62
Chapter 1

Introduction

Technology scaling allows increasing transistor count and chip density. However, utilizing these transistors to build a single complex microprocessor which can meet power and performance demands is challenging. Power density and temperature constraints limit increases in frequency of a microprocessor to improve its performance. Moreover, limitations in minimizing the size of a pipeline stage and poor wire scaling are other obstacles to using higher clock rate. These challenges lead the industry to integrate many simpler processors on a chip instead of a single large complex core [48, 50, 79, 118]. Intel’s Xeon Phi [2] is an example of today’s many-core with 61 cores on a single chip. Abundant on-chip resources in many-core systems provide opportunity for applications to improve their performance.

Applications are written in a parallel manner (multi-threaded) instead of a sequential format (single-threaded) to take advantage of on-chip resources and benefit from running multiple threads simultaneously. Many-core systems enable tightly coupled communication between threads. This tight coupling facilitates the exchange of dependency information between the different threads allowing the application to better utilize the available resources. To provide a correct program, threads are required to synchronize. This synchronization adds extra overhead to the inter-thread communication. Therefore, the speedup achieved from running parallel applications, compared to sequential versions, does not scale linearly with the number of threads [41]. Thread level parallelism creates the need for designing a communication architecture that can support communication of hundreds of threads and coordinate effective use of the computational units in many-core systems. Effective data communication is a key factor to exploit the plenitude of resources in many-core systems and improve performance.

Communication in chip multiprocessors can be seen from two different perspectives. The first is physical communication, which aims to provide an efficient interconnect infrastructure among the cores. The second is logical communication, which focuses on the inter-thread communication, since the performance and correctness of the application depends on it. This dissertation focuses on improving the physical and logical communication in a multi-processor by utilizing emerging technologies (silicon photonics and 3D integration) and machine learning. In what follows, I will elaborate on interconnect infrastructure and inter-thread communication.
1.1 Interconnect Infrastructure

Due to technology scaling, global interconnect has become a bottleneck for performance; with each new technology generation, global interconnect delay has not scaled at the same rate as logic delay [19, 139]. Furthermore, the rapid increase in the capability and complexity of applications demand greater interconnect bandwidth [69], which leads to an increase in the density and intricacy of on-chip interconnects. When the number of cores increases in a many-core system, using dedicated wires is not an effective communication substrate because of the global interconnect delay and the amount of required wiring [10]. Therefore, interconnects have shifted to networks-on-chip (NoCs). An important feature of the network-on-chip is its ability to share resources, which makes it a suitable substrate for parallel communication in many-core architectures.

Electrical NoCs are an effective solution for current communication requirements. As future many-core systems, with more than a hundred cores, require higher throughput and operate under increasingly tight power budgets, the continued scalability of electrical NoCs is questionable [103][116]. For a 36-node electrical mesh at 32 nm with 168-bit flits\(^1\) and 4-flit buffers per input port, the energy to transmit one flit across a 1 mm link and subsequent router (energy-per-flit-per-hop) is 197 pJ [116];\(^2\) whereas energy-per-access of a 128 MB DRAM is 280 pJ.\(^3\) Evaluation of a 256-node mesh in 22nm shows that the network channel power and buffering power of an electrical NoC exceeds the allocated network power budget by an order of magnitude [103]. Furthermore, as the network size grows, the latency in an electrical 2D-NoC quickly increases [74]. Power and latency constraints in future CMPs severely limit the scalability of electrical NoCs.

Since future many-core systems need a low power and low latency communication fabric, studying alternatives, which can overcome these limitations, is valuable. We consider two approaches to improve the power and performance of NoC as a many-core interconnect infrastructure: optical NoC and 3D NoC.

1.1.1 Optical NoC

Integrated silicon-compatible photonic technology (Chapter 2) is an alternative approach to communication, which has the potential to support large-scale networks (with hundreds of cores) with low power, low latency and high bandwidth [24, 53]. Photonics reduces network power consumption by omitting electrical buffering and switching. Although nano-photonics promises low-power, high-throughput communication, it presents a number of design challenges. Despite the fact that research in nano-photonic devices and materials is a hot area, there are some technical hurdles that could limit the deployment of optical networks. To overcome these

---

\(^1\)Data packet is split into small pieces called flits.

\(^2\)Power consumption is estimated using the method by Eisley and Peh [40]. The total energy consumed in a clock cycle is proportional to the average number of flits traversing links per clock cycle.

\(^3\)Using CACTI [101]
impediments and to fully understand nano-photonics benefits and limitations, architects need to explore this new technology in their system designs. Furthermore, they should carefully consider the optical power and the number of optical components to provide scalable optical networks.

1.1.2 3D NoC

In 3D integrated circuits (3D ICs), various layers of components are stacked vertically on each other; communication among these layers is through vertical interconnects. By combining NoC and 3D ICs, we take advantage of their different benefits [35, 106]. Some of the advantages of 3D ICs are abatement of overall interconnection length, scaling of chip dimensions, decrease of interconnect power and the possibility of mixing different technologies, e.g., logic, analog, DRAM, on one chip.

Thermal management in 3D ICs is much more significant issue than in 2D ICs [35, 106]. High temperature has several undesirable effects on nano-scale designs, such as performance degradation of transistors, increased static power consumption, increased RC delay in interconnects, and reliability issues such as reduced Mean Time To Failure and thermo-migration [107, 127]. As a result of these effects, designing and utilizing temperature control algorithms for many-core systems are essential.

Several methods have been proposed to control chip temperature [5, 32, 33, 37, 100]. Task mapping and migration are post-silicon techniques that do not require extra effort during hardware design and can manage chip temperature on the fly. Using different thermal constraints, e.g., implicit or explicit constraints, highly affects the efficiency of these techniques. Therefore, a comprehensive study on the thermal criteria and their impacts on design and usefulness of thermal-aware post-silicon techniques is crucial.

1.2 Inter-thread Communication

Although, multi-threaded applications improve the utilization of many-core resources, their parallel speed up and scalability are limited due to synchronization mechanisms [41, 42, 43]. Threads need to synchronize to ensure the consistency of shared data across all threads [114]. Synchronization mechanisms such as barriers and locks (critical sections) add some performance restrictions in parallel programs. As a result, some threads, called critical threads, have less progress and require more time to finish their execution. The other threads should wait till critical threads complete their work. The critical path of a multi-threaded application can be defined as a chain of dependent code blocks that determine which thread will be last to finish execution. A code block is defined as a sequence of instructions between two consecutive synchronization events. These blocks can belong to different threads.

Decreasing the latency of any of the blocks on the critical path directly decreases the total application runtime. Therefore, identifying the critical path is crucial to improving application
performance. Previous techniques have tried to determine the critical path offline [27, 91]. Although information collected offline can be used for high-level software optimizations, it cannot be used to guide hardware acceleration techniques during runtime. Prioritizing one thread over the others during runtime directly affects the critical path of the application rendering the critical path information obtained offline obsolete. While other works have tried to predict the critical path online [15, 22, 38, 39, 65, 66], they either ignored inter-thread dependencies or did not prioritize these dependencies based on their contribution to overall application progress.

Recognizing and analyzing the critical path along with performance optimizations to reduce the length of critical path or to accelerate its execution can reduce the running time of the application. Moreover, utilizing power saving techniques, such as Dynamic Voltage and Frequency Scaling (DVFS), on non-critical threads, can reduce the energy consumption of the application.

1.3 Thesis Contributions

My thesis makes the following primary contributions:

- **QuT: A Low-Power Optical Network-on-Chip** As a first solution to address challenges in physical communication, we propose the Quartern Topology (QuT), a novel all-optical NoC (Chapter 3). QuT targets key optical challenges to provide a low-power communication infrastructure for many-core systems. Furthermore, we propose a deterministic routing algorithm that presents contention-free network and reduces the optical components by optimizing the optical switches.

- **Exploration of Thermal Constraints for 3D Network-on-Chip.** 3D NoC can be used as a solution to tackle the physical communication’s issues. However, heat dissipation in this design is one of its important hurdles, which should be overcome. For that purpose, we propose three thermal-aware mapping algorithms, based on integer linear programming (Chapter 4). We also employ these algorithms to explore the thermal constraints and their effects on temperature and performance.

- **CPath: Dynamically Predicting the Critical Path**

As a solution to the logical communication issues, we propose a novel algorithm called *CPath* to find the critical path in multi-threaded applications on the fly (Chapter 5). We use machine-learning-based classification to choose which statistics are more indicative of a thread’s future progress. We then use these findings to predict the critical thread. We further utilize the dependencies between the critical sections to propagate a thread’s criticality from one thread to another.

Our work on the optical NoC has been published at the 8th IEEE/ACM International Symposium on Networks on Chip (NoCS) [72]; our work on thermal-aware mapping was published
at 20th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP) [71] and its extension has been published at International Journal of Adaptive, Resilient and Autonomic Systems (IJARAS) [73].

1.4 Dissertation Organization

The rest of this dissertation is organized as follows: The next chapter provides an overview of silicon photonics, 3D integration and the critical path of multi-threaded applications. Chapters 3 and 4 present the proposed solutions related to interconnect infrastructure: QuT, a new optical NoC and the exploration of thermal constraints. In Chapter 5, CPath, an online critical path predictor, is presented. In Chapter 6, conclusions and avenues of potential future research are discussed.
Chapter 2

Background

This chapter provides background for the work presented in this thesis. First, I provide an overview of photonic interconnect for Chapter 3. Then, the 3D integration is presented in Section 2.2 for Chapter 4. Finally, the chapter ends with an overview of critical path (Section 2.3) for Chapter 5.

2.1 Optical Interconnect

In this section, I provide the background on devices and characteristics of photonic interconnects. Silicon photonics can provide a low power, low latency and high bandwidth on-chip communication infrastructure (Section 1.1.1) [24, 53]. Figure 2.1 illustrates the typical structure of an optical interconnection network. Optical interconnects include components to provide light (laser), convert electrical signals to optical signals (transmitter), transmit optical signals to their destinations (medium) and finally regenerate electrical signals from optical signals (receiver).

- **Transmitter.** The transmitter translates electrical data into the optical domain. An optical transmitter includes a serializer, a driver and a modulator (Figure 2.2). The serializer...
Figure 2.2: Optical Transmitter

and driver circuitry make the electrical data compatible with the modulator. Electrical data is sent through buses employing multiple wires. The serializer accommodates the clock mismatch between the sender and optical devices. After serialization, data passes through the driver for providing the appropriate peak-to-peak voltage and adequate current for the optical modulator [12, 111]. The modulator converts the electrical signal into an optical signal by utilizing a light source and a modulation device.

- **Medium.** Optical data is routed to the destination by employing an optical medium that includes waveguides and optical switches. More details of optical routing are described in Section 2.1.5.

- **Receiver.** The receiver converts the optical signal into the electrical domain. The receiver is comprised of a photo-detector, amplifier and deserializer. The photo-detector generates an electron in the presence of a photon [12, 75]. In other words, it turns a photocurrent to an electrical current. A trans-impedance amplifier [105] produces voltage from the photodetector’s output current. Then, the deserializer converts the incoming data to a parallel data stream suitable for the destination.

- **Laser.** An off-chip laser can be used as a light source, which is integrated into a separate chip. Lasers based on III-V compound semiconductors are capable of generating several wavelength channels [129].

An optical interconnection network requires a few photonic devices that are utilized in the transmitter, medium and receiver: waveguides, switches, modulators and detectors. In the following sections, I present these optical components.

### 2.1.1 Waveguide

A basic photonic component is the waveguide, which propagates the optical signal between two points. CMOS compatible waveguides are crystalline silicon [85] and silicon nitride [142] waveguides. An optical signal loses its power when propagating through an optical waveguide. **Insertion loss** (IL) is the attenuation of signal power from a device insertion into an optical link [12]. Free carrier absorption, light scattering and substrate leakage are the sources of insertion loss in an optical waveguide [12]. Furthermore, bends and crossing, which are required in
waveguides, are the other sources of insertion loss. To compensate for the insertion loss, the light source must consume more power. Therefore, to reduce the optical power, insertion loss should be minimized. In nanophotonics, several optical carrier signals can be multiplexed onto a single waveguide to generate a parallel optical data stream. In wavelength-division multiplexing (WDM), each optical carrier has a unique wavelength to prevent interference with other signals. Utilizing WDM provides a high bandwidth optical interconnect. Recent studies show that more than 60 wavelengths [109] and up to 32 wavelengths [142] can be multiplexed on crystalline silicon and silicon nitride waveguides, respectively. In this dissertation, we use the crystalline silicon waveguide due to its smaller footprint\textsuperscript{1} and higher number of wavelengths available for multiplexing.

2.1.2 Micro-Ring Resonator

A micro-ring resonator, a circular waveguide with a small diameter ($\approx 3\mu m$ [140]), is an important optical component employed to build optical switches, modulators and detectors. Figure 2.3a shows a micro-ring resonator. When a micro-ring resonator is on-resonance with the wavelength that passes through the waveguide coupled to that micro-ring resonator, it removes the wavelength from the waveguide [12], as shown in Figure 2.3b. However, if a micro-ring resonator is off-resonance with a wavelength, a wavelength transmits on a waveguide, as shown in Figure 2.3a. The resonance mode of a micro-ring resonator with a specific wavelength has a direct relationship with its diameter and its effective refractive index [12]. The refractive index of an active micro-ring resonator [12] can be changed by applying electrical charge or temperature. Active rings are utilized in ring-based modulators, as explained in Section 2.1.3, and optical switches. Optical switches based on active micro-rings require a set-up. This set-up is used to tune active micro-rings to be on- or off-resonance with a wavelength, before transmitting optical data. Furthermore, passive micro-ring resonators [12] can be used in filters and switches for adding or dropping a wavelength into or from a waveguide. Passive micro-ring

\textsuperscript{1}Crystalline silicon and silicon nitride waveguides have $520 \times 220 \text{ nm}^2$ and $1800 \times 750 \text{ nm}^2$ cross sections, respectively.
resonators are always on-resonance with a wavelength and they do not need any set-up, i.e., electrical charge, for operations; they consume less power compared with active micro-ring resonators. Figure 2.3c depicts a optical switch, which diverts an on-resonance wavelength into the secondary waveguide. Micro-ring resonators are sensitive to the process and temperature variations in the system. Thus, they are a major source of faults in an optical interconnect. As a results, a small number of micro-ring resonators should be used in an optical path to have a low bit-error-rate.

2.1.3 Modulator

A crystalline silicon micro-ring resonator electro-optic modulator utilizes a micro-ring resonator which is tuned for a specific wavelength. Figure 2.4 depicts crystalline silicon micro-ring resonator electro-optic modulator array. An electrical data stream is used to change the refractive index of a modulator to be on- or off-resonance with a specific wavelength. Therefore, each modulator encodes its on-resonance wavelength and an optical data stream is generated. This dissertation, similar to recent work [98], employs a micro-ring resonator electro-optic modulator array due to its high-speed, low-power consumption and compact footprint.

2.1.4 Detector

Germanium (Ge), which is available in CMOS technology and can absorb light to generate electrical current at the output, is employed in photodetectors [12]. Before detecting optical data streams, wavelength demultiplexing is required. First, micro-ring resonators are utilized to isolate each wavelength channel in an optical data stream. Then, a photodetector is used to detect the data from the corresponding wavelength, as shown in Figure 2.4.

2.1.5 Routing Architecture

All of the components that are explained earlier can be employed to provide an optical network as a communication infrastructure. Different optical network architectures can be designed
based on which optical components are utilized and how these components are combined. Therefore, the literature on optical networks-on-chip may be divided into two main categories based on which micro-ring resonators are used: 1) circuit switching that employs active micro-ring resonators and, 2) wavelength routing that utilizes passive micro-ring resonators.

2.1.5.1 Circuit Switching

Several photonic NoCs [116, 24, 8, 25, 55, 58] are implemented based on active ring resonators. Active ring resonators are utilized as optical switches to transmit a set of wavelength channels in an optical path. An electrical signal is used to change the resonance mode, i.e., on- or off-resonance with a wavelength, of an active ring resonator. These optical NoCs use a circuit-switching approach and establish a path through the network before data is injected. Each data transmission requires three steps. The first step is path setup. Control packets are sent on an electrical control network to reserve the path on optical network and configure active ring resonators. A path is reserved while a control packet traverses through the electrical routers. When a control packet reaches the destination, an acknowledgment is sent to the source and all active ring resonators are tuned. The second step is sending optical data on the optical network. The final step is destroying the established path. After optical data transmission between a source and intended destination is completed, a teardown control message is sent on the electrical control network to free the reserved path for other optical data communications on the network [12, 116]. Thus, the performance of the electrical network and active ring resonators, and the use of electrical path-reservation affect the latency and power of the NoC. Two main issues in circuit switching networks are the overhead of path setup and a lack of the notion of fairness [12]. Large data messages can mitigate the cost of path reservation [24]; however, common packet sizes in many-core architectures are small (64-128 bytes to transmit a single cache line). Furthermore, long messages can easily block short messages in this circuit switching mechanism.

2.1.5.2 Wavelength Routing

Wavelength routing uses passive micro-ring resonators to route optical data based on its wavelength. A source node can send data to the destination of interest by modulating its data on an appropriate set of wavelengths. Then, that optical data is routed by the waveguide and micro-ring resonators of the optical network. Wavelength routing does not require path reservation similar to circuit switching. Most wavelength routing networks put some limitation on the number of concurrent data streams that can pass through shared links or that a destination can accept, to reduce the total number of micro-ring resonators and wavelength channels. In that case, a separate arbitration network is used to resolve access conflicts to the shared link or node. For instance, when a destination can only accept the optical data from one source at a time, a source needs to send a message on the arbitration network to inform the intended destination and receives an acknowledgement for sending its optical data. The arbitration network can be
electrical or optical. Although, an electrical arbitration network reduces the number of required wavelength channels, its latency limits the performance of a NoC. The transmission latency of the wavelength routing network is smaller than circuit switching networks. In wavelength routing, an arbitration network can be implemented with an optical network, which can provide a low-latency network. That is because passive micro-ring resonators do not require electrical signals for operation. Moreover, the wavelength routing network does not require a tear-down step, which is needed in an optical circuit switching network. Therefore, other sources can use the shared resources faster. Furthermore, arbitration is only required at the destination and unlike circuit switching there is no contention in an optical arbitration network. In circuit switching path setup, if contention is detected in the middle of the path reservation phase, all of the reserved resources should be released before other sources can use them. Furthermore, passive micro-rings consume less power than active ring resonators. This dissertation focuses on a wavelength routing optical NoC.

Several optical NoC architectures leveraging wavelength routing have been proposed differ in topology, complexity, power and latency. The simplest topology is an optical non-blocking point-to-point connection (Single-Writer-Single-Reader) [77, 78, 82]. This topology provides a fully connected crossbar, which suffers from low bandwidth and poor scalability. Other works try to alleviate these issues by proposing different optical crossbars (Single-Writer-Multiple-Reader or Multiple-Writer-Single-Reader) with various arbitration techniques. The Firefly architecture [104] is an example of a Single-Writer-Multiple-Reader (SWMR) interconnect. In a SWMR network, a source node needs to broadcast a head flit to notify the target destination node. All nodes listen to the broadcast links and tune themselves in/out based on the broadcast message. As a result of that, a SWMR network has high insertion loss which increases with the number of nodes on the broadcast links.

Corona [131] is based on a Multiple-Writer-Single-Reader (MWSR) crossbar. Figure 2.5 shows a 4-node MWSR crossbar. In MWSR, each node can receive data from a dedicated waveguide, on which the other nodes can send optical data. In other words, a node can send data on all waveguides but it can only receive data from one of them. For instance, node 3 (to access point 3) can receive data from the top waveguide, while the lower waveguide is assigned
to node 2 (to access point 2) in Figure 2.5. A node can utilize all wavelengths to modulate data on a waveguide. In that case, the number of wavelengths does not depend on the number of nodes in the system. However, the number of waveguides and micro-rings increase with the number of nodes. We choose Corona as a candidate of optical crossbars to compare against.

R-3PO [98] is a multi-layer optical interconnect based on a MWSR crossbar, similar to Corona. R-3PO leveraged 3D photonic stacking to provide a dynamically reconfigurable network to improve performance and re-allocate bandwidth of a faulty channel. It also used token slot as an arbitration mechanism. FlexiShare [18] has been proposed to improve the link utilization by combining the advantages of SWMR (Firefly) and MWSR (Corona) crossbars. FlexiShare can be considered as a Multiple-Writer-Multiple-Reader (MWMR) crossbar. A MWMR crossbar introduces large insertion loss compared with SWMR and MWSR. This is due to a high degree of link sharing and the fact that micro-ring resonators are required at both senders and receivers. Both SWMR and MWSR crossbars require a large number of optical resources such as micro-rings. Moreover, they have high insertion loss leading to the high power consumption which is proportional to the network size.

Furthermore, several wavelength routing architectures (other than crossbar-based architectures) have been proposed to incorporate resource sharing and improve performance of optical NoCs. These optical NoCs use different optical switch structures suited to their topologies, such as ring-based [80, 84, 134] and mesh-based [54, 136, 141] topologies, multi-stage design [9, 31, 67, 87, 88, 97] and λ-router [20].

Subsequent to our work presented in this dissertation, Amon [136] has been proposed as a mesh-based topology with six different switch designs to provide tile-based infrastructure. It employed a MWSR crossbar for its arbitration network. LumiNOC [87, 88] is a multi-stage design whose primary stage is optical. Since it uses electrical routers in its intermediate stage, at high injection rate, its performance is limited by electrical routers. Its arbitration policy is based on optical collision detection and dynamic channel scheduling techniques, which efficiently reuses optical resources for power efficiency.

Figure 2.6: 16-node Spidergon structure and wavelength assignment
Spidergon [80] assigns dedicated wavelengths to each destination, utilized by sources for optical communication. Figure 2.6 shows a 16-node optical Spidergon with dedicated wavelengths to each node. Spidergon is a ring-based topology with extra links, called cross links. The cross links are used to decrease the network diameter and the number of required wavelengths. This optical network requires $N/2$ distinct wavelength sets, where $N$ is the total number of nodes in the system. A source modulates its optical data on a wavelength set assigned to the corresponding destination. Then, the source chooses one of the connected links, i.e., cross or ring link, to send the modulated optical data. Optical switches transmit data to the destination. Although, Spidergon is based on wavelength routing and uses passive micro-ring resonators, its proposed arbitration network is electrical, which limits its performance. We compare our proposed optical NoC with Spidergon, since both optical NoCs are ring-based topologies.

$\lambda$-router [20] is a contention-free optical network. Figure 2.7 shows an $N$-core $\lambda$-router structure. A source uses a truth table to modulate its data on a specific wavelength for sending to a destination. For example, source $I_N$ should modulate data on wavelength $\lambda_N$ for sending optical data to destination $T_1$. In $\lambda$-router, all of the sources can send data to a destination without collisions. Therefore, it does not require any arbitration mechanism. However, its number of micro-ring resonators increases quadratically with the number of nodes in the network. We consider $\lambda$-router as another candidate of non-crossbar optical NoC and compare our proposed optical NoC with it.

2.1.6 Concluding Remarks

To enable the adoption of optical NoCs and allow them to scale to large systems, they must be designed to consume less power and energy. Therefore, optical NoCs must use a small number of wavelengths, avoid excessive insertion loss and reduce the number of micro-ring resonators. In this thesis, we propose the Quartern Topology (QuT), a novel low-power all-optical NoC. QuT is a wavelength routing optical NoC. We also propose a deterministic routing algorithm based on
WDM that allows us to reduce the number of wavelengths and micro-ring resonators in optical routers. The key advantages of QuT network are simplicity and low power consumption. We compare QuT against three alternative all-optical NoCs: optical Spidergon [80], λ-router [20] and Corona [131] under different synthetic traffic patterns. QuT demonstrates good scalability with significantly lower power and competitive latency.

2.2 3D Integration

In three-dimensional integrated circuits (3D IC), multiple device layers are stacked vertically by employing bonding technology [139]. Each layer is fabricated separately. This emerging technology can offer several advantages: reduction in interconnect length, increase in memory bandwidth and heterogeneous integration [139]. 3D integration can alleviate the interconnect scaling issue by reducing the length of global interconnects. The reduction in the global wire length can be equal to the square root of the number of layers [68]. As a result, latency and power are improved. Furthermore, 3D integration can mitigate the effect of I/O pin limitations on off-chip memory bandwidth. Stacking memory on the core layers can provide enough data for a large multicore system and improve its performance [17, 70]. Furthermore, different technologies, such as analog, photonics and nonvolatile memories, e.g., magnetic RAM, can be integrated on a chip using 3D integration.

3D IC uses two different bonding technologies: Face-2-Face and Face-2-Back, as shown in Figure 2.8. The face of a die includes the metal layers, while the back of a die consists of silicon substrate or active layer. The face-to-face method bonds faces, i.e., metal layers, of two dies. In this scheme, microbumps are used as the inter-tier connections for two stacked dies. However, for stacking more than two dies, the face-to-face method has to use Through Silicon Vias (TSVs). TSVs are vertical interconnects that go through stacked layers, as shown in Figure 2.8. The face-to-back method bounds the face of a die to the back of the other. Inter-tier connections are implemented with TSVs. The face-to-back bonding process provides a more symmetrical structure compared with the face-to-face technique. In this thesis, we consider the face-to-back technology.

TSVs are the most important component in 3D integration. These vias are etched through dies and filled with metal, e.g., copper [95]. The length of a TSV depends on the thickness of a die. A thinning process is used to reduce the thickness of a die. The thickness of a die in the face-to-back method is between 10µm and 20µm. Moreover, the pitch of TSVs is between 10µm × 10µm and 1µm × 1µm [95]. The pitch of a TSV affects the transistor layouts and the inter-die bandwidth. Since TSVs go through the silicon layers, transistors cannot be placed in the area assigned to TSVs. The large pitch of TSVs also reduces the inter-die bandwidth per area. The latency of a 10µm TSV is 8 ps which is similar to the short interconnect in 2D integration [95].

Although 3D integration has many advantages, it presents several design challenges such
as test, power delivery and heat dissipation. 3D ICs require extra processing steps, e.g., die thinning, alignment and stacking, compared to 2D IC. These processing steps introduce some new defects. Delamination might happen if any foreign particles become trapped between dies during the bonding process [86]. Dies must be checked for any cracks after the stacking process [86]. TSVs need to be precisely aligned to provide correct connectivity and minimize their resistance [86]. Reliable power delivery in 3D ICs is more significant than 2D IC due to: I) In 3D IC, the small footprint reduces the ratio of supply pins to the supply current [63]. Therefore, the package parasitics are increased. II) Furthermore, TSVs add extra resistance in the supply path which impacts the noise characteristics in the supply network [63].

The thermal issue is a bottleneck for both 2D and 3D designs. This issue is worsened in 3D ICs due to high power density and large thermal resistance. Small chip footprint and high transistor density increase the power density in 3D chips. Moreover, multiple silicon layers of stacked dies in the path of thermal dissipation increase the 3D chip’s thermal resistance, since silicon has a large thermal resistance. Heat dissipation is slow in the presence of a large thermal resistance. High temperature has a negative effect on transistor performance and leakage power. Thermal voltage$^2$ increases with respect to temperature. High temperature also increases the wire latency due to an increase in wire resistance. The wire resistance depends

\[ v_T = kT/q, \text{ where } T \text{ is the temperature, } q \text{ is the magnitude of the electrical charge and } k \text{ is the Boltzmann's constant} \]
on collisions between free electrons and captive electrons. When the temperature rises, the more collisions occur. As a result, wire resistance increases. Another undesirable effect of high temperature is a reduction in reliability and Mean Time To Failure (MTTF) [107, 127]. MTTF might reduce when electromigration increases with rising temperature. The electrical current in metal transports material. Thus, wires are eventually thinned. Therefore, it is essential to develop thermal-aware mechanisms at all levels of the design process.

Several methods have been proposed to control chip temperature in 3D ICs. We can classify these methods into three categories. The first group of methods is devised and implemented before the chip is built. Examples are thermal-aware placement [32] and thermal-aware routing [143]. The second group contains methods, such as DVFS [100], that are devised before the chip is built. Whether or not they are used (and how) depends on the runtime conditions of the system. Finally, the third group represents post-silicon techniques. These techniques, such as task mapping and migration [37, 57] algorithms, are applied after building the chip. They manage chip temperature while the system is running.

Employing NoC in 3D integration can simplify the design and placement of TSVs. In addition, 3D integration reduces the network diameter and improves power and latency of NoC. Both NoCs and 3D integration tackle the interconnect issues in different ways. Therefore, we can take advantage of both techniques by combining them. In this thesis, we consider a network-on-chip as an interconnect infrastructure for multi-core system and focus on a post-silicon technique: thermal-aware mapping. Thermal-aware mapping algorithm is orthogonal to the other thermal-aware methods. Since application mapping is one of the most important problems in NoC due to its impact on power and performance, several works have been dedicated to it. In an application mapping, different concurrent tasks are assigned to the cores based on an optimization constraint [96]. The optimization constraint can be minimizing the network communication rate [99] or energy [52, 60, 112] or minimizing the temperature [5, 23, 28, 117, 122, 145].

Zhou et al. [145] proposed a multi-objective mapping algorithm based on a genetic algorithm for 2D NoC. They tried to find an optimal mapping, while minimizing the average hop count and achieving the thermal balance. For 3D NoC, Addo-Quaye [5] proposed three genetic-based mapping algorithms targeting temperature, communication and hybrid, which both temperature and communication are considered. At each iteration of the genetic algorithm, it generates random pool of solutions and evaluates them using fitness score of objective function. In this case, the fitness score of thermal-aware or communication-aware mapping algorithm is inversely proportional to the peak temperature or communication volume, respectively. The hybrid algorithm’s fitness score is sum of the two previous scores. Addo-Quaye showed that the hybrid approach provides better results in terms of temperature and performance. Sun et al. [122] proposed a heuristic thermal-aware algorithm for 3D MPSoCs including task mapping, scheduling

---

3 As stated earlier, heat dissipation is a critical problem even for 2D ICs, for which previous literatures tried to provide solutions [21, 37] In this thesis, we focus on thermal management techniques in 3D ICs.
and voltage scaling. Their thermal-aware task mapping tries to balance spatial power density
by estimating the task’s power and calculating the task’s thermal impact on core temperature.
They used the temperature feedback to identify the hotspot. Then, they utilized voltage scaling
to reduce the temperature of the hotspot by considering the maximum permitted slowdown for
tasks. Although few thermal-aware mapping algorithms have been proposed, there has been
a lack of comprehensive studies on how and to what extent different thermal constraints in
mapping algorithms affect performance and temperature. Due to this lack of study, we propose
thermal-aware mapping based on integer linear programming (ILP) to explore the effect of
different criteria in the thermal-aware mapping algorithm on the chip temperature. ILP-based
algorithms provide optimal or approximate solutions.

Subsequent to our work, several new thermal-aware mapping algorithms have been pro-
posed [23, 28, 117]. Cheng et al. [28] proposed a thermal-aware mapping considering intercon-
nect energy consumption. The mapping started with an initial thermal-balanced solution based
on tasks’ power. It then employs a greedy algorithm to minimize the interconnect energy con-
sumption. Finally, it uses a simulated annealing to further optimize the result. Singh et al. [117]
proposed a mapping algorithm for 3D videos on 3D chips. First, they analyzed the frames of
3D videos to identify compute- and communication-intensive tasks. Later on they used this
information to map tasks on cores considering temperature, energy and throughput. Cao et
al. [23] proposed an online heuristic task mapping considering temperature and communication
overheads. They first estimated the core’s temperature based on its current temperature and
future scheduling of a specific task. They used the profiling data from previous steps to map
tasks on cores. They took into consideration that fewer tasks should be run on hot cores.
They improved temperature by employing DVFS considering timing constraints. After using
the maximum permitted DVFS, tasks were migrated from hot cores to cool cores.

2.2.1 Concluding Remarks
In this thesis, we propose three thermal-aware mapping algorithms for 3D NoC, Chapter 4.
Unlike the previous works, we model the application mapping with integer linear programing
considering thermal, power and performance constraints. To pave the way for future research in
this area, we then explore the effects of these constraints on the performance and temperature
of multi-core systems.

2.3 Critical Path
The critical path of an application can be defined as the most prolonged sequence of events from
the beginning to the end of the application [59]. Several works have been proposed to identify
the critical path of sequential applications at the instruction-level granularity. Some instructions
have more of an effect on performance than the others. By running them faster, application
performance will be improved. Moreover, since the number of these critical instructions is
small [92], performance optimization methods can be implemented with low overhead. To detect the critical instructions, Tune et al. [128] and Prieto et al. [110] employed the position of the instruction in the ReOrder Buffer (ROB). Since, the instruction at the head of the ROB can block other ready-to-commit instructions, criticality of an instruction is increased when it is closer to the head. Fields et al. [45, 44] proposed a dependence-graph. The dependence-graph model can capture data and resource dependences between instructions. A node in the dependence-graph model represents an event: dispatch, execute and commit. An edge represents a dependence between instructions: micro-architectural dependence, e.g., control dependence or commit follows execution, and program order dependence, e.g., in-order commit. Moreover, they designed online token-passing hardware for the dependence-graph to predict the critical instructions. These schemes, targeting sequential application, cannot be directly applied to the multi-threaded applications due to dynamic dependences between threads.

Prior proposals have been developed to detect and predict the critical path of parallel applications. Some of the previous works aim to predict the critical threads by considering the thread’s progress [15, 22]. These techniques estimate the threads’ progress without directly utilizing the synchronization events and inter-thread dependencies, which are important characteristics of multi-threaded programs. Cai et al. [22] proposed a technique, called meeting point thread characterization, to dynamically identify the critical thread and the amount of slack time in non-critical threads. They used the workload imbalance to find the critical thread. They measured the workload imbalance at intermediate points, called meeting points. Meeting points were added at the back edge of the parallel loops. They used a counter per thread to accumulate the number of iterations at the meeting point and broadcast the counter values at specific time intervals. Furthermore, they employed the thread criticality information for two optimizations: thread delaying and thread balancing. Thread delaying employs DVFS to reduce energy consumptions of non-critical threads, while thread balancing gives the critical thread higher priority in the usage of issue slots to accelerate critical thread. Bhattacharjee et al. [15] determined the critical threads by using memory hierarchy statistics. They showed that the predictor based on the cache misses at different levels of hierarchy has the best accuracy compared with other techniques such as instruction counts. They applied their predictor to two optimizations: task stealing from critical threads to improve load balancing and frequency scaling of non-critical threads to reduce energy consumption. Although, these techniques can be implemented with low hardware overhead, they are not guaranteed to capture the application phases and effects of threads on each other to predict the accurate critical threads.

Several works have considered the thread communication information via synchronization events to predict critical threads [38, 65, 66, 83, 89, 94]. Although, these techniques utilized the multi-threaded program behavior, the metrics that they used for measurement of criticality affects their prediction accuracy and their hardware complexity and overhead. The thrifty barrier [89] predicted barrier stall time to reduce the energy of non-critical threads by forcing the core into a low-power sleep mode. Liu et al. [94] predicted barrier idle time and scaled
the frequency of the processors to reduce these idle times and power consumption without affecting performance. Lakshminarayana et al. [83] predict the critical thread, which has the longest remaining execution time, at each barrier. They assumed that all threads have similar execution lengths, which is not a correct assumption for all of the applications. Therefore, they utilized the number of threads’ executed instructions to estimate the remaining execution time. Then, they executed the thread lagging behind on a fast core.

In this thesis, we choose Criticality Stacks [38] and Bottleneck Identification and Scheduling [65], which proposed different methods to predict the critical path of multi-threaded application, to compare against our critical path predictor in Chapter 5. DuBois et al. [38] proposed a metric to measure thread criticality, Criticality Stacks. Their metric utilized the amount of time that a thread is executing useful work and the number of threads concurrently waiting. Useful work is indicated by synchronization behavior, e.g., critical sections or barriers. In other words, if a thread is waiting on a synchronization event, it is not performing any useful work. This metric measures the amount of parallel work that is done between two time intervals. A thread’s criticality is updated at each time interval. A thread is more critical if more threads are waiting concurrently on it.

To evaluate the threads’ criticality, they employed a 64-bit counter per thread and an active bit that shows whether a thread is waiting or not, as shown in Figure 2.9. Software support has been used for marking the synchronization events. The active bit (A in Figure 2.9) is set or reset based on calls from software at each synchronization event, e.g., if a thread waiting to acquire a lock, its active bit is reset. Moreover, a counter and a timer are used to count the number of active threads and keep track of time in the interval. They used 10 ms for each time interval. At the beginning of a new time interval, the timer is reset. When a signal from software call is received, the value of the timer divided by the active thread counter is added.
to each active thread’s criticality counter. Then, the active bits are updated. At the end of each time interval, counters are checked. The thread with the largest counter is predicted to be critical. Criticality Stacks is a coarse-grained critical path predictor. It uses DVFS to increase the frequency of critical threads based on the criticality metric. In other words, it accelerates threads that will reach to the barriers last or the critical section which has high contention.

Bottleneck Identification and Scheduling (BIS) [65] has been proposed to identify the critical bottlenecks in parallel programs. BIS utilizes software instructions to mark and identify bottlenecks, e.g., barriers and locks. It measures the total waiting time of threads for each bottleneck. If a thread waiting cycles on a bottleneck is larger than a predefined threshold, the bottleneck is accelerated by running on the large core. For that purpose, it used an associative, centralized cache called the Bottleneck Table (BT). Each entry in the BT keeps track of a bottleneck, the number of waiters on that bottleneck and its thread waiting cycles (TWC). Every cycle, TWC is incremented by the number of waiters. The number of waiters is the number of threads explicitly or implicitly waiting for the corresponding bottleneck. A Nested bottleneck can make a thread implicitly wait for a specific bottleneck. For instance, if thread T1 is waiting for a lock L1 which is followed by barrier B1 and thread T2 is waiting for barrier B1, T2 is implicitly waiting for lock L1. Therefore, the waiting cycles of T2 should be added to L1 not B1. Thus, BIS adds a new field to each BT entry, executer_vec, which records threads executing the corresponding bottleneck. Moreover, it uses a table, Current Bottleneck Table (CBT), indexed with thread ID to record a bottleneck for which thread is waiting. BIS updates TWC in BT based on simple algorithm. This algorithm tries to find a root of a dependence chain between bottlenecks and increment its TWC. BIS is fine-grain critical path prediction and it finds bottlenecks limiting performance by considering thread waiting time. Utility Based Acceleration (UBA) [66] utilized the BIS [65] and lagging thread prediction [83] techniques to predict the critical code segments. They defined a metric to estimate the expected speed-up from running a segment on a larger core.

Use cases of critical thread prediction. If the critical thread can be determined dynamically, the performance or energy consumption of a multi-threaded application will be improved by employing some optimization techniques.

- **Energy Saving and DVFS.** As mentioned earlier, some of the prior works use DVFS to scale the frequency of non-critical threads for power saving [15, 22, 89, 94]. On the other hand, CS [38] employs DVFS to increase the frequency of the critical thread and improve its execution time.

- **Priority Based Mechanism.** Cai et al. [22] employ the criticality-aware issue queue allocation. In their technique, the critical thread has higher priority in the utilization of the issue slots. Jin et al. [64] reduce the communication latency of critical threads in the network-on-chip by using bypass flow control and priority-based arbitration.

- **Scheduling.** Ebrahimi et al. [39] propose a memory scheduling algorithm for multi-
threaded applications with locks and barriers, to manage the inter-thread memory interference. First, they predict the set of critical threads using the technique similar to a meeting point [22]. Then, they design the memory scheduler that prioritizes the critical threads requests and shuffles the priorities of non-critical threads to reduce DRAM interference. Suleman et al. [120] utilize a technique that schedules the running of the critical sections on a high-performance core to accelerate their execution. The proposed priority-based memory scheduler [51] used the load criticality information in each thread. Load instructions have been predicted to be critical if they block a core’s reorder buffer (ROB) in a CMP.

- **Reducing Cache Misses.** Demetriades et al. [36] propose a synchronization point based coherence prediction. Their predictor extracts the communication patterns and predicts the target destination of coherence requests at the synchronization points to improve coherence communication and miss latency. Trancoso et al. [124] use data prefetching and forwarding to reduce the cache misses inside the critical sections. Therefore, they reduce the critical section running time to improve the performance.

- **Load Balancing Technique.** Bhattacharjee et al. [15] employ their thread criticality predictor, based on cache misses, to guide Intel Threading Building Blocks (TBB) [3] task stealing technique for performance improvement. TBB uses a dynamic scheduler to randomly choose threads and steal tasks from them to keep all threads busy. Therefore, the critical threads, which are identified by the thread criticality predictor, are ideal candidates for the task stealing approach.

### 2.3.1 Concluding Remarks

We propose *CPath* to predict the critical path of a multi-threaded application on the fly, Chapter 5. Unlike previous techniques, *CPath* utilized different metrics for criticality prediction at different synchronization events. It employs barriers to capture the coarse-grain application’s phases. Then, it utilizes a machine learning algorithm at barriers to predict the critical threads. To improve the criticality prediction, it propagates the criticality information to the code blocks in different threads based on the lock’s contention. *CPath* does not predict the critical path based on the amount of waiting time at synchronization events. It tries to follow the dependency between synchronization events for a more accurate estimation of a code block’s effect on the overall progress of the application. As mentioned earlier, we compare *CPath* with a coarse-grained, Criticality Stacks [38], and a fine-grained, Bottleneck Identification and Scheduling [65], criticality predictors. Furthermore, we compare these techniques with an oracle to evaluate the accuracy of predictions. Moreover, to exploit the criticality information, we utilize DVFS on non-critical threads to save power with negligible performance degradation.
Chapter 3

Optical NoC

In this chapter,\(^1\) we examine the first of two solutions that focuses on the interconnect infrastructure, as explained in Chapter 1. First, we elaborate on optical design challenges. Then, in Section 3.2, we propose a new optical NoC. Finally, Section 3.3 presents the latency, power, energy, throughput and area evaluations of the proposed architecture.

3.1 Introduction

Although nano-photonics promises low-power, high-throughput communication, it presents a number of design challenges. Despite the fact that research in nano-photonic devices and materials is a hot area, there are some technical hurdles that could limit the deployment of optical networks. To overcome these impediments, scalable optical networks should carefully consider power, the number of microrings and the number of wavelengths.

- **Insertion Loss:** To fully realize the potential of nano-photonics, the network must be carefully designed with low-power in mind. Laser power is one of the main factors in the power consumption of an optical NoC. The laser power is calculated based on the maximum insertion loss (IL) in an optical system. Therefore, to reduce optical power, we must decrease IL.

- **Number of micro-rings:** In terms of reliability, micro-rings are a major source of faults in an optical system, since they are sensitive to process and temperature variation. An optical NoC should have simple switches to reduce the number of micro-rings in the path of the optical streams.

- **Number of Wavelengths:** The number of wavelengths used to modulate the data in an optical system, should be reduced to have a low-power and practical optical NoC.

\(^1\)This chapter was published at the 8th IEEE/ACM International Symposium on Networks on Chip (NoCS) ©2014 IEEE/ACM [72]
– Wavelength-division multiplexing (WDM) can substantially improve the optical network bandwidth. WDM allows the transmission of different data streams through a single optical waveguide [24]. Although, existing work relies heavily on Dense WDM, the number of wavelengths that can be transmitted in a single waveguide is limited. Current research estimates that less than 100 wavelengths can be supported per waveguide [109].

– Laser power is proportional to the number of wavelengths in an optical system. Supporting a large number of wavelengths increases the required laser power.

To design a low-power optical architecture, we must address these challenges. Focusing on one or two of them is not sufficient to have a low-power optical NoC. In this chapter, we propose the Quartern Topology (QuT), a novel low-power all-optical NoC targeting key optical challenges. We use passive micro-ring resonators [81] to route optical streams based on their wavelength. QuT is a ring-like topology with strategically placed extra links to reduce the diameter and number of wavelengths required. QuT takes steps towards addressing the optical challenges including IL, number of wavelengths and micro-rings. Therefore, QuT consumes less power compared with state-of-the-art proposals. Our proposed deterministic routing algorithm avoids collisions between streams that need to transmit on the same wavelength. It provides contention-free network traversal, while reducing the number of wavelengths required by optimizing the optical switches.

3.2 Quartern Architecture

In this section, we present our low-power all-optical NoC that features a small number of wavelengths and simple switches that leads to low resource requirements. These features are essential to realizing optical networks in practice and scaling them to large systems. QuT is a contention-free topology; data streams intended for different destinations do not block each other. However, a given destination can only accept traffic from a single source at a time; as a result, QuT requires a control network to serialize access by multiple sources to a particular destination. Both the control and data networks are optical. We first explain the data network and optical switches. Then, we explain the WDM routing and clarify the details of QuT through an example. Finally, we explain the control network.

3.2.1 Data Network

QuT uses passive micro-ring resonators, which route optical streams based on their wavelengths.\textsuperscript{2} Figure 3.1 shows a 16-node QuT network and Figure 3.2 shows a layout of QuT, which reduces the waveguide crossings. Extending QuT structure to larger numbers of nodes is straightforward; the connectivity for an arbitrary N nodes is given below. QuT includes two

\textsuperscript{2}It is a wavelength routing network as explained in Section 2.1.5.2.
differently types of switches and three different connections between nodes which enable a WDM routing approach to communicate between nodes. The link for sending data is chosen based on the distance between the current node and the destination. Data traverses Ring links when the distance between the current node and the destination is less than $N/4$. Otherwise, Cross or Bypass links are used for sending data streams. Utilizing these different links reduces the
network diameter to $N/4 + 1$. The network diameter is the longest of all shortest paths in the network.

QuT is based on the following connectivity formulation:

- **Ring links (bidirectional)**
  
  $x_i$ is connected to
  
  \[
  \begin{cases}
  (x_i + 1) \mod N \\
  (x_i - 1) \mod N
  \end{cases}
  \]

- **Cross links (bidirectional)**
  
  if $i$ is even, $x_i$ is connected to
  
  \[
  \begin{cases}
  (x_i + N/4) \mod N \\
  (x_i + 3N/4) \mod N
  \end{cases}
  \]

- **Bypass links (unidirectional)**
  
  if $i$ is odd, $x_i$ is connected to
  
  \[
  \begin{cases}
  (x_i + 1) \mod N \\
  (x_i - 1) \mod N
  \end{cases}
  \]

Where switch $x_i$ is connected to node $i$ and $N$ is the number of nodes in QuT.

Although bypass links appear to have the same connectivity as ring links, there are two important differences. First, they are unidirectional: they only emanate from odd nodes. Second, their connection within the switch is different from ring links. We use bypass links, instead of additional cross links in odd switches, to prevent the data from being absorbed by the wrong node as we will see in Section 3.2.2.2.

### 3.2.2 Router Microarchitecture

In this section, we describe the assignment of wavelengths to nodes and the design of the optical switches.

#### 3.2.2.1 Wavelength Assignment

Each node in the QuT data network has a specific, dedicated but not unique wavelength onto which other nodes will modulate their data to communicate with this node. We have designed QuT, its deterministic wavelength routing and its optical switches to use a small number of wavelengths. In an $N$-node QuT, we use $N/4$ distinct wavelength sets to modulate all data in the network. A wavelength set of $\lambda(i \mod N/4)$ is assigned to node $i$ (Figure 3.1). Each wavelength set can include one or more wavelengths based on the required network bandwidth.

#### 3.2.2.2 All-Optical Switches

QuT requires two different switches, one for even (Figure 3.3a) and one for odd (Figure 3.3b) nodes. A source chooses one of the four injection channels\(^3\) in its optical switch, based on the

\(^3\)The injection channels and the ejection channel are connected to the nodes modulators and detector, respectively.
Figure 3.3: Structure of QuT Switches
destination’s index. Choosing among four injection channels helps reduce the total number of wavelengths required by the network.

Micro-ring resonators in the even and the odd switches are classified in four groups: (a) Add micro-ring Resonators (A\(\mu\)R) that multiplex optical streams from injection channels or cross links to ring links (b) Bypass micro-ring Resonators (B\(\mu\)R) in the even switches that switch optical streams from bypass links to cross links (c) Cross micro-ring Resonators (C\(\mu\)R) in odd switches which turn optical streams from ring links to bypass links (d) Drop micro-ring Resonators (D\(\mu\)R) that send the data, modulated on the wavelength of the corresponding destination node, from ring links into the switch’s detector.

A\(\mu\)R, B\(\mu\)R, C\(\mu\)R and D\(\mu\)R are arrays of micro-rings, where each micro-ring is sensitive to a specific wavelength. By assuming that \(\lambda_{(i \mod N/4)}\) is devoted to the \(i\)th node, D\(\mu\)R in the \(i\)th optical switch is only sensitive to \(\lambda_{(i \mod N/4)^4}\). A\(\mu\)R is used to prepare suitable turns for all optical data streams. Hence, A\(\mu\)R is sensitive to all wavelengths in QuT. B\(\mu\)R is used to send data on one of the cross links connected to the even node’s switch. Therefore, B\(\mu\)R is sensitive to all wavelengths in QuT. Even nodes benefit from C\(\mu\)Rs in odd switches when they want to send data to the destination that is a distance of \(N/2\) from them. In QuT, nodes with distance \(N/4\) or \(N/2\) from a source, have the same wavelength because we use \(N/4\) different wavelengths. As a result, if a source wants to send data to the destination that is a distance of \(N/2\), it should use a path that will bypass the node that is \(N/4\) from it to prevent the data stream from being absorbed by the wrong node. Employing C\(\mu\)Rs in odd switches prepares suitable paths for that purpose. Hence, the left C\(\mu\)R (right C\(\mu\)R) in the \(i\)th optical switch is sensitive to the wavelength assigned to the even switch to the left (right) of the \(i\)th optical switch. The left C\(\mu\)R is sensitive to \(\lambda_{(i+1 \mod N/4)}\) and the right C\(\mu\)R is sensitive to \(\lambda_{(i-1 \mod N/4)}\).

3.2.3 QuT WDM Routing

We develop a novel deterministic optical routing algorithm tailored to the QuT architecture and based on wavelength routing. The wavelength assignment coupled with the routing algorithm allows us to realize our contention-free architecture. No two distinct source-destination pairs using the same wavelength will collide in the network; if they use the same wavelength, they will take different routes. Also, using dedicated deterministic optical routing helps optimize the optical switches and reduces the number of micro-rings compared with non-deterministic optical routing algorithm.

First, the source node chooses an injection channel (designated I\#)\(^5\) based on the destination. Then, data is modulated onto the destination wavelengths. The optical data stream is transmitted through optical switches and links until it is ejected at the destination. There are several scenarios depending on the source and destination, listed as follows:

\(^4\)Notice that to increase the optical network bandwidth we assign more than one wavelength, e.g., 8 wavelengths, to each node. Each optical data stream is modulated on several wavelengths assigned to the corresponding destination. Therefore, D\(\mu\)R is an array of micro-rings.

\(^5\)I1-I4 in top of Figures 3.3a and 3.3b
• If the source is even and the distance between the source and the destination is:

  – less than \(N/4\), \(I_2\) (\(I_3\)) is chosen if the destination is to the left (right) of the source. Data is sent on the ring link to the left (right) of the current switch.
  
  – equal to \(N/4\), \(I_1\) (\(I_4\)) is chosen if the destination is to the left (right) of the source. Data is sent over the cross link to the left (right) of the current switch. When the distance between two nodes is divisible by \(N/4\), these nodes have the same dedicated wavelength.
  
  – equal to \(N/2\), the source and the destination have the same dedicated wavelength. Therefore, \(I_2\) is chosen to send data over the ring link to the left of the current switch. The odd switch to the left of the current switch will send the data stream over its bypass link, to bypass the node that is a distance of \(N/4\) from the source and prevent the data from being absorbed by wrong node.
  
  – greater than \(N/4\) and not equal to \(N/2\), \(I_1\) (\(I_4\)) is chosen if the destination is to the left (right) of the source. The cross link to the left (right) of the current switch is used.

• If the source is odd and the distance between the source and the destination is:

  – less than or equal to \(N/4\), \(I_2\) (\(I_3\)) is chosen if the destination is to the left (right) of the source. Data is sent on the ring link to the left (right) of the current switch.
  
  – greater than \(N/4\) and not equal to \(N/2\), \(I_1\) (\(I_4\)) is chosen if the destination is to the left (right) of the source. Data is sent over the bypass link to the right (left) of the current switch.

3.2.4 Example

Figure 3.4 gives an example of routing in QuT. A common wavelength (\(\lambda_0\)) is used for both data streams from \(N4\) to \(N12\) and \(N2\) to \(N8\); data transmission through different paths prevents optical collision. If \(N2\) wants to send data to \(N8\), it chooses injection channel \(I_1\). Therefore, data is transferred over a cross link connected to \(N6\). Then, the data stream is turned onto the ring link by \(A_{\mu}R\) between the cross and ring links in \(N6\). Data is transferred through the ring link without changing direction in \(N7\). Finally, in \(N8\), data turns into the ejection channel through \(D_{\mu}R\). If \(N4\) wants to send data to \(N12\), it chooses injection channel \(I_2\). Therefore, data is transferred over a ring link connected to \(N5\) by \(A_{\mu}R\) in \(N4\). Then the bypass link transmits the data through \(C_{\mu}R\) in \(N5\). In \(N6\), data is transferred through the cross link by \(B_{\mu}R\) to \(N10\). In \(N10\), data is sent over the ring link by \(A_{\mu}R\). Data is transferred through ring link without changing direction in \(N11\). Finally, in \(N12\), data is ejected by \(D_{\mu}R\).
QuT checks the status of the destination’s ejection channel to avoid optical stream collisions from multiple sources. To check if the corresponding ejection channel is free, we use a separate optical control network (CN). Our CN is based on a Multiple-Writer-Single-Reader optical bus [104]. As mentioned in Section 2.1.5.2, in MWSR, each node is capable of using all the wavelengths and sending its optical data on all the waveguides. However, a node can only receive data from an assigned waveguide.

The CN uses three kinds of control packets: request, acknowledgment (ACK) and negative acknowledgment (NACK). In the CN, each source node has a dedicated wavelength; a node modulates a control packet on its dedicated wavelength. Therefore, a destination can accept multiple optical streams simultaneously as each will be on a different wavelength. Each source first sends a request packet over the CN waveguide connected to the desired destination. The source must wait for a response from the destination. An ACK is sent by the destination through the CN if the requested ejection channel is available (not being used by another source.
in the data network). Otherwise a NACK will be sent. Once the source receives the ACK, it begins sending data through the data network. If the destination is busy, a subsequent request packet will be sent after a back-off period which is determined by the average time required for sending a packet.

The CN is implemented as several waveguides (Figure 3.5). In the CN, sixteen nodes can accept data from each waveguide by utilizing splitters, which split an optical stream across different paths [130]. However, all nodes can send request packets on each waveguide. This structure helps reduce the waveguide crossings between laser power waveguide and CN waveguides. Hence, CN structure in an N-node QuT has N/16 waveguides and N wavelengths. By using more splitters on a waveguide, we can further reduce waveguide crossings. However, this increases the CN energy consumption, as a request packet is detected by more nodes per waveguide. As an example, if node 15 wants to send data to node N − 2, it will modulate its control packet onto waveguide N/16 − 1. Each control packet has 6 bits: 4 bits for addressing 16 nodes and 2 bits for encoding the packet type. The receiver must have a (N − 1) × 6 bit buffer to hold simultaneous requests from all other nodes.

3.2.6 Methodology

We compare a QuT with 64 and 128 nodes, in terms of latency and energy efficiency, against three alternative optical NoCs: λ-router [20], all-optical Spidergon [80] and Corona[131]. We use PhoenixSim [26], an event-driven simulator, to implement the optical NoCs. PhoenixSim is based on OMNet++ [132], providing a component-based C++ library and discrete-event simulation framework. We have implemented the optical NoCs by connecting optical components defined in PhoenixSim to model their performance and insertion loss. As mentioned earlier, we use modulators along with off-chip multi-wavelength laser [76] to modulate data. The off-chip laser provides continuous waves for the on-chip communication. In other words, the laser supplies the required power for the on-chip communication. Therefore, the maximum insertion loss should be considered for the required laser power, as explained in Section 3.3.2.

λ-router [20] is an all-optical contention-free NoC. It uses passive micro-ring resonators and routes optical streams in accordance with their wavelengths. We use an ideal λ-router which has infinite receiver-side buffering to eliminate the need for an arbitration mechanism. In practice, the λ-router’s performance will be limited when a more realistic buffer size is chosen. The λ-router’s scalability is limited by the number of wavelengths and switches. Since an N-node λ-router uses N/2 × (N − 1) switches, the number of switches increases quadratically with the number of nodes in network. A λ-router with N nodes needs N distinct wavelength sets. A non-ideal λ-router would require a control network just as QuT does to avoid collisions at the destination due to limited receiver-side buffering. We choose to compare against an idealized version which allows us to assess the cost and performance degradation that occurs with the addition of a control network.

Optical Spidergon [80] uses passive optical components but has an electrical control
network. We replace its electrical control network with the optical CN described in Section 3.2.5 to achieve better power and latency. In an all-optical Spidergon, each node has a dedicated but not unique wavelength, used by other nodes to modulate data. An all-optical Spidergon needs \( N/2 \) wavelength sets as each node is connected to its neighbors in a ring and to the node that is \( N/2 \) nodes away. We choose Spidergon to compare to QuT since it is also a ring-based topology and uses cross links to reduce the NoC's diameter.

**Corona** \([131]\) uses an optical crossbar with optical token-ring arbitration to permit a node to send data. It uses a Multiple-Writer-Single-Reader optical bus; when the number of nodes increases, both the waiting time for receiving a token and the network diameter increase. Therefore, we choose the best proposed token-ring arbitration scheme, slot-token-ring arbitration, as a control network of Corona \([130]\). The original Corona implementation uses a lot of wavelengths \((64)\) to modulate data packets to improve the latency; however, this requires an impractically large number of micro-rings (one million) and high power consumption, which rapidly increases with network size. To enable a consistent evaluation, we keep a constant flit-width of 1 byte\(^6\) across all networks. In Corona, the number of wavelengths equals the flit-width. Hence, we reduce the total number of wavelengths in Corona to eight distinct wavelengths. Therefore, our implementation improves Corona’s power consumption and area by reducing the total number of micro-rings and waveguides, but its latency is increased. We choose Corona to compare with QuT since unlike QuT, its number of wavelengths is independent of the number of nodes, it does not need optical switches and it has a very simple architecture to implement the optical crossbar. We believe these three design points cover a range of interesting optical topologies that have been proposed in the literature.

We evaluate these NoCs with 64 and 128-nodes to compare their scalability. We assume the size of the die is the same for 64 and 128 nodes \((225 \text{ mm}^2)\). We keep the optical bandwidth constant for all-optical NoCs; we assign eight distinct wavelengths to each node. Therefore, each optical data stream is modulated on *eight* wavelengths assigned to the corresponding destination. However, each control packet is modulated on one wavelength assigned to the corresponding source. Since the control packets are small, the CN does not require large bandwidth. We assume a data packet size of 256 bits and 10Gb/s modulator and detector. We evaluate the topologies under the following synthetic traffic patterns: random, bitreverse, neighbor, tornado and hotspot in which a random node receives 30% of all requests and the remaining 70% is uniformly distributed. We evaluate these patterns with various offered loads. Offered load \((\alpha)\) is \( T_{md}/(T_{md} + T_p)\), where \( T_{md} \) is the time that a packet takes to be transmitted through the network and \( T_p \) is the exponentially distributed inter-message gap \([115]\).

---

\(^6\)The effect of small flit-width on data serialization latency is mitigated by high-speed optical modulator and low-latency optical transmission.
Table 3.1: Insertion Loss Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C_{\text{FtoW}}$</td>
<td>1 dB [98]</td>
</tr>
<tr>
<td>$L_e$</td>
<td>5 dB [98]</td>
</tr>
<tr>
<td>Waveguide propagation loss</td>
<td>1 dB/cm [93]</td>
</tr>
<tr>
<td>Passing through ring loss</td>
<td>0.5 dB [31]</td>
</tr>
<tr>
<td>Passing by ring loss</td>
<td>0.01 dB [31]</td>
</tr>
<tr>
<td>Waveguide bending loss</td>
<td>0.005 dB [31]</td>
</tr>
<tr>
<td>Waveguide crossing loss</td>
<td>0.12 dB [18]</td>
</tr>
<tr>
<td>Splitter loss</td>
<td>0.1 dB [130]</td>
</tr>
<tr>
<td>Receiver Sensitivity</td>
<td>-17 dBm [144]</td>
</tr>
</tbody>
</table>

3.3 Evaluation

In this section, we evaluate delay, power, energy, throughput and area of QuT, Spidergon, $\lambda$-router and Corona.

3.3.1 Delay Evaluation

The total latency in an optical path includes: delay of the optical switches in the path, delay in bends and waveguide crossings, propagation delay in waveguides (11.4 $\mu$s/µm), modulator delay (23.8 ps) and detector delay (4.2 ps) [80]. Improving modulator bit rate, e.g., from 10 to 40 Gb/s, can improve the latency, since modulating data to an optical stream is one of the major factors in latency. The total delay per packet includes the path delay through the control and data networks and the time the packet waits in the processor’s output buffer to be transferred into the network.

Figures 3.6 and 3.7 show the average packet latency at $\alpha = 0.5$ for different traffic patterns for 64- and 128-node networks, respectively. Latency is reported in processor cycles. We assume a 5 GHz clock for the processors. QuT and Spidergon must check the status of the destination to avoid contention in the ejection channel. This destination checking and the ability to only accept one packet at a destination leads to an increase in latency for QuT and Spidergon, especially in high contention traffic such as hotspot. In hotspot traffic, 64- and 128-node QuTs saturate at $\alpha = 0.44$ and $\alpha = 0.38$, respectively because a destination can only receive from one source at a time.

The latency in 64- and 128-node QuT is up to 32% and 24% higher compared to 64- and 128-node Corona, respectively. Corona uses slot-token-ring arbitration to reserve a destination, which has better performance than QuT’s control network. However, it consumes more power and energy. The latency in a 128-node Corona is 11.5% higher than that of a 64-node Corona. In Corona, token waiting time and network diameter both increase as the number of nodes increases.

Waiting time in a processor’s output buffer coupled with the delay associated with modulat-
ing the packet, primarily contribute to the latency in QuT. When the network size grows from 64 to 128 nodes, the average optical path latency only increases by two cycles. Altogether, QuT’s latency does not rapidly increase for 128 nodes compared to 64 nodes, assuming the packet size, the number of wavelengths used to modulate a packet and the network offered load are the same.

λ-router has better latency compared to QuT since we assume that λ-router can accept data from all of the sources at a time. Buffering for one packet per source is added to each destination. Hence, each node has 4064 bytes of buffering for 128-node λ-router and only 32 bytes for 128-node QuT, Spidergon and Corona. If the buffer size in λ-router is reduced, the network latency will increase sharply.

3.3.2 Power Evaluation

Total power consumption in an optical NoC includes off-chip laser power and on-chip micro-ring heating power.

- Laser Power: Off-chip laser power is constant and independent of network traffic. This
power is computed from the maximum optical insertion loss in the NoC. The insertion loss is the summation of: passing through the micro-ring loss, passing by micro-ring loss, waveguide crossing\textsuperscript{7} and bending loss and waveguide propagation loss.

The laser power per $\lambda$, before entering to the chip, is calculated by $laserpower(\lambda) = -17 + IL + L_e + C_{FtoW}$, where $-17$ is the receiver sensitivity and $L_e$ and $C_{FtoW}$ represent laser efficiency and coupling coefficient, respectively. Finally, the total laser power is a multiplication of laser power per $\lambda$ and total wavelengths in the NoC.

To perform a fair comparison, we use the same optical parameters for all of the optical topologies, listed in Table 3.1. Again, we consider aggressive parameters to show the full opportunity for power savings across these networks. More conservative parameters can increase the power consumption of optical NoCs, especially Corona.

- **micro-ring Heating Power**: We use thermally-tunable micro-rings. Extra power helps these micro-rings maintain their resonance wavelength when they experience temperature

\textsuperscript{7}This includes the crossing between: 1) network waveguides, 2) laser power waveguide and network waveguides.
variation on chip. We use 20 $\mu$W/ring, under typical conditions, when the rings in the system would experience a temperature range of 20K [67].

- **Total Power**: In QuT, Spidergon and Corona, the power of the CN must be factored in. Total power is the summation of total power consumption in the data and control networks.

The power dissipation for various optical networks with 64 and 128-nodes are shown in Tables 3.2 and 3.3. Although QuT has higher optical loss compared with Spidergon, this is compensated by fewer wavelengths. QuT uses 256 wavelengths, while Spidergon needs 512 wavelengths for a 128-node network (Table 3.4). The required wavelengths are determined such that each topology has an 8-bit flit size. micro-rings are sensitive to process and temperature variation. Therefore, they are a major source of faults in an optical NoC. An optical NoC should reduce the number of micro-rings to improve its reliability. QuT requires fewer micro-rings by 32%, 51.1%, 33.5% and 52.4% compared to a 64-node Spidergon and $\lambda$-router, and a 128-node Spidergon and $\lambda$-router, respectively (Table 3.4). We use the following equation to calculate
Chapter 3. Optical NoC

the total number of micro-rings in QuT.

\[ T_{\text{microrings}} = \frac{N}{2} \times \text{SetSize} \times \left( (\text{num}_{A\mu R}(\text{odd}) + \text{num}_{A\mu R}(\text{even})) \times \text{num}_{\text{set}}(A\mu R) + \right. \\
\left. \text{num}_{B\mu R}(\text{even}) \times \text{num}_{\text{set}}(B\mu R) + \text{num}_{C\mu R}(\text{odd}) \times \text{num}_{\text{set}}(C\mu R) + \right. \\
\left. (\text{num}_{D\mu R}(\text{odd}) + \text{num}_{D\mu R}(\text{even})) \times \text{num}_{\text{set}}(D\mu R) + \right. \\
\left. N \times \text{SetSize} \times (\text{num}_{\text{modulators}} + \text{num}_{\text{detectors}}) \right) \]

\[ \text{num}_{A\mu R}(\text{odd}) = 2 \quad \text{num}_{A\mu R}(\text{even}) = 4 \quad \text{num}_{B\mu R}(\text{even}) = 2 \quad \text{num}_{C\mu R}(\text{odd}) = 2 \]

\[ \text{num}_{D\mu R}(\text{odd}) = 3 \quad \text{num}_{D\mu R}(\text{even}) = 3 \quad \text{num}_{\text{set}}(A\mu R) = \frac{N}{4} \quad \text{num}_{\text{set}}(B\mu R) = \frac{N}{4} \]

\[ \text{num}_{\text{set}}(C\mu R) = 1 \quad \text{num}_{\text{set}}(D\mu R) = 1 \quad \text{num}_{\text{modulators}} = \frac{N}{4} \quad \text{num}_{\text{detectors}} = 1 \quad \text{SetSize} = 8 \]

In this equation, \( N \) is the number of the nodes in the system. For example, \( \text{num}_{A\mu R}(\text{odd}) \) and \( \text{num}_{A\mu R}(\text{even}) \) shows the number of \( A\mu R \) micro-ring resonators in odd and even switches, respectively. Also, \( \text{num}_{\text{modulators}} \) and \( \text{num}_{\text{detectors}} \) are the number of modulators and detectors per node. In our design, we are using 8 wavelengths to modulate the optical data. Thus, \( \text{SetSize} \) is equal to 8.

Power consumption increases for the 128-node optical NoCs due to increases in IL, the number of micro-rings and the total number of wavelengths. Reducing the number of wavelengths required to modulate data can reduce the power consumption. There is a trade-off between bandwidth and power consumption in an optical NoC. Future innovation in detector sensitivity will have a positive effect on the laser power [98]. For example, by using a receiver that requires minimum power of -20 dBm, laser power of 128-node QuT is 2.63W, half of the laser power with a receiver requiring minimum power of -17 dBm (Table 3.3). Among these optical NoCs, Corona’s total number of wavelengths remains constant when the network size grows. However, Corona and its CN have the largest IL for 64 nodes among the other optical NoCs as shown in Table 3.2. Therefore, increasing its IL, its CN IL and the number of wavelengths for 128 nodes, dramatically increases Corona’s power consumption. The IL increases because of longer waveguide length and more waveguide crossings.

64-node QuT consumes up to 23%, 73% and 8% less power than Spidergon, \( \lambda \)-router and Corona, respectively. However, 128-node QuT achieves power reductions of up to 23.5%, 86.4% and 52.9% over Spidergon, \( \lambda \)-router and Corona, respectively.

3.3.3 Energy Evaluation

Energy consumption in an optical NoC includes the energy dissipation in the laser, micro-ring heating and back-end circuitry, electrical to optical (E/O) and optical to electrical (O/E) conversion. Electrical parts of an optical architecture are implemented on the electrical layer. This electrical layer send the electrical charges to the optical layers using TSVs. The electrical parts include the serializer and driver circuitry in the transmitters, and deserializer and amplifier
Figure 3.8: Average Energy-per-bit. Traffic Patterns (excluding hotspot) use $\alpha = 0.5$.

in the receivers. Based on the model in Section 3.3.2, Figures 3.8a and 3.8b show the average energy-per-bit for different optical NoCs at $\alpha = 0.5$ for different traffic patterns. E/O and O/E conversions consume 100 fJ/b [98].

In QuT and Spidergon, the average energy-per-bit includes the energy consumption in the data network and in the CN for multiple request packets per data packet if necessary. Energy consumption in Corona includes the energy consumption in both the data and control network. Since QuT has lower power dissipation and a smaller average optical path delay, it has lower energy-per-bit compared to Corona and $\lambda$-router. QuT achieves an energy-per-bit reduction of up to 69%, 92% over 128-node $\lambda$-router and Corona, respectively.

In Tornado traffic, a large number of packets are transferred between nodes with $N/2$ distance from each other; for a 128-node network, the average optical path delay increases. Thus, a data stream is sent through more optical switches and waveguides in QuT. Conversely, Spidergon benefits from this traffic pattern, since a data stream passes through less optical switches and waveguides; it has 34% reduction in energy-per-bit compared with QuT. However, QuT has fewer wavelengths and micro-rings and lower power consumption.

For hotspot traffic in QuT and Spidergon, a large number of requests are sent through the CN for a specific node. Due to destination contention, this results in a sharp increase in
energy-per-bit (Figures 3.8a and 3.8b). At the saturation point in QuT and Spidergon, a small fraction of the energy-per-bit is related to the data network, e.g., the data network consumes 9.11% and 7.2% of the energy-per-bit in the 64-node QuT and Spidergon, respectively. Corona’s CN always consumes energy even when the data network is idle, as Corona has a slot-token-ring arbitration. Also, Corona’s data network has an order of magnitude larger energy consumption compared with the CN. Thus, the average energy-per-bit of Corona at the saturation point in hotspot traffic does not increase significantly. Since \( \lambda \)-router does not require a control network and a destination can accept data from all sources simultaneously, it has lower energy-per-bit at saturation point.

### 3.3.4 Throughput Evaluation

Effectively, QuT trades off throughput for lower cost and lower power consumption. Figure 3.9 shows the normalized average throughput-per-watt for optical NoCs.\(^8\) The 64-node QuT has lower throughput-per-watt than Corona, because both networks are low-power and Corona has better latency than QuT. However, since power consumption of the 128-node Corona rapidly increases (Section 3.3.2) and QuT consumes less power, throughput-per-watt of Corona decreases compared to QuT (specifically on hotspot traffic). In hotspot traffic for the 128-nodes, the ability of \( \lambda \)-router to simultaneously accept data from all sources at a destination mitigates the effect of its high power consumption on throughput-per-watt compared to QuT. The 128-node QuT achieves throughput-per-watt improvements of up to 43%, 28% and 85% over the 128-node Corona, Spidergon and \( \lambda \)-router, respectively, which indicates better scalability for QuT.

### 3.3.5 Area Evaluation

Area overhead in an optical NoC includes the area required for micro-rings, waveguides and detectors. Table 3.5 shows the normalized area for optical NoCs considered in this work. The 128-node Spidergon and \( \lambda \)-router require 44% and 154% more optical area than QuT, since they have more micro-rings. \( \lambda \)-router uses more detectors to allow each node to accept data from different sources simultaneously. Although Corona has fewer micro-rings, it needs more waveguides and detectors compared to QuT. Therefore, its area is almost equal to QuT.

\(^8\)Maximum power is considered.
3.3.6 Discussion

QuT outperforms Spidergon, λ-router and Corona in terms of power dissipation and achieves better energy-per-bit consumption compared with λ-router and Corona. Although Spidergon slightly outperforms QuT in energy-per-bit, QuT has lower power consumption, fewer wavelengths (reduced by a factor of 2) and micro-rings. Due to lower resource requirements, QuT is easier to implement. QuT also uses fewer micro-rings and wavelengths compared with λ-router.

To have a low-power optical NoC, all of the optical factors mentioned in Section 3.1 should be considered. Having a topology only with smaller IL or fewer wavelengths does not lead to the best design choice in terms of power consumption.

Although QuT scales better compared to other optical NoCs, its power consumption does not scale well with node count. Therefore, without improving detector sensitivity or reducing optical bandwidth to compensate for the extra IL with increasing network size, QuT is not suitable for networks larger than 128 cores. However, if we use electrical or optical clustering, we can use QuT for systems larger than 128 cores.

Electrical clustering: several cores can be...
clustered to connect to one optical node; e.g., a 128-node QuT with cluster size of 4 can be used to implement a 512-core system. **Optical clustering**: a single large optical NoC can be divided into several smaller parallel optical NoCs by using 3D stacking [98]; e.g., in a 256-core system, instead of one 256-node QuT, we can use six 128-node QuTs to connect all of the cores. For this design, we assume 4 partitions of nodes; each one includes 64 nodes: $P_1$ includes node 0 to node 63, $P_2$ includes node 64 to node 127, $P_3$ includes node 128 to node 191, $P_4$ includes node 192 to node 255. We can connect all groups by six different combinations; each combination has 128 nodes. These optical NoCs are implemented in different optical layers to reduce the optical losses. Also, by using the electrical and optical clustering, we can connect 1024 cores with 6 parallel 128-node QuTs with electrical clustering size of 4.

### 3.4 Concluding remarks

In this chapter, we present QuT, an all-optical NoC using passive micro-rings to address challenges such as power, the number of micro-rings and wavelengths. We propose a novel deterministic wavelength routing algorithm to optimize optical switches, reduce the number of wavelengths and micro-rings and lower the IL. QuT consumes less power and energy which allows it to scale better than state-of-art proposals. When the network size is increased, QuT is able to achieve lower power compared to Corona, $\lambda$-router and Spidergon.
Chapter 4

Exploration of Thermal Constraints for 3D Network-on-Chip

In this chapter, we examine the second of two solutions that focuses on the interconnect infrastructure as explained in Chapter 1. In Sections 4.3, 4.4 and 4.5, we propose thermal-aware mapping algorithms. Our methodology is presented in Section 4.6. Section 4.7 presents the effects of these algorithms on temperature and performance.

4.1 Introduction

The increase in chip power density due to increasing the transistor counts and operating frequency can cause the chip temperature to rise. Employing 3D stacking aggravates this problem. To avoid the negative effects of higher temperatures, such as transistor performance degradation, increased static power consumption, increased RC delay in interconnects, and reliability issues, we need to design thermal management algorithms. Although the power consumption of a 3D IC is less than that of the equivalent 2D chip due to the decrease in average interconnection length, power density of a 3D IC is higher due to smaller chip area and the existence of the vertical layers. Also, the decrease in overall interconnect length can improve the operating frequency of a 3D chip, which can also contribute to the growth of chip power density [35, 106]. In addition, the increase in transistor count and the higher operating frequency increase chip power consumption, which results in a higher power density on chip, which in turn increases the chip temperature.

In this chapter, we focus on mapping algorithms to study the effects of thermal and performance constraints on the chip temperature. Application mapping greatly impacts performance and power consumption. A mapping algorithm decides how various tasks are assigned to cores.
based on optimization criteria [96]. The criteria used by the mapping algorithm impacts the quality of the obtained results and the complexity of the algorithm. Therefore, we need to explore temperature constraints for thermal-aware mapping. We propose three static thermal-aware mapping algorithms utilizing integer linear programming (ILP). Then, we study different performance and thermal constraints and their impacts on chip temperature. We explore answers to the following questions to lay the foundation for the future research:

- Is it important for chip temperature that the mapping algorithm consider thermal constraints, such as power or temperature?
- Which thermal constraints are sufficient to achieve an acceptable chip temperature: implicit constraints such as power estimation, or explicit constraints such as temperature?
- To what extent do these constraints affect the chip temperature?
- Do these constraints negatively affect performance and power consumption?
- Is an implicit constraint such as router power consumption, which is used to estimate the temperature of a tile, sufficient to obtain the optimal power and performance?

A suitable mapping algorithm must be chosen for this exploration. A heuristic search algorithm such as a genetic algorithm accelerates the process of finding reasonably good solutions; however, an approximation algorithm, can guarantee that the solutions are close to optimum [47]. Unlike heuristic algorithms, we can analytically verify the performance of an approximation algorithm. We focus on employing a static mapping algorithm in order to extensively explore the constraints in thermal-aware mapping. Dynamic algorithms must also be concerned with algorithm run time, which makes it difficult to isolate the effect of optimization criteria.

### 4.2 Static Thermal-aware Mapping

We consider a system with homogeneous processing elements (PE). Since no PE is specialized for a particular task, a task can be scheduled on any PE. We assume that the applications of interest can be broken down into tasks where the communication requirements are known at design time. The mapping algorithm takes information related to the tasks, their communication and the topology structure in order to find a PE for each task to run on, while it considers metrics such as performance, power consumption or temperature to be optimized.

In the remainder of this section, we propose three thermal-aware mapping algorithms. Our mapping algorithms take the communication task graph (CTG), as shown in Figure 4.1, as the input to schedule the tasks on the PEs such that chip temperature is optimized. The $CTG = G(T, E)$ is a directed graph. In a CTG, each $T_i \in T$ and each $e_{i,j} \in E$ indicate the task and communication dependence between $T_i$ and $T_j$ with communication rate $w_{i,j} \in W$ bytes per second, respectively [96]. For example, in Figure 4.1, task $T_4$ communicates with task $T_6$ with communication rate 362 MB/s.
4.3 OptHeraMap: Optimal thermal-aware mapping

In this section, we propose an ILP-based algorithm, $OptHeraMap$, to map the tasks on PEs to minimize the maximum temperature of the chip while considering bandwidth and performance constraints. $OptHeraMap$ employs technology and NoC design information for task mapping, as shown in Figure 4.2. Technology information includes the thickness of wires, interlayer dielectric (ILD), glue layer, silicon layer, NoC’s topology, and routing algorithm. This information is utilized to extract thermal resistances for all PEs 1, and the 3D NoC’s information such as the neighbors of each PE and all paths between pairs of PEs in the routing algorithm 2. Furthermore, $OptHeraMap$ requires the NoC’s physical information, such as the channel width between routers, bit-power consumption of the router, and the operating frequency of the NoC to estimate a tile’s temperature 3. A tile includes a PE and its associated router. The exact bit-power consumption of each router is not known until the end of the third phase; therefore, we use an estimate from the Orion2 library [133]. Next, $OptHeraMap$ maps tasks on PEs 4. After mapping, simulation calculates the power consumption of each router, the total power consumption and the total latency 5. Finally, the temperature of each tile is calculated 6.

In the next sections, we elaborate on $OptHeraMap$’s ILP objective function, constraints
and its parameters and variables. Then, we discuss its linearization and complexity.

4.3.1 Objective function

Our objective is to minimize the maximum temperature caused by running the tasks on PEs. A fast thermal model is important to have an efficient ILP-based thermal-aware mapping. Furthermore, thermal capacitances are not required to calculate the steady state temperature of each PE. Therefore, we use a simplified closed-form thermal model [62]. This model only considers the vertical heat-dissipating path. Although this compact model is not as accurate as finite element methods [30], finite difference methods [126] or compact resistive network [119], it is the fastest thermal model. This closed-form model can be easily replaced by compact resistive network [119], which also considers the horizontal heat-dissipating path, at the expense of speed of ILP-based thermal-aware mapping.

\[
T_{hi,j,k} = T_{Amb} + \sum_{m=1}^{k} \frac{R_{i,j,m}}{A} \times \left( \sum_{s=m}^{n} PT_{i,j,s} \right)
\]

where \( PT_{i,j,s} = P_{i,j,s} + PR_{i,j,s} \)

\[
P_{i,j,s} = \sum_{\forall T_k \in E} power_{T_k} \times M_{T_i}^{Pvi,j,s}
\]

where \( PR_{i,j,s} = \sum_{\forall L_u \text{ connected to } R_{i,j,s}} \lambda_{L_u} \times PRpB \)

where:

- \( T_{hi,j,k} \) shows the temperature of the PE at position \((i, j, k)\). The heat sink is connected to the first layer in the 3D chip and \(n^{th}\) layer is the farthest layer from the heat sink.

- \( T_{Amb} \) is the ambient temperature.

- \( R_{i,j,m} \) is the thermal resistance of the PE at position \((i, j, m)\). \( R_{i,j,m} = (t_{Si}/k_{Si}) + (t_{glue}/k_{glue}) + (t_{ins}/k_{ins}) \), where \( t_{Si} \), \( t_{glue} \), and \( t_{ins} \) are the thickness of \(i^{th}\) silicon, glue and insulator layers, respectively. Here, \( k_{Si} \), \( k_{glue} \) and \( k_{ins} \) are the thermal conductivity of \(i^{th}\) silicon, glue and insulator layers, respectively.

- \( A \) is the area of each PE in the formulae for temperature calculation.

- \( P_{i,j,s} \) is the average power consumption of the PE at position \((i, j, s)\), which is calculated based on the average power consumption of the task(s) that runs on it. For our evaluation, we assign a random power to each task based on the average power consumption of the PE (in the range of 10 to 60 W/cm\(^2\) for the specific technology [125]).

- \( PR_{i,j,s} \) is the average power consumption of the router at position \((i, j, k)\). The router power is calculated based on the router power consumption per bit and the amount of
4.3.2 Constraints

*OptHeraMap* has four constraints, as shown in Algorithm 1, to meet the requirements of performance and mapping.

- The first constraint controls the task assignment such that each task is mapped to only
Algorithm 1 LP formulae of optimal thermal-aware mapping (OptHeraMap)

Minimize $H$

\[ \text{where } H = \max \{ \text{Temp}_p : p = 1 \cdots m, \text{ for a system of } m \text{ cores} \} \]

Subject to :

1) $\forall T_m \in T : \sum_{\forall P_e_i \in PE} M_{T_m}^{P_e_i} = 1$

\[ M_{T_m}^{P_e_i} = \begin{cases} 1 & \text{if map} (T_m) = P_e_i \\ 0 & \text{else} \end{cases} \]

2) $\forall P_e_i \in PE : \sum_{\forall T_m \in T} M_{T_m}^{P_e_i} < \text{Virt}_{P_e_i}$

3) $\forall L_k \in \text{Link} : \lambda_{L_k} \leq BW_{L_k}$

\[ \lambda_{L_k} = \sum_{\forall (T_m, T_n)} \sum_{i=1}^{\left| PE \right|} \sum_{j=1}^{\left| PE \right|} w_{T_m, T_n} \times L_{E_{L_k}}^{P_e_i, P_e_j} \times M_{T_m}^{P_e_i} \times M_{T_n}^{P_e_j} \]

\[ \text{where } L_{E_{L_k}}^{P_e_i, P_e_j} = \begin{cases} 1 & \text{if } L_k \in \text{Link}(P_e_i, P_e_j) \\ 0 & \text{else} \end{cases} \]

4) Comm. Cost $\leq CC_{th}$

\[ \text{Comm. Cost} = \sum_{\forall (T_m, T_n)} \sum_{i=1}^{\left| PE \right|} \sum_{j=1}^{\left| PE \right|} w_{T_m, T_n} \times \text{dist}(P_e_i, P_e_j) \times M_{T_m}^{P_e_i} \times M_{T_n}^{P_e_j} \]

\[ \text{where } \text{dist} (P_e_i, P_e_j) = \text{manhatan distance between } P_e_i \text{ and } P_e_j \]
one PE.\textsuperscript{2}

- The second constraint controls the number of tasks that can be run on each PE.
- The third constraint ensures that the total communication rate of the tasks assigned to each PE is less than the total available bandwidth of the link to reduce the congestion.
- The fourth constraint is the explicit performance constraint. To simplify the model, we use zero load latency and ignore the congestion in routers. However, we consider buffering delay when evaluating our design in Section 4.7.

### 4.3.3 Parameters and variables

In this section we explain the parameters and variables that are used in the LP formulae in Algorithm 1.

- We present the LP variables as the binary matrix of \(M\) with \(|PE| \times |T|\) elements, where \(|PE|\) and \(|T|\) are the number of PEs and the number of tasks in the CTG, respectively. Each element of the \(M\) matrix, \(M_{i,j}\), will be 1 or 0 depending on whether the task \(T_{m}\) is mapped on the \(P_{e_i}\) or not.
- The parameter \(w_{T_{i},T_{j}}\) is the communication rate between tasks \(T_i\) and \(T_j\).
- The parameter \(BW_{L_k}\) is the maximum available bandwidth of link \(L_k\).
- The parameter \(Virt_{P_{e_i}}\) is the maximum number of tasks that can run simultaneously on the \(P_{e_i}\).
- The parameter \(CC_{th}\) is the average communication cost in the NoC topology, which is calculated based on the average distance for the 3D mesh as follows:

\[
CC_{th} = \frac{1}{3} \sum_{(T_i,T_j) \in E} w_{T_i,T_j} \left( d_1 - \frac{1}{d_1} \right) \left( d_2 - \frac{1}{d_2} \right) \left( d_3 - \frac{1}{d_3} \right)
\]

where \(d_1\), \(d_2\), and \(d_3\) are dimensions of the 3D Mesh.
- The set of links between \(P_{e_i}\) and \(P_{e_j}\), based on the NoC routing algorithm, is \(Link(P_{e_i},P_{e_j})\).

### 4.3.4 Linearization

The variables in the objective and the third constraint appear as multiplication. Therefore, they cannot be solved by utilizing LP. In order to transform them into the LP model, we insert the following extra variables [29]:

\textsuperscript{2}If a task requires more resources to meet its timing constraints, more PEs can be assigned to the task by changing the one in the equation to the larger number, e.g., two, assuming the task can be split on several PEs.
\[ M_{Tu}^{Pe_i} + M_{Tv}^{Pe_j} - 1 \leq M_{Tu}^{Pe_i} \times M_{Tv}^{Pe_j} \leq \frac{M_{Tu}^{Pe_i} + M_{Tv}^{Pe_j}}{2} \] (4.1)

\[ 0 \leq M_{Tu}^{Pe_i} \times M_{Tv}^{Pe_j} \leq 1 \] (4.2)

The Formula 4.1 determines lower and upper bounds for the quasi-convex function. The lower bound underestimates the non-convex function, which is derived from multiplication of two convex functions [6]. We can impose the upper bound on them since they are binary variables.

### 4.3.5 Algorithm complexity

We employ the simplex method [102] to solve the LP problem. On average, the complexity of the simplex method is polynomial to the number of the variables. Moreover, we use branch and bound to solve the ILP model. The branch and bound algorithm explores branches in the tree of the complete solution space. Each branch represents a subset of solution space. A branch can be pruned by comparing against upper or lower bounds on the optimal solution. Therefore, the complexity of OptHeraMap increases exponentially with the number of variables. As mentioned in Section 4.3.3, the variable matrix M has \(|PE| \times |T|\) elements. In addition, we add extra variables to linearize the problem. The total number of extra variables is on the order of \(|PE|^2 \times |E|\), because nonlinear variables appear as the tensor product of M and M. As a result, we can calculate the complexity of the algorithm as follows: \(O(b(Var_1) \times g(Var_2))\)

Where:

- \(Var_1\) is the number of variables in matrix M.
- \(Var_2\) is on the order of extra variables.
- \(b(Var_1)\) is the total number of branches in the branch and bound algorithm.
- \(b(Var_1) = O(2^{Var_1})\).
- \(g(Var_2)\) is the complexity of the simplex method, which is polynomial. \(g(Var_2) = O(Var_2^3)\).

The number of variables directly affect the algorithm execution time. Therefore, it is not efficient to use this model for the next generation of chips; e.g., for 128 PEs and 100 edges in the CTG, the number of variables is 1,638,400. Due to this complexity, we propose two faster mapping algorithms in Sections 4.4 and 4.5.

### 4.4 PartiMap: Partitioning-based mapping

In this algorithm, we map the tasks onto PEs using hierarchical partitioning. Figure 4.4 shows the walk-through example. First, we have one partition which contains all of the tasks, Figure 4.4a. At each step, we employ ILP to split each partition into two partitions, Figure 4.4b.
Figure 4.4: PartiMap: walk-through example. In each figure, the left part represents the NoC structure and the right part represents the task partitions.
The minimum cut between two partitions is determined using bandwidth constraints. Furthermore, we consider the power consumption balancing constraint to improve the hotspot temperature. Task power consumption balancing means that we try to distribute the tasks across two partitions, so that the sum of one partition’s task power is on the order of the corresponding sum in the other partition. At the end of each step, the number of partitions is doubled. Simultaneously, we partition the PEs with horizontal and vertical cuts (X and Y dimensions) based on the level of task partitioning in the current step, Figures 4.4c and 4.4d. We proceed accordingly until the number of PEs in each partition is less than or equal to the number of layers in the third (Z) dimension, Figure 4.4e. Then, we sort and map the tasks in each partition based on their power consumptions; i.e., the task with the highest power consumption is mapped on the PE nearest the heat sink. The LP formulae are shown in Algorithm 2.

4.4.1 Objective function

Our objective is to minimize the cut between two partitions, as shown in Algorithm 2:

\[ \text{Cut} = \sum_{(T_i, T_j) \in E} w_{T_i, T_j} \times P_{T_i}^{C_l_i} \times P_{T_j}^{C_l_j} + w_{T_i, T_j} \times P_{T_i}^{C_l_1} \times P_{T_j}^{C_l_0} \]

Where:

- \( C_l_i \) is the \( i^{th} \) partition.
- The variable matrix is binary matrix P with \( |C_l| \times |T| \) elements, where \( |C_l| \) and \( |T| \) are the number of partitions created at each step and the number of tasks in the CTG, respectively. Each element of the P matrix, \( P_{T_m}^{C_l_i} \), will be 1 or 0 depending on whether task \( T_m \) is assigned to the partition \( C_l_i \) or not.
- The parameter \( w_{T_i, T_j} \) is the communication rate between tasks \( T_i \) and \( T_j \).

4.4.2 Constraints

PartiMap has nine constraints, as shown in Algorithm 2, to meet the requirements of performance, temperature and mapping.

- The first constraint controls the task assignment such that each task belongs to only one partition.
- The second and third constraints control the number of tasks assigned to each partition based on the number of PEs in each partition.
- The fourth to seventh constraints are bandwidth constraints. The total communication rate of the tasks assigned to each partition should be less than the total available bandwidth inside of the partition, constraints 6 and 7. Moreover, the total communication
Algorithm 2 LP formulae of partitioning-based mapping

Minimize Cut

\[
\text{where Cut} = \sum_{\forall (T_i, T_j) \in E} w_{T_i, T_j} \times P_{T_i}^{Cl_0} \times P_{T_j}^{Cl_1} + w_{T_i, T_j} \times P_{T_i}^{Cl_1} \times P_{T_j}^{Cl_0}
\]

Subject to :

1) \( \forall T_m \in T : P_{T_m}^{Cl_0} + P_{T_m}^{Cl_1} = 1 \)
   where \( P_{T_m}^{Cl_i} = \begin{cases} 1 & \text{if Partition } (T_m) = Cl_i \\ 0 & \text{else} \end{cases} \)

2) \( \sum_{\forall T_m \in T} P_{T_m}^{Cl_0} = K_0 \)

3) \( \sum_{\forall T_m \in T} P_{T_m}^{Cl_1} = K_1 \)

4) \( \sum_{\forall (T_i, T_j) \in E} w_{T_i, T_j} \times P_{T_i}^{Cl_0} \times P_{T_j}^{Cl_1} \leq W_{\text{cut_LtoR or cut_DtoU}} \)

5) \( \sum_{\forall (T_i, T_j) \in E} w_{T_i, T_j} \times P_{T_i}^{Cl_0} \times P_{T_j}^{Cl_1} \leq W_{\text{cut_RtoL or cut_UtoD}} \)

6) \( \sum_{\forall (T_i, T_j) \in E} w_{T_i, T_j} \times P_{T_i}^{Cl_0} \times P_{T_j}^{Cl_1} \leq W_{\text{flow_{Cl0}}} \)

7) \( \sum_{\forall (T_i, T_j) \in E} w_{T_i, T_j} \times P_{T_i}^{Cl_1} \times P_{T_j}^{Cl_1} \leq W_{\text{flow_{Cl1}}} \)

8) \( \sum_{\forall T_i \in T} \text{Pow}_{T_i} \times P_{T_i}^{Cl_0} \leq (\text{Avg}_{\text{power}} + \chi) \times K_0 \)

9) \( \sum_{\forall T_i \in T} \text{Pow}_{T_i} \times P_{T_i}^{Cl_1} \leq (\text{Avg}_{\text{power}} + \chi) \times K_1 \)
rate of the tasks between two partitions should be less than the total available bandwidth between partitions, constraints 4 and 5.

- We use the eighth and ninth constraints to balance the power consumption of tasks between partitions.

### 4.4.3 Parameters and Variables

The other parameters and variables used in Algorithm 2 are as follows:

- The parameters $K_0$ and $K_1$ are the number of the tasks that can be assigned to the partition.
- The parameters $W_{\text{cut}_{\text{LtoR}}}$ or $W_{\text{cut}_{\text{DtoU}}}$ and $W_{\text{cut}_{\text{RtoL}}}$ or $W_{\text{cut}_{\text{UtoD}}}$ are the total available bandwidth between PE partitions from left (down) to right (up) and right (up) to left (down), respectively.
- The parameters $W_{\text{Flow}_{\text{cl0}}}$ and $W_{\text{Flow}_{\text{cl1}}}$ are the total available bandwidth inside of each PE partition. Sometimes it is impossible to map tasks such that all bandwidth requirements are satisfied. To handle such cases, we increase the total available intra- and inter-partition bandwidth to let the mapping process proceed. Moreover, because at each step of partitioning, we do not have enough detailed information about the mapping, we do not consider the constraints on single links. These modifications model the performance degradation that will occur in practice.
- The parameter $\text{Pow}_{T_i}$ is the average power consumption of task $T_i$ if it is run on the PE.
- The parameters $\text{Avg}_{\text{power}}$ and $\chi$ are the mean and the standard deviation of power consumption of all tasks that should be partitioned. This bound has been specified empirically, since our experiments showed that amounts smaller than one standard deviation make the LP model infeasible, while larger values make this upper bound so large that the related constraint cannot bound the LP model.

### 4.4.4 Linearization

In the 4th to 7th constraints, variables appear as multiplication. We employ a method similar to that of OptHeraMap, Section 4.3.4, and insert the following extra variables for linearization.

\[
P_{T_u}^{C_{ij}} + P_{T_v}^{C_{ij}} - 1 \leq P_{T_u}^{C_{ij}} \times P_{T_v}^{C_{ij}} \leq \frac{P_{T_u}^{C_{ij}} + P_{T_v}^{C_{ij}}}{2}
\]

\[
0 \leq P_{T_u}^{C_{ij}} \times P_{T_v}^{C_{ij}} \leq 1
\]
4.4.5 Algorithm complexity

As mentioned in Section 4.4.1, the variable matrix P has $|CL_i| \times |T_i|$ elements at $i^{th}$ iteration of partitioning. As the partitioning algorithm splits a partition into two partitions, $|CL_i|$ is equal to 2 at each iteration of partitioning. $|T_i|$ is the number of tasks in the partition in the $i^{th}$ iteration. In the first iteration, $|T_i|$ is equal to $|T|$ in CTG. In addition, to linearize the problem we add extra variables. The total number of extra variables is on the order of $|CL_i|^2 \times |E_i|$, because nonlinear variables appear as the tensor product of P and P. In the first iteration, $|E_i|$ is equal to $|E|$ in CTG. The number of variables in the first iteration of this model leads to a complexity of $O(|E|+|T|)$.

The branch and bound and the simplex method are used to solve the ILP model and the LP problem respectively. We can calculate the complexity of algorithm as follows:

$$O \left( \sum_{i=1}^{u} 2^{i-1} b_i ( \text{Var}_{1,i} ) \times g_i ( \text{Var}_{2,i} ) \right)$$

Where:

- $\text{Var}_{1,i}$ is the number of variables in matrix P in the $i^{th}$ iteration.
- $\text{Var}_{2,i}$ is the number of extra variables and variables in matrix P, $O(|E_i|+|T_i|)$ in the $i^{th}$ iteration.
- $b_i ( \text{Var}_{1,i} )$ is the total number of branches in the branch and bound algorithm in the $i^{th}$ iteration.
- $b_i ( \text{Var}_{1,i} ) = O(2^{\text{Var}_{1,i}})$.
- $g_i ( \text{Var}_{2,i} )$ is the complexity of the simplex method, which is polynomial in the $i^{th}$ iteration.
- $u$ is the total number of iterations in the hierarchical partitioning. Because we split partitions until the number of PEs in each partition is less than or equal to the number of layers in the third (Z) dimension, $u$ is equal to $\log_2 \frac{|PE|}{|Z|} + 1$.

Due to the fact that the number of variables at each iteration of partitioning in PartiMap is reduced, we can derive the following equation.

$$b_{i-1} ( \text{Var}_{1,i-1} ) \times g_{i-1} ( \text{Var}_{2,i-1} ) \geq b_i ( \text{Var}_{1,i} ) \times g_i ( \text{Var}_{2,i} )$$

$$\sum_{i=1}^{u} 2^{i-1} \leq 2^u$$

As a result, the complexity of algorithm is $O(2^u b_1 ( \text{Var}_{1,1} ) \times g_1 ( \text{Var}_{2,1} ))$. 
Figure 4.5: ApproxiMap: walk-through example
4.5 ApproxiMap: approximation mapping

PartiMap reaches its final result quickly because it does not consider detailed information for thermal estimation and its main emphasis is on power balancing. To see how important this parameter is in achieving a balanced temperature, we propose another algorithm which takes more information into account. ApproxiMap considers similar information to OptHeraMap, but can be solved much faster. To improve system performance, this algorithm has two phases: partitioning and merging. Figure 4.5 shows a walk-through example. In the partitioning phase, we employ PartiMap to attain several smaller sub-graphs from the CTG, Figures 4.5a and 4.5b. In the merging phase, we use OptHeraMap to map and merge the sub-graphs created in the first phase. Before running OptHeraMap, we search within the sub-graphs to find the partition whose sum of internal communication rate is the highest in all sub-graphs, Figure 4.5c. After we find its mapping using OptHeraMap, we search the rest of the sub-graphs to find the one that has the highest communication rate with the first mapped sub-graph, Figure 4.5d. Then, we merge these two sub-graphs into a larger sub-graph and employ OptHeraMap to map the remaining unmapped tasks. We proceed with the merging process until all tasks in the CTG are mapped, Figure 4.5e and Figure 4.5f. Merging the sub-graphs does not increase the number of undetermined variables. This is because all tasks in the sub-graphs are mapped before each merging in the second phase.

4.5.1 Algorithm complexity

We split partitions until the number of tasks in each partition is less than or equal to \( k \), then we use OptHeraMap on each partition to map the tasks. As a result, the number of iterations in PartiMap is equal to \( \log_2 \left( \frac{|T|}{k} \right) + 1 \), and OptHeraMap is used \( \frac{|T|}{k} \) times. For simplicity, to extract this equation we assume that \( |T| \) and \( k \) are powers of 2. Moreover, in OptHeraMap, the number of variables in matrix M for each partition is \( k^2 \), and the number of extra variables is \( k^2 \times |E_k| \), where \( |E_k| \) is the number of edges among tasks in the partition. As a combination of OptHeraMap and PartiMap is used in ApproxiMap, we can derive its complexity as follows:

\[
O(\text{PartiMap}) + \frac{|T|}{k} \times O(\text{OptHeraMap})
\]

4.6 Methodology

The proposed mapping algorithms take the technology parameters, topology information, and system-level information as input, and formulate them as a LP model. Then, we utilize Integer Linear Programming to solve the model. Table 4.1 shows the technology information extracted from ITRS [4]. We implemented the algorithms in C++, and used the LPSolve package [13] to solve the linear model. The LPSolve package uses the Simplex Method to solve the linear model. We wrote a branch and bound algorithm to solve the ILP based on the LP model.
Table 4.1: Technology Information (45 nm) [4]

<table>
<thead>
<tr>
<th></th>
<th>12</th>
<th>1.1</th>
</tr>
</thead>
<tbody>
<tr>
<td>No. of metal layers</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TSV area($m^2$)</td>
<td>25e-12</td>
<td></td>
</tr>
<tr>
<td>$V_{dd}$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$T_{Amb}$ (C)</td>
<td>25</td>
<td></td>
</tr>
<tr>
<td>Width(m)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Thickness(m)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cu local</td>
<td>45e-9</td>
<td>81e-9</td>
</tr>
<tr>
<td>Cu semi-global</td>
<td>45e-9</td>
<td>81e-9</td>
</tr>
<tr>
<td>Cu global</td>
<td>67.5e-9</td>
<td>162e-9</td>
</tr>
<tr>
<td>Thermal conductivity(w/mk)</td>
<td>72e-9</td>
<td>72e-9</td>
</tr>
<tr>
<td>Si</td>
<td>150</td>
<td></td>
</tr>
<tr>
<td>Cu</td>
<td>400</td>
<td>148e-9</td>
</tr>
<tr>
<td>Glue</td>
<td>0.25</td>
<td>2e-6</td>
</tr>
<tr>
<td>ILD</td>
<td>0.2</td>
<td>10e-6</td>
</tr>
<tr>
<td>Si bulk</td>
<td></td>
<td>500e</td>
</tr>
</tbody>
</table>

Table 4.2: Benchmarks’ information

<table>
<thead>
<tr>
<th>Name</th>
<th>T</th>
<th>E</th>
<th>NoC (x,y,z)</th>
<th>Name</th>
<th>T</th>
<th>E</th>
<th>NoC (x,y,z)</th>
</tr>
</thead>
<tbody>
<tr>
<td>GSM Coder[113]</td>
<td>53</td>
<td>81</td>
<td>4,4,4</td>
<td>MPEG4[14]</td>
<td>12</td>
<td>23</td>
<td>2,3,2</td>
</tr>
<tr>
<td>GSM Decoder[113]</td>
<td>34</td>
<td>48</td>
<td>3,4,3</td>
<td>VOPD[14]</td>
<td>12</td>
<td>15</td>
<td>2,3,2</td>
</tr>
<tr>
<td>MMS[61]</td>
<td>29</td>
<td>23</td>
<td>3,4,3</td>
<td>MWD[14]</td>
<td>12</td>
<td>13</td>
<td>2,3,2</td>
</tr>
</tbody>
</table>

results instead of using branch and bound option in LPSolve. This gives us more control on creating and cutting branches effectively. For simulating the NoC, we used Booksim [34]. We employed Orion2 [133] to evaluate power consumption for interconnects and routers with 128-bit filts operating at 2 GHz. Orion2 estimates the NoC’s power consumption at the architecture level [133]. This library estimates the power consumption of the NoC router’s pipeline stages, taking into account the architecture- and circuit-level parameters of the NoC, such as fabrication technology, operating frequency, etc. Power modeling in Orion2 has been devised for 2D NoCs, therefore, estimation of interconnect power consumption is based on calculating the capacitance of the horizontal interconnects, and adding buffers to them. The method used in Orion2 is not sufficient for modeling the 3D NoC power, due to the existence of vertical interconnects and their distinct structure and size, compared to horizontal interconnects. Therefore, we added the extra technology information to Orion2, such as TSV size (Table 4.1), and formulae to calculate TSV capacitance [135]. Table 4.2 represents benchmarks that we used to evaluate our proposed algorithms.

Table 4.3 shows the number of matrix and extra variables per benchmarks for different mapping algorithms. The complexities of the proposed algorithms depend on the number of variables in the ILP model, as demonstrated in Sections 4.3.5, 4.4.5 and 4.5.1. PartiMap is the fastest mapping algorithm, since it has the smallest number of variables.
Table 4.3: Number of variables in the mapping algorithms

<table>
<thead>
<tr>
<th></th>
<th>OptHeraMap</th>
<th>PartiMap</th>
<th>ApproxiMap (k=8)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Name</td>
<td>Matrix var.</td>
<td>Extra var.</td>
</tr>
<tr>
<td>GSM Coder</td>
<td>3,392</td>
<td>331,776</td>
<td>53</td>
</tr>
<tr>
<td>GSM Decoder</td>
<td>1,224</td>
<td>62,208</td>
<td>34</td>
</tr>
<tr>
<td>MMS</td>
<td>1,044</td>
<td>29,808</td>
<td>29</td>
</tr>
<tr>
<td>MPEG4</td>
<td>144</td>
<td>3,312</td>
<td>12</td>
</tr>
<tr>
<td>VOPD</td>
<td>144</td>
<td>2,160</td>
<td>12</td>
</tr>
<tr>
<td>MWD</td>
<td>144</td>
<td>1,872</td>
<td>12</td>
</tr>
</tbody>
</table>

4.7 Evaluation

To answer the questions posed in Section 4.1, we examine the different constraints and objectives of PartiMap and ApproxiMap. We do not evaluate OptHeraMap due to its algorithmic complexity, which makes it unsuitable for regular-sized systems. We first evaluate the task power balancing effect on temperature. Then, we analyze the impact of explicit thermal constraints on chip temperature. Finally, we investigate the effects of different LP objectives, in terms of performance and temperature.

4.7.1 Effect of task power balancing on temperature

To answer the question of how important it is for chip temperature to consider power balancing during the mapping algorithm based on partitioning, we performed a series of simulations on two mapping algorithms based on PartiMap. The first mapping, PartiMap_CutPower (PM_CuP), is similar to PartiMap where both the cut minimization, objective and constraints 4 to 7 in Algorithm 2, and task power balancing, constraints 8 and 9 in Algorithm 2, are taken into account to partition the CTG. In the other algorithm, PartiMap_Cut (PM_Cu), we find the mapping based only on the cut minimization and omit the power balancing constraints.

Figures 4.6a, 4.6b and 4.6c show power delay product (PDP), peak and average temperatures, respectively. Applications that have a small number of tasks and utilize 3D design with a few number of layers have low peak and average temperature, e.g., MPEG, VOPD and MWD. Since both algorithms try to reduce the global task communication by minimizing the communication rate between partitions, they achieve the similar results in PDP for most of the applications. There are no major differences between PM_CuP’s and PM_Cu’s results in terms of peak and average temperatures. The maximum peak temperature difference between PM_CuP and PM_Cu is 6% for VOPD, where the PM_CuP performs better than PM_Cu. As the results show, considering the task power balancing in the partitioning-based mapping has no major effect on chip temperature. This is due to the fact that in the early stages of the mapping algorithm, little information is available about the final mapping. Furthermore, if the upper bound of the power balancing constraint is selected incorrectly, the LP model can become infeasible. Therefore, we do not recommend considering task power balancing for thermal
Chapter 4. Exploration of Thermal Constraints for 3D Network-on-Chip

Figure 4.6: Effect of task power balancing

(a) Power delay product

(b) Peak temperature

(c) Average temperature
management in mapping algorithm.

### 4.7.2 Effect of performance and explicit thermal constraints on temperature

It is important to know the effects of the explicit thermal constraint on performance and temperature. Also, we need to know whether including the implicit performance constraint in thermal mapping affects performance in a way comparable to that of the explicit performance constraint. In order to investigate these issues, we evaluate three mapping variants based on ApproxiMap with $k = 8$. In the first variant, referred to as ApproxiMap\_Communication (AM\_C), we change the ApproxiMap objective to minimize the zero load latency, as introduced in the fourth constraint in Algorithm 1. In this algorithm, we do not consider the temperature constraint. The second variant, referred to as ApproxiMap\_Communication\_Temperature (AM\_CT), is similar to ApproxiMap, in that the objective is to minimize the peak temperature and the performance, considering zero load latency. Finally, in the third variant, referred to as ApproxiMap\_Temperature (AM\_T), the performance constraint is omitted; therefore, it finds the mapping solely based on the objective of minimizing the maximum temperature. For the partitioning phase of these variants, we run the partitioning based on cut and task power balancing.

Figure 4.7 shows the simulation results for maximum and average temperatures and PDP. As explained in Section 4.7.1, considering task power balancing has no major effect on chip temperature. PM\_CuP uses communication and bandwidth constraints to localize traffic and minimize the amount of traffic between partitions. Therefore, we can consider that PM\_CuP focuses on performance improvement. We compare mapping results of PM\_CuP with those of ApproxiMap.

As shown in Figure 4.7, the performance-based algorithms, i.e., AM\_C and PM\_CuP, have better PDP than the temperature-based algorithms; i.e., AM\_T and AM\_CT. PDP is increased up to 32% for the MMS benchmark, which has the highest communication rate. However, the AM\_T and AM\_CT algorithms have the best values for peak and average temperature. They achieve a 24% decrease in peak temperature for the MMS benchmark. For the GSM\_C benchmark, which has the largest number of tasks, AM\_T and AM\_CT algorithms reduce the peak temperature up to 22%; however, they increase PDP by 26%. Furthermore, the results of AM\_CT do not show major differences compared to AM\_T. Altogether, the thermal constraint can affect the peak and average temperatures.

Since we use branch and bound methods to solve the ILP model, AM\_T and AM\_CT can find a mapping faster than AM\_C. In AM\_CT and AM\_T algorithms, we can cut the branches earlier compared to AM\_C, because in AM\_C we have to wait for most of the tasks to be mapped in order to conclude that it yields a bad mapping; however, in AM\_CT and AM\_T algorithms, branches are cut early based on the temperature cost. For example, AM\_C creates 1622 branches to find the best mapping for MMS benchmark, while AM\_T and AM\_CT create 119 and 335 branches, respectively; the LP solver must run once for each branch.
Figure 4.7: Effect of performance and explicit thermal constraints
In order to calculate the tile temperature, we consider the power consumed by each router, based on the rate of the data it transfers. To minimize the peak temperature, the thermal-aware algorithm tries to decrease the data transferred by routers to gain lower temperature. This decrease in data transfer is an implicit performance constraint. Furthermore, congestion constraint, i.e., constraint 3 in Algorithm 1, control the amount of traffic in each router. As the results show, this implicit performance constraint can play the same role on performance management as explicit performance constraints, such as zero load latency. Therefore, considering explicit performance constraints is not crucial in the thermal mapping algorithms, neither for achieving good performance, nor for a good temperature result. It is sufficient to consider the temperature constraint along with congestion control constraint. We also compare results of our mapping algorithms with those of a random mapping algorithm. Random mapping does not achieve suitable results, especially for MMS, GSM.D, and GSM.C. Using the thermal-aware mapping becomes crucial when the number of the layers in 3D IC or the rate of task communication increases.

4.7.3 Effect of different combinations for LP objective on performance and temperature

To find a suitable thermal-aware mapping algorithm, we need to choose the best objective for the LP model. The following objectives can be proposed for LP based thermal-aware mapping:

Minimize $H_1+H_2$

1) where \[
H_1 = \max\{\text{Temp}_p: p = 1 \cdots m, \text{for a system of } m \text{ cores}\}
\]

$H_2 = 0$

2) where \[
H_1 = \text{avg}\{\text{Temp}_p: p = 1 \cdots m, \text{for a system of } m \text{ cores}\}
\]

$H_2 = 0$

3) where \[
H_1 = \text{Comm. Cost}
\]

$H_2 = 0$

4) where \[
H_1 = \alpha \times \max\{\text{Temp}_p: p = 1 \cdots m, \text{for a system of } m \text{ cores}\}
\]

$H_2 = (1-\alpha) \times \text{Comm. Cost}$

We evaluate some of these objectives in previous sections, such as objective_1 with and without the performance constraint in Section 4.7.2, and objective_3 without the temperature constraint in Section 4.7.1.
Chapter 4. Exploration of Thermal Constraints for 3D Network-on-Chip

Figure 4.8: Effect of different LP objective combinations

(a) Power delay product

(b) Peak temperature

(c) Average temperature
In the previous section, we conclude that considering explicit performance constraints is not crucial in thermal mapping algorithms. It is sufficient to solely consider the temperature constraint. Therefore, objective \(_3\) with the temperature constraint and objective \(_4\) cannot provide better results than objective \(_1\) which we evaluate in previous section.

In order to compare objective \(_1\) and objective \(_2\), we change the objective function in \(\text{ApproxiMap}\) and simulate these mapping algorithms. In the first variant, referred to as \(\text{AM}_T\), the objective of \(\text{ApproxiMap}\) is to minimize the maximum temperature, as explained in Section 4.7.2. In the second variant, referred to as \(\text{ApproxiMap\_Average\_Temperature\_Average (AM\_ATA)}\), we change the objective of \(\text{ApproxiMap}\) to minimize the average temperature. Again, to solve the ILP model, we use the branch and bound algorithm. To cut the branches in the tree created in the branch and bound algorithm, we calculate the cost function based on the objective function. Finally, in the third variant, referred to as \(\text{ApproxiMap\_Average\_Temperature\_Maximum (AM\_ATM)}\), our objective is to minimize the average temperature. The cost function, which is used to cut the branches in the tree created in the branch and bound algorithm is different from the objective function: The cost function is based on comparing the maximum temperature instead of comparing the average temperature.

The simulation results for maximum and average temperatures and PDP of these variants are shown in Figure 4.8. As the results show, comparing \(\text{AM\_ATA}\) and \(\text{AM\_ATM}\), these variants have the same effect on the average and peak temperatures; however, \(\text{AM\_ATM}\) does not follow the same trend in terms of reducing PDP. Comparing \(\text{AM\_ATA}\) and \(\text{AM}_T\), \(\text{AM\_ATA}\) results in up to 4.6% better average temperature for the MMS benchmark, while the peak temperature and PDP are increased 0.6% and 3.4%, respectively. Finally, when the numbers of layers in a 3D IC, e.g., GSM\^C, or the communication rates among the tasks are increased, e.g., MMS, minimizing the average temperature achieves better results in terms of thermal management.

4.7.4 Discussion

Subsequent to our work, Hassanpour et al. \[56, 57\] proposed a thermal-aware migration algorithm using game theory. Their distributed thermal-aware migration algorithm tried to minimize the peak temperature while considering the overhead of migration on the application’s performance. They compared their results against our thermal-aware mapping algorithm, \(\text{ApproxiMap}\). They showed that thermal-aware migration can achieve peak temperature reduction compared with static thermal-aware mapping algorithm.

Although our proposed thermal-aware mapping algorithms are used to explore the effects of thermal constraints, they are orthogonal to the other thermal management techniques, e.g., DVFS, and the most of the recent thermal-aware mapping algorithms. Therefore, we can combine them to further improve the chip temperature. For example, Cheng et al. \[28\] proposed a thermal-aware mapping considering interconnect energy consumption. First, they started with an initial thermal-balanced solution using tasks’ power. Second, they utilized a greedy algorithm to minimize the energy consumption of interconnects. Finally, they used a simulated
annealing to optimize the result. We can use our ILP-based thermal-aware mapping algorithms instead of first two steps. In that case, the simulated annealing can converge faster due to improving the results of initial solution and energy-aware algorithm.

4.8 Concluding remarks

In this chapter, we proposed three thermal-aware mappings based on the LP model. We explored the effects of considering task power balancing early in the mapping, and the effects of performance and explicit thermal constraints on temperature and performance. Based on our evaluations, considering the task power balancing in the partitioning-based mapping has no major effect on the chip temperature. Furthermore, in a mapping based on temperature minimization, since we consider the average router power to calculate temperature, considering the explicit performance constraint has no major effect on PDP. Therefore, considering the temperature constraint alone, which yields a simpler LP model, is sufficient. Moreover, our proposed algorithm is capable of reducing the peak chip temperature by up to 24% and 22% for the benchmarks with the highest communication rate and largest number of tasks, respectively.
Chapter 5

CPath: Dynamically Predicting the Critical Path

In this chapter, we examine the solution that focuses on the inter-thread communication, as explained in Chapter 1. In Section 5.2, we elaborate on the motivation to this work. Then, in Section 5.3, we propose a critical path predictor. Section 5.4 presents our methodology. Finally, in Section 5.5, we discuss the evaluation of proposed critical path predictor.\textsuperscript{1}

5.1 Introduction

Multi-threaded applications allow programmers to extract the most performance out of modern chip multiprocessors. An application is divided into threads that operate in parallel rather than sequential manner leading to decreased runtime. However, performance gains achieved are far from ideal. One major reason for that is the dependency between threads through different synchronization primitives. Therefore, some threads cause others to wait and waste precious core cycles. Hence, identifying such critical dependencies can be crucial in maximizing the utilization of the different cores. In this chapter, we propose a new scheme, CPath, to predict the critical path of a multi-threaded application running on a shared-memory chip-multiprocessor (CMP) architecture on the fly. Our online propagation-based critical path predictor uses a thread’s arrival time at a barrier, last level cache (LLC) statistics at barriers and critical section dependencies of multi-threaded applications. We formulate the thread criticality prediction as a ML classification problem using a novel sort-based normalization for ML features to reduce the computation and hardware cost and provide a scalable predictor. We also use lock contention information to propagate the criticality of these critical threads to other threads that they depend on. Using criticality propagation, we improve prediction for benchmarks that heavily

\textsuperscript{1}Section 5.3.1, which presents criticality propagation, has been developed by Shehab Elsayed and I. We both contributed equally to this part. Some parts of evaluation section, Sections 5.5.3 and 5.5.4, have been developed by Shehab Elsayed, which are presented in this thesis to provide a coherent work. Sections 5.5.5 and 5.5.6 have been codeveloped.
depend on locks. We improve the critical path prediction by 111% and 191% over two recent prediction techniques for a 16-core system.

5.2 Motivation

Previous work that addresses criticality in multi-threaded applications can be divided into two classes. One class [15, 22, 38, 64, 83] tries to predict which thread is likely to take more time based on statistics collected online such as cache miss rate [15], parallel loop progress [22] or the amount of work being done in parallel [38]. However, these metrics do not directly convey the dependency between the threads (dictated through software synchronization). Moreover, the accuracy of their predictions greatly depends on the interval at which they measure the metrics of interest.

The other class tries to identify the synchronization bottlenecks that prevent threads from making forward progress towards the completion of the application [65, 39]. Schemes in this class generally base their criticality predictions on the total thread waiting time at each bottleneck. The longer the wait, the more likely the corresponding bottleneck is predicted to be critical. However, inspecting the bottlenecks individually could lead to wrong criticality estimations [27]. Instead, bottlenecks must be considered in light of their contribution to the progress of the whole application. Figure 5.1 shows an example where the critical sections guarded by Lock-1 and Lock-4 are on the critical path of the application although the ones guarded by Lock-2 and Lock-3 are the ones responsible for the longest waiting time across threads. Chen and Stenstrom [27] use this observation to develop an algorithm to determine critical locks offline, however it cannot be used to predict them during runtime.

Figure 5.2 shows the accuracy (how many of the total predictions were correct) and coverage (how much of the actual critical path was predicted correctly) product, ACP, for two critical path predictors based on Bottleneck Identification and Scheduling (BIS) [65] and Criticality Stacks (CS) [38]. With a maximum achievable ACP of one, there is clearly room for
improvement in critical path prediction for multi-threaded applications.

Therefore, we build on the observation that waiting times could lead to misleading conclusions about the critical path [27], to develop a technique that predicts which parts of each thread lie on the critical path of the application. Unlike previous techniques, we do not predict based on the amount of waiting time at synchronization events. We make use of collected statistics, barrier arrival time and LLC statistics, at barriers to predict a thread’s criticality. We use a machine learning algorithm to effectively find patterns in the collected statistics for criticality prediction. Furthermore, the critical thread propagates its criticality backward to threads blocking it to reflect the effect of inter-thread dependencies on the critical path of the application.

**Why focus on the LLC?** In each cycle, one thread, referred to as the critical thread, is participating in the critical path. Identifying the critical thread can be used as a coarse-grained prediction for the application’s critical path. When a critical thread predictor is designed: 1) the metric used for prediction should be chosen carefully by considering the application’s characteristics, 2) the algorithm employed for prediction using that metric should be good enough to capture the behavioural patterns of the application.

Prior work [15] has demonstrated that L1 and L2 cache miss rates can be an effective indicator of thread criticality. We expand upon this observation to consider a wider range of LLC statistics to aid in our thread criticality predictions at barriers in data parallel applications. In data parallel applications, different parts of the data are assigned to each thread. Thus, the main difference between threads’ work is how they access their data from memory. Even though threads operate on different parts of the data, they still need to compete for shared resources, such as the LLC and memory. Threads communicate and affect each other through memory. LLCs\(^2\) hold information about shared data between cores that is not available anywhere else on chip (L1s or memory controllers). For example, the LLC can show if data brought on-chip by one thread is used by other threads and is likely to be critical, or that a thread with a large

\(^2\)We assume the directory is co-located with the LLC.
LLC miss rate may fall behind in execution. Therefore, we propose to model thread criticality as a function of a thread’s LLC statistics. While we focus on data parallel applications in this thesis, our approach extends to any parallel shared memory applications that feature barriers and locks as their synchronization mechanisms.

## 5.3 Critical Path Algorithm

In this section, we present our critical path algorithm, *CPath*, which uses collected LLC statistics and barrier arrival times, and threads’ dependencies to find the application’s critical path on-the-fly. For that purpose, *CPath* uses the following synchronization events:

- **Barriers**: Due to their small number and repeated use, barriers can be a simple and inexpensive candidate to capture the program behavior and its phase changes [36]. Therefore, at each barrier, *CPath* collects the required statistics and predicts thread criticality.

- **Locks**: *CPath* detects the dependency between threads based on lock contention. Then, it employs this information to propagate the criticality between threads.

In the next sections, we show how the criticality propagation mechanism works. Then, we show how we formulate and use the collected statistics as a machine learning classification problem. We then discuss the hardware overhead associated with *CPath*. 

---

*Figure 5.3: CPath: High-level overview*
5.3.1 Criticality Propagation Algorithm

Figure 5.3 shows an overview of our proposed approach to predict the criticality at the block level. We define a block as a sequence of instructions between two consecutive synchronization events in one thread. We employ barriers to predict the critical threads. Furthermore, we use lock dependency information to pass thread criticality to other threads responsible for delaying the critical thread. If a non-critical thread is holding a lock required by the critical thread, the non-critical thread inherits the criticality of the critical thread, i.e., the non-critical thread becomes critical. Criticality propagation aims at finding a more direct relationship between a critical section in a certain thread and the progress of the whole application instead of depending on waiting times and the amount of lock contention as done previously [65, 39].

Figure 5.3 depicts three iterations of a parallel loop with a barrier and four locks in an application with four threads. At the barrier, a thread is predicted to be critical or not based on the collected statistics. At the end of the first iteration, Thread-2 is predicted to be critical based on the collected statistics. Thus, CPath marks all of the blocks in Thread-2 to be critical. Therefore, when Thread-3 reaches Lock-4 and stalls Thread-2 in the second iteration, CPath propagates the criticality backwards to Thread-3 and marks its corresponding blocks to be critical. The criticality propagations are represented as batches of arrows with common origin in Figure 5.3. In the third iteration, since Thread-3 was marked to be critical, CPath propagates the criticality to Thread-1 after detecting contention at Lock-1 between these two threads. Therefore, at each iteration, we go one level deeper in the thread dependency chain. Although this delays the criticality propagation to some blocks, it prevents making premature propagation. Note that CPath does not stall the application to predict the critical path.

The main goal of criticality prediction is to improve an application’s power or performance. Therefore, a criticality-aware optimization such as DVFS is required. These optimizations will change the thread’s criticality. Thus, we employ a criticality counter, to avoid ping ponging between predicting a thread to be critical or not. A block’s criticality prediction depends on its criticality counter value relative to a preset threshold. A block’s counter is incremented in two cases:

1. its corresponding thread is predicted to be critical at the barrier, or

2. it receives propagated criticality from a critical block.

On the other hand, the counter is decremented only if the corresponding thread is predicted to be non-critical at the barrier. The values of the threshold, increment and decrement affect the result of the critical path prediction algorithm. For example, if the value of increment is too large along with small values for the threshold and decrement, the algorithm will mark all of the blocks in the application to be critical. We investigate the impact of these values in Section 5.5.4.
Chapter 5. CPath: Dynamically Predicting the Critical Path

5.3.2 Critical Thread Predictor: A Machine Learning Approach

We use machine learning to effectively find patterns in the LLC and barrier arrival time statistics for criticality prediction. Machine learning can capture complex, non-obvious patterns. To predict criticality, we formulate the cache statistics and the barrier arrival times as a machine learning problem. We employ a simple supervised machine learning algorithm, k-neighbors classification (KNN), to extract useful statistics, e.g., cache line or barrier arrival time statistics, and design a critical thread predictor. Neighbor-based classification computes a classification based on the majority vote of the nearest neighbors of that point [108]. We provide a machine learning model for each application. We train the KNN on a small, representative input set to determine the statistic that correlates to thread criticality. This training is done offline. Then, we use that statistic on a larger input set to identify the critical threads on-the-fly.

Our approach relies on LLC statistics that indicate the following: a) why a thread accesses this shared resource, e.g., reading data or instructions, b) how many times a thread accesses the LLC or memory and c) whether they are successful, in terms of finding the required data without competing for other shared resources, or not. Hence, at each barrier, we collect the following statistics: 1) requestor: the number of times that a thread requests a cache line from memory; 2) provider: the number of times cache lines brought into the LLC by a thread are used in other threads; 3) consumer: how many times a thread uses cache lines that exist in the LLC; 4) victim: the number of cache lines brought into the LLC by a thread that are chosen to be replaced; 5) misses: the total L1 and LLC (L2) misses of a thread. Each of these statistics is divided into: 1) instructions, 2) reads, and 3) writes. Each thread at each barrier is a data point for the KNN algorithm and the features for each data point are the LLC and barrier arrival time statistics as shown in Table 5.1. Each cache statistic subset can have a different range of values, especially when the application input data set changes. Therefore, we propose a sort-based normalization. Hence, our KNN-based predictor can be scalably applied to different application input sets. Table 5.1 shows an example of our sort based normalization.

<table>
<thead>
<tr>
<th>Data points</th>
<th>Features : Requestor</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Data points</th>
<th>Normalized Features : Requestor</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

Table 5.1: Sample of data inputs, features and sort based normalization
on the requestor statistics. After collecting the statistics at the barrier, we sort each statistics subset. Then, we change the original values of the statistics to their IDs in the sorted list. After applying the normalization, each statistic has a value between 0 and \#Threads − 1. This range reduction reduces the computation cost and the memory required for storing the values. To avoid the curse of high dimensionality,\(^3\) we run KNN on each set of statistics separately. Therefore, we have several criticality predictors based on each set of statistics. We compare these predictors and provide additional discussion in Section 5.5.1.

5.3.3 Implementation Details

In this section, we walk through the steps associated with our criticality prediction and the relevant hardware structures. We also detail the associated hardware overheads.

Propagation. CPath requires per-core tables (\textit{barrierCrit} and \textit{blockCrit} tables) for preserving the criticality counters for each block, as shown in Figure 5.4a. The \textit{barrierCrit} table has an entry per barrier. Each entry contains a counter (barrier-epoch-counter) and a pointer to \textit{blockCrit} tables. The barrier-epoch-counter keeps track of the criticality of code between two barriers. \textit{BlockCrit} tables contain criticality counters of the blocks for the corresponding barrier. Each entry in \textit{blockCrit} tables also has a tag to preserve the lockID.

Next we explain how the tables are accessed and used. A block is addressed by the barrierID and lockID, since we classify code between two barriers into blocks based on locks. We use

\(^3\)A machine learning algorithm may perform poorly when the number of features increases.
a simple hash function, \( h(\text{barrierID}) \), to access the barrierCrit table \( \footnote{2} \). Then, we use a simple hash function, \( g(\text{lockID}) \), and pointer from barrierCrit to access an entry in blockCrit table \( \footnote{3} \). The reason for this design choice is that a lock can appear between different barriers and show different behaviour depending which barrier it is associated with, e.g., locks in \textit{fmm} benchmark from SPLASH-2 \( \footnote{138} \). A thread can find its block criticality by comparing the block counter in its blockCrit table to the predefined threshold \( \footnote{5} \). If there is a tag mismatch \( \footnote{4} \), the block uses the barrier-epoch-counter to find its criticality \( \footnote{5} \).

\textbf{ML.} Our ML-based criticality predictor requires a table (\textit{origin table}) for collecting the LLC statistics as shown in Figure 5.4b. The origin table has an entry per cache line or a subset of cache lines to reduce the overhead. Each entry preserves the ID of the thread that brought that line into the cache and uses it as a pointer to the other tables (\textit{stat tables}). The stat table preserves the values of the designated LLC statistics. We envision a reconfigurable interface for the stat tables which chooses the metrics used to designate which threads are critical. A counter in the stat table is updated when a thread requests data at LLC. For instance, to collect the provider statistics for \textit{Thread-X}, when \textit{Thread-Y} accesses a cache line brought to the LLC by \textit{Thread-X} \( \footnote{A} \), we access the origin table to get the pointer to the stat table \( \footnote{B} \). Then, based on \textit{Thread-Y}’s operation on the LLC (read/store), the corresponding counter in stat table is incremented \( \footnote{C} \).

As we are using a sort-based normalization, we need to sort the values from the stat table before using them to predict the thread criticality. At the end of each barrier, all counters in the stat table are sorted in hardware separately and reset \( \footnote{D} \). KNN uses the sorted values as the ML features to predict thread criticality \( \footnote{E} \).

Our KNN-based predictor requires a simple look-up table for saving the training data to use for majority vote \( \footnote{F} \). This table has an entry for each sampling data point. By using a sort-based normalization, the number of bits for saving the feature values of a data point is reduced. Since the training is offline, we are able to choose the best data structure for storage; this enables efficient online nearest neighbor search. We use a KD-tree as the data structure for KNN’s data sample table, which only needs \( O(\log_2 M) \) distance computations to determine the nearest neighbor of a query in \( M \) samples \( \footnote{49} \)[\footnote{108}]. A KD-tree employs two pointers per data point \( \footnote{11} \). The KD tree is built offline.

\textbf{Altogether.} After predicting the thread criticality, a message is sent to that thread to update the corresponding criticality counters \( \footnote{G} \). For example, assume an application that has two barriers, \textit{Barrier-X} followed by \textit{Barrier-Y}. After executing the code following \textit{Barrier-X} (i.e., at \textit{Barrier-Y}) \textit{CPath} either uses the collected statistics to update the criticality counters of the blocks following \textit{Barrier-X} (\_rept) or those following \textit{Barrier-Y} (\_prev). We add a small controller \( \footnote{6} \) to update the corresponding barrier-epoch-counter and blockCrit table. This controller uses a few bits to save the current barrierID, a bit for predictor type (\_rept or \_prev) and a bit for the criticality prediction. When a critical block is waiting to acquire a lock, it sends a signal to the blocking thread to increment the counters in its blockCrit table for the
blocks visited before that lock. A core requires a small amount of storage to preserve the earlier lockIDs in the current barrier, e.g., 0.39 Kbytes for 290\(^4\) different lockIDs.

**Hardware Cost.** The barrierCrit table with \(B\) entries has \(B\) blockCrit tables. Thus, the barrierCrit table with \(K\) bits per counter has \(B \times (K + \log_2 B)\) bits. Each blockCrit table with \(L\) block-criticality counters, \(C\) bits per counter and \(T\) bits per tag has \(L \times (C + T)\) bits. Therefore, the total size of these tables in an \(N\)-core system is \(N \times B \times (K + \log_2 B + L \times (C + T))\) bits.

For example, for a 16-core system, we use 20 (\(B\)) entries for the barrierCrit table. Therefore, the barrierCrit and blockCrit tables with 6 (\(K,C\)) bits per counter, 1 (\(T\)) bits per tag and 136 (\(L\)) criticality counters have 28 bytes and 2.35 Kbytes per core, respectively. Updating the blockCrit table is not on the critical path of our prediction. Since the table size is not large; the power consumed to make these updates is negligible. We used CACTI [101] 32nm to calculate the power consumption. In our design, the total dynamic read energy per access of a blockCrit table is 0.88 pJ. The total leakage power of blockCrit tables is \(20 \times 0.45 = 9\) mW.

The origin table and stat table in an \(N\)-core system with a subset of \(C\) cache lines, \(S\) LLC statistics counters and \(K\) bits for each counter require \(C \times \log_2 N\) and \(S \times K \times N\) bits, respectively. For example, in our design, for a 16-core system, the origin table and stat table with 8K (\(C\)) cache lines (512 Kbytes L2 cache), 9 (\(S\)) LLC statistics counters and 16 bits (\(K\)) for each counter have 24 Kbytes and 0.28 Kbytes, respectively.

In an \(N\)-core system, each feature in a data point requires \(\log_2 N\) bits after the normalization. The KD-tree table size for \(M\) training data points with \(D\) features is \(M \times (D \times \log_2 N + 1 + 2 \times \log_2 M)\) bits. The KNN distance computation per query is \(D \times \log_2 M \log_2 N\)-bit integer operations and \(\log_2 M \times ([\log_2 D] + 2 \times \log_2 N + 1)\)-bit comparisons. The KNN-predictor has a \(\log_2 N\)-bit integer adder and multiplier per feature, one \(([\log_2 D] + 2 \times \log_2 N + 1)\)-bit integer adder and comparator. By assuming one cycle for adder, comparator and accessing the table and \(O(\log_2^2(\log_2 N))\) cycles for multiplier, KNN-predictor latency is \(O(\log_2 M \times \log_2^2(\log_2 N))\) cycles per query. In the following example, we consider the maximum number of data points, 10976 (\(M\)) (based on ocean\_cp). For a 16-core system, the largest table size with 9 (\(D\)) features is 87 Kbytes \(\dagger\). For a large number of data points, we can apply data reduction to have a smaller training set [7, 137]. The KNN distance computation per query is 126 4-bit integer operations and 14 13-bit comparisons. The KNN-predictor has a 4-bit integer adder and multiplier per feature, one 13-bit integer adder and comparator. Its latency is 56 cycles per query. A smaller number of data points leads to lower latency, e.g., 28 cycles for 1u\_cb. The origin table is clock gated by the reconfigurable interface, if a statistic, such as number of LLC misses which does not need the originator information of cache line, is used. Furthermore, when a small number of features per query is considered, the unused portions of sample table is switched into a power saving mode, e.g., drowsy mode [46].

\(\dagger\)See Section 5.5.3
5.4 Methodology

In this section, we explain how we instrumented our benchmarks to obtain the necessary information about barrier and locks. We also explain our testing framework comprised of the offline critical path calculation (oracle) and the two schemes we compare against. Finally, we explain our evaluation metric, the prediction $ACP$, and how it combines both accuracy and coverage.

5.4.1 Software Support

Since there is no direct way to detect and differentiate between synchronization events (specifically barriers and locks), software instrumentation is used to mark such events [39, 65]. We follow a similar approach where we mark the different events as shown in Figure 5.5.

5.4.2 Testing Framework

To test CPath, we collected traces for the synchronization events of interest for applications from SPLASH-2\textsuperscript{5} [138]. We chose ones that exhibit an interesting range of synchronization behaviour. We are mainly interested in the number of barriers and locks in each benchmark as well as how many times a certain barrier or lock is repeated. Table 5.3 shows some of the synchronization characteristics of the chosen benchmarks. We found a trace-based approach

---

\textsuperscript{5}We used the SPLASH-2X from PARSEC-3.
Table 5.3: Synchronization characteristics of the used benchmarks (16-threads, simmedium). \textit{Lock Contention} is the ratio of the number of lock acquire failures to the total number of lock acquires to be sufficient since we do not perturb the critical path when testing our algorithm. We use gem5 [16], a full-system cycle accurate simulator, to collect traces for 16 threads running on 16 cores. Table 5.2 shows the full system configuration. We use a python-based machine learning software package, scikit-learn [108], to implement our ML model. Using the collected traces, we train our ML-based predictor on the simsmall input set. Then, we measured the accuracy of the predictors on the simsmall and simmedium input sets.

We run \textit{CPath} on the collected traces and compare its performance to an offline critical path identification technique described in Section 5.4.3. We also compare \textit{CPath} with two previous block criticality prediction techniques:

- \textbf{Bottleneck Identification and Scheduling (BIS)} [65]: A bottleneck is defined as a part of the program that can be responsible for making other threads wait. \textit{BIS} keeps track of the total waiting times across all threads for the different bottlenecks. Any thread that starts executing a bottleneck with a total waiting time larger than a preset threshold is deemed to be critical.

- \textbf{Criticality Stacks (CS)} [38]: A thread’s criticality is based on the amount of time it was doing useful work as well as the number of parallel threads running during that time. Each thread maintains a counter that is incremented whenever a thread changes its state from running to idle or vice-versa. These counters are checked at regular intervals and the thread with the largest counter is deemed to be critical.

Neither of these previous techniques explicitly evaluated the accuracy of their criticality predictions. Instead, their main focus was on deciding which blocks to accelerate either by executing on a larger core as in \textit{BIS} or frequency scaling as in \textit{CS}. Therefore, each of these schemes further differentiated between the different code blocks based on properties related to their acceleration method. In our implementation of the two schemes, we ignored such acceleration-specific properties, however, the core criticality prediction algorithm is unchanged.\footnote{For BIS: Threshold = 1024, TWC halving period = 100Kcycles. For CS: Time Slice = 10msec, $\alpha = 1.2$ and $\beta = 0.8$.}
5.4.3 Oracle

The offline algorithm (oracle) [27] consists of two steps. First, the algorithm creates a dependency tree based on the relationship between the different blocks where a block starts and ends with synchronization events. A block starting with a barrier is dependent on the last block to reach that barrier. A block starting with a lock is dependent on either the current lock owner (if exists) or the preceding block in the same thread. Second, the algorithm traverses the dependency tree backwards to find the chain of nodes leading to the last block in the tree. This chain is the critical path we are looking for.

5.4.4 Evaluation Metrics

An effective prediction scheme is expected to predict most of the blocks on the offline critical path (coverage) while keeping mis-predictions to the minimum (accuracy). Therefore, we evaluate the performance of the different prediction schemes using two metrics: Accuracy and Coverage as shown in Equations 5.1 and 5.2. Accuracy measures the number of correctly predicted blocks relative to the total number of blocks (predictions). This favors algorithms with a low number of mis-predictions. Coverage, on the other hand, measures the number of correctly predicted critical blocks relative to the actual number of critical blocks as calculated by the offline algorithm. This favors algorithms with a large number of correct critical predictions regardless of the number of mis-predictions. We further combined both metrics into the Accuracy Coverage Product (ACP).

\[
\text{Accuracy} = \frac{\# \text{Correct Predictions}}{\# \text{Blocks in Application}} \quad (5.1)
\]

\[
\text{Coverage} = \frac{\# \text{Correct Critical Block Predictions}}{\# \text{Blocks of Oracle}} \quad (5.2)
\]

\[
ACP = \text{Accuracy} \times \text{Coverage} \quad (5.3)
\]

In a multi-threaded application, the number of non-critical blocks is much larger than the number of critical ones, especially as the number of threads increases. Generally, at a given point in time only one thread (i.e., block) limits the application’s performance while others run in parallel. At one extreme, an algorithm predicting all blocks to be non-critical can still have very high accuracy yet it does not provide any useful information regarding the criticality of the different blocks. Therefore, we include both coverage and accuracy in the ACP of the prediction algorithm.

To further show the importance of having these metrics, consider three algorithms A, B and C. We use them to predict which blocks are critical out of a total of 400 blocks. The offline algorithm determines only 100 blocks are on the critical path. Table 5.4 summarizes the results of the three algorithms. We argue that B is the best among all three algorithms since it was able to achieve a balance between accuracy and coverage (highest ACP). Accuracy
and coverage focus on two different aspects of prediction: quality vs. quantity. Favouring one on account of the other leads to either under-/over-prediction of critical blocks. By ACP, we capture both aspects for easier comparison between schemes. Criticality predictions will be used to improve performance for critical blocks. Therefore, we must ensure we identify enough critical blocks to achieve significant overall improvement. At the same time, we have to minimize the chance of accelerating the wrong blocks to prevent any negative interference between critical and non-critical blocks.

### 5.5 Evaluation

We first evaluate ML-based critical thread predictors. Then, we investigate the effect of limited resources, the sensitivity of \textit{CPath} to criticality counter increment value and the effect of criticality propagation on its ACP. We also compare \textit{CPath} with two other prediction techniques.
5.5.1 ML-based predictors

As mentioned in Section 5.3.2, we train our KNN model with the simsmall input set for each application using barrier arrival times and different LLC statistics. Then, we use the KNN model to predict the criticality of threads for the simsmall and simmedium input sets without any extra training or adjusting. We set the number of neighbors used for majority vote in KNN equal to 3. Figure 5.6 shows the product of critical threads (CT) and non-critical threads (NCT) accuracies (how many of the CT or NCT predictions were correct) of the proposed critical thread predictors based on different LLC and barrier arrival time statistics. In Figure 5.6, the sets with the suffix `rept (_.prev)` show that the predictor uses the statistics on the latest repetition (previous barrier) of corresponding barrier to predict its criticality. Predictors without the suffix use the current statistics at each barrier.

Although we evaluate several LLC statistics, e.g., provider, requestor, consumer victim and miss, for readability we only show the statistics that provide the best results among the others for these benchmarks in Figure 5.6.

To select the predictor, we consider its product of CT and NCT accuracies using simsmall input set. An efficient predictor should be reasonably accurate in both CT and NCT. If a predictor has a high accuracy in CT but low accuracy in NCT, it means that it predicts most of the threads to be critical. If predictor has low accuracy in CT or NCT, the statistics used for the prediction cannot capture the parallel application’s behavior. Hence, we choose `time_prev`, a predictor based on barrier arrival time statistics, for `lu_cb`, `lu_ncb`, `ocean_cp` and `radix`, `provider_prev` for `ocean_ncp` and `volrend`, `miss_prev` for `barnes` and `miss_rept` for `fmm`. Our mechanism can easily use both LLC and barrier arrival time statistics, based on their efficiency determined from offline analysis on a small input set. As mentioned in Section 5.3.3, predictors using time do not require origin and stats tables. These extra hardware can be clock gated.

5.5.2 Use Case: Power Saving

In this section, we employ dynamic voltage and frequency scaling (DVFS) along with our thread criticality prediction to reduce the system’s power consumption. For that purpose, we train our ML-based predictor to predict the most non-critical threads, the top 25% fastest threads, at the end of each barrier. Then, we utilize DVFS to lower the frequency of these threads. Furthermore, we compare our power-saving results with CS. For that purpose, we use CS to predict the top 25% non-critical threads. We use 10 ms for each time slot in CS.\(^7\) Figure 5.7a shows the power improvements for a 16-core system. We use 2 GHz as the high frequency with 1.15 V and 1.67 GHz as low frequency with 1.05 V for DVFS. We utilize McPAT [90] to calculate power for 32 nm. These power saving algorithms are conservative, since they only reduce the frequency of a few non-critical threads, the top 25% fastest threads.

Our power saving algorithm achieves up to 8.4% power saving. As shown in Figure 5.7b,\(^7\) This value is suggested for CS [38].
the effect on performance of the power saving algorithms using both our predictor and CS is negligible. In all benchmarks, our ML-based predictor achieves better results in term of power compared to CS. Our scheme can save 45% more power than CS, since CS underpredicts the non-critical threads and misses power saving opportunities. For example, in barnes, cores use low frequency only in 8% of the time when using CS, while our ML predictor results in low frequency usage 19% of the time.

5.5.3 Effect of Hardware Limitations

As explained in Section 5.3.3, four main factors control the hardware overhead of CPath:

1. Number of entries in Barrier Criticality Table,

2. Number of entries in Block Criticality Table,

3. Size of criticality counter,

4. Length of Inter-barrier Block History.
Since the number of barriers is usually not very large (maximum of 20 in the studied benchmarks) we chose to design the Barrier Criticality Table to accommodate all of them.\footnote{If the number of barriers exceeds the Barrier Table Size, newer barriers will overwrite older ones.} However, the number of critical sections exhibits more variation and larger range compared to the number of barriers. Hence, accommodating all locks without any aliasing in the Block Criticality Table introduces a lot of overhead.

In a case with unlimited resources, the size of the Block Criticality Tables mainly depends on the maximum number of critical sections between two successive barriers. Based on Table 5.3, \texttt{fmm} has the largest number of critical sections followed by \texttt{barnes} with more than 10× difference between them. Therefore we based our design choices on \texttt{barnes} and chose a table size of 136 entries to limit the hardware overhead. However, surprisingly \texttt{CPath} provided better ACP of 0.14 for \texttt{fmm} compared to the unlimited case ACP of 0.05. \texttt{fmm} has a very large number of locks which are used repeatedly between two barriers. Hence, the corresponding blocks appear multiple times in the Inter-barrier Block History and their criticality counters might be incremented multiple times during criticality propagation. This eventually leads to a very large criticality counter making it harder for the block to switch its criticality prediction which results in a high rate of mis-prediction if these blocks turn out to be non-critical in the future. However, with a small table size newer blocks will replace the older ones resetting their criticality counters and accordingly switch their prediction to non-critical much faster.

The size of the criticality counter determines how easily a block can change its criticality prediction. On the one hand, if the counter size was unlimited, a block would require an equal number of increments and decrements to switch its prediction from critical to non-critical (assuming the increment and decrement are of the same value). This leads to many blocks being mispredicted as critical. On the other hand, a very small counter size makes it very easy for a block to switch from critical to non-critical prediction states. Consequently, many blocks can be mispredicted as non-critical. Therefore, we tried various maximum counter values ranging from 3 to 255 and we found that 63 provided the best balance between the two extremes.

The Inter-barrier Block History length affects the criticality propagation. The longer the history, the more blocks will receive propagated criticality. Ideally the Inter-barrier Block History would be able to accommodate the maximum number of critical sections that can exist between any two consecutive barriers for a given benchmark. However, the propagation is also controlled by the size of the Block Criticality Table. The longer the history compared to the size of the table, the more the chance of updating a block that has already been replaced. We experimented with setting the history length to 5000, 2810 and 290 to accommodate the three benchmarks with the largest number of critical sections between consecutive barriers; \texttt{fmm}, \texttt{barnes} and \texttt{volrend} respectively. Based on our experiments a history length of 290 gave the best results. This is mainly due to the relatively small difference between the number of entries in the Block Criticality Table and the Inter-barrier Block History. It should be noted that there could be duplicate blocks in the block history (the same lock might be repeated multiple times
5.5.4 Sensitivity to Criticality Counter Increment Value

Among the main factors affecting the accuracy and coverage of CPath are the values of the criticality threshold and the increment and decrement values for the criticality counters. In order to simplify the sensitivity analysis, we study the effect of changing each of these parameters individually. In other words, we only change one parameter at a time and keep the value that gave the best results fixed for the rest of the analysis.

Figure 5.8 shows the performance of CPath averaged across the studied benchmarks as the increment, decrement and threshold values increases from one to eight. In general, as the increment value increases, the chance of a block being predicted as critical increases. This means that coverage is expected to increase (recall from Equation 5.2 that coverage is measured relative to the total number of blocks on the offline critical path which is fixed). However, accuracy decreases since it becomes harder for blocks to be demoted from being critical to being non-critical. Therefore, the number of blocks in the online critical path increases but in reality not all of them are necessarily critical. A similar argument applies to the effect of the decrement and threshold values. A large value of either means it is harder to predict a block to be critical and vice versa. We conclude that values of (2, 2, 1) for the increment, decrement and threshold respectively achieve a good balance between accuracy and coverage leading to an average ACP of 0.32 across all considered benchmarks. The main difference between setting the increment and decrement to (2,2) compared to (1,1) with a threshold of one is that the criticality counters can reach the threshold faster.

5.5.5 Effect of Criticality Propagation

In this section, we study the effect of the criticality propagation mechanism on our predictor’s ACP. In order to do so, we compare the original CPath to two variations:
Figure 5.9: Comparison between the accuracy, coverage and ACP for the studied schemes (16-threads, simmedium).

1. **CPath\textsubscript{noProp}:** where we turn off the criticality propagation (criticality counters are only updated upon reaching a barrier);

2. **CPath\textsubscript{dist}:** where the increment value decreases as the distance the criticality propagates increases.

Figure 5.9 shows the accuracy, coverage and ACP for three variations of CPath (rightmost three bars for each benchmark). The importance of criticality propagation can be seen by the difference in coverage between CPath\textsubscript{noProp} and the original CPath especially for barnes, fmm and volrend. Hence, the overall ACP of CPath\textsubscript{noProp} is 15.5% less than CPath on average.

CPath\textsubscript{dist} is based on the intuition that earlier blocks could have less deterministic impact on the progress of a thread up to a certain block since these blocks could actually be dependent on blocks from other threads. One example for that is the dependency between Thread-3 and Thread-1 on Block-(n+1) in Figure 5.3. We chose to model this by decreasing increment values to earlier blocks instead of exactly following the dependency chain to simplify the criticality propagation mechanism. As Figure 5.9 shows, CPath\textsubscript{dist} reaches a middle ground between both CPath and CPath\textsubscript{noProp}. However, this does not result in improved ACP.

The largest differences between the three variations of CPath appear in barnes, fmm and volrend. One of the main reasons for this is the large number of locks in between barriers. As shown in Table 5.3, these three benchmarks have the largest number of lock acquires (critical sections) and the smallest number of barriers among the benchmarks studied. Therefore, criticality propagation affects a larger number of blocks than any of the other benchmarks. On the other hand, the three variations perform exactly the same for lu\_cb, lu\_ncb and radix. As shown in Table 5.3, these benchmarks do not have any locks. Consequently, criticality propagation does not play a role in the critical path predictions for these three benchmarks.
5.5.6 Comparison to Previous Work

Figure 5.9 shows the overall ACP comparison between CPath, BIS and CS (left-most three bars for each benchmark). It shows that CPath achieves better or at least comparable ACP compared to the other algorithms for most of the benchmarks.

One issue with BIS is that its block criticality predictions are shared across all threads. In BIS, a table keeps track of the total waiting time of all threads at a certain synchronization event. Then, whenever any thread reaches the same event again, it checks the waiting time and makes the criticality prediction accordingly. This leads to a very high rate of mis-predictions (low accuracy) yet high coverage. On the other hand, CPath keeps track of block criticality per thread. Therefore, the same block can be critical for one thread but not the other. This explains why BIS performs very close to CPath for barnes. As shown in Table 5.3, locks in barnes are used repeatedly while barriers are not repeated as much. In CPath, we depend on barriers to predict the critical threads which then pass their criticality to the locks before that barrier, therefore barnes does not offer a large opportunity to create enough history for the barriers and their corresponding locks. On the other hand, BIS has the chance to create its needed history and predict accordingly. Eventually, this leads to its large coverage, but at the same time, low accuracy. However, CPath still manages to achieve 7.7% better ACP than BIS for that specific benchmark.

Both CPath and BIS make their predictions based on past behaviour of the threads at the same synchronization event. On the other hand, CS predicts thread criticality at regular time intervals based on runtime statistics (number of active threads and the duration of activity) regardless of whether a certain event was repeated before or not. However, it does not directly take into consideration the dependency between the threads. This gives CS an advantage for benchmarks with a low number of block repetitions such as fmm which has lowest repetitions for both barriers and locks as shown in Table 5.3.

5.5.7 Scalability

One of the main challenges facing critical thread prediction is how it scales as the number of threads increase. Having more threads implies that there is a higher chance of mis-predicting threads to be critical since the pool from which we need to pick a critical thread increases. As explained in the previous section, BIS shares its block criticality predictions among all threads. Therefore, as the number of threads increases, more threads will be predicted to be critical but in fact are not. Therefore, CPath’s ACP benefit over BIS increases from 10% to 111% when going from 4 cores to 16 cores. At the same time CPath’s benefit over CS increases from 78% to 191% mainly due to CS’s low coverage.

Summary: CPath balances accuracy and coverage. We achieve relatively high accuracy by using a machine learning based classification algorithm to choose which statistics to use in our predictions. At the same time, our criticality propagation mechanism increases our prediction
coverage. As a result, we are able to achieve higher ACP than the other schemes we compared against.

5.6 Concluding remarks

Multi-core systems allow applications to achieve more performance by running multiple threads simultaneously. However, the performance bottleneck in multi-threaded applications is their sequential parts, which is a side effect of using synchronization. We propose a new online critical path predictor, CPath, for multi-threaded applications on-the-fly. CPath employs collected statistics at barriers and lock contention information to propagate the criticality from critical threads to the other threads that they depend on. CPath can achieve high accuracy by using a machine learning based criticality predictor. Also, its criticality propagation mechanism increases its prediction coverage. On average, CPath is able to achieve 111% and 191% better critical path prediction ACP than BIS and CS respectively for a 16-core system. Using DVFS we were able to achieve more than 8% power saving.
Chapter 6

Conclusion

The communication framework in many-core system is a potential power and performance bottleneck. This is because of the poor global interconnect energy scaling and the restriction on performance of thread level parallelism due to need of synchronization primitives. As complex applications in many-core systems demand high performance and energy efficient data movements, increasing attention is being paid to provide an effective communication architecture. In this dissertation, we presented three different solutions to tackle the communication issues in many-core systems. Two of the proposals focus on physical communication infrastructure while the last one is for inter-thread communication.

First, we utilized an emerging technology, silicon photonics, to provide a low-power on-chip interconnects. We proposed a new low-power all-optical NoC approaching optical challenges such as power, the number of micro-rings and wavelengths. In our optical NoC, switches were designed based on the passive micro-ring resonators. We also proposed a novel deterministic wavelength routing algorithm. By employing this wavelength routing, we were able to provide contention-free network traversal, optimize optical switches, reduce the number of wavelengths and micro-rings, and lower the insertion loss. We showed that our optical NoC consumed less power compared with state-of-the-art proposals.

Second, we considered 3D NoC as an alternative to improve the performance of on-chip interconnects. Since one of the major issues in 3D integration is its poor heat dissipation, we proposed three thermal-aware mappings based on the LP model. Furthermore, to lay the groundwork for future research on thermal management with mapping algorithm, we explored the effects of thermal and performance constraints on chip temperature. We showed that the task power balancing had no major effect on the chip temperature. In addition, if congestion and router power constraints along with temperature constraints are used in mapping algorithm, explicit performance constraint does not affect PDP of a NoC. Furthermore, our thermal-aware mapping algorithm was capable of reducing the peak chip temperature compared with communication-aware mapping algorithm.

Finally, we proposed a dynamic critical path predictor for multi-threaded applications. We collected statistics at barriers. Then, we used a machine learning classification to choose
which statistics are more indicative of a thread’s future progress. To improve the prediction for applications that heavily depend on locks, we proposed criticality propagation. Thus, we employed lock contention information to propagate the criticality from critical threads to the other threads. The predictor results were used to guide dynamic voltage and frequency scaling to decrease the overall power consumption. We showed that our predictor achieved high accuracy and coverage by using the machine learning and criticality propagation, compared with state-of-art proposals.

This thesis advances our understanding of utilizing emerging technologies and algorithms to improve communication fabric. Although, research in 3D IC and nano-photonics devices are hot areas, several barriers remain to the adoption of them. Nowadays several industries make use of 3D integration to stack memory layers, e.g., Hybrid Memory Cube (HMC) by Micron and Knights Landing by Intel [1]. 3D IC faces poor thermal dissipation, complex and costly test mechanism and power delivery. These issues limit commercialization of stacking multiple core layers; multiple core layers provides higher power density and more complex logical functions to test than stacked memories. As an example of recent advancement in usage of on-chip optical interconnect is an integration of transistors and optical devices on a single chip to build an electronic-photonic system [121]. This system employs photonic components to communicate with other chips. While this system has demonstrated the ability to use on-chip optical interconnects, we still see challenges in terms of the limitation in number of wavelengths, insufficient laser output power and, reliability to build a complex optical NoC. These issues are device level problems and need to be addressed at that level. However, what might be considered a minor issue at the device level can be compounded considering a large system-level design; which means that it is not sufficient to consider devices in isolation. For example, although a waveguide crossing has a small insertion loss (0.12 dB), in a large optical NoC waveguide crosses are the major factor of the power loss. To help device level designer better understand the requirements and critical barriers of these technologies at the system level, computer architects should utilize these technologies in the system designs and investigate their benefits and limitations. Finally, when number of threads in a system increases, threads serialization overhead is aggravated due to the usage of synchronization primitives. This issue advocates for the need for dynamically analyzing and optimizing critical path of multi-threaded applications to meet their high-performance demands.

### 6.1 Future Work

In this section, I present some of the future research directions based on the proposed techniques and architectures in this dissertation:

- **Application-specific optical NoC.** Our proposed low-power optical NoC can be extended by considering an application’s bandwidth and latency requirements to provide a good quality of service. Based on threads’ bandwidth requirements, an available optical
bandwidth (number of wavelengths) at a destination can be shared between several sources to allow the destination to accept optical data streams from different sources simultaneously. For that purpose, an algorithm should be designed to decide how a destination responds to the sources’ requests on the control network. For instance, a destination can accept data from source$i$ on $\lambda_1$ and data from source$j$ on $\lambda_2$ to $\lambda_4$. Furthermore, using a thread mapping/migration algorithm that considers threads’ communication and their required bandwidths can improve the performance of optical NoC.

- **Online thermal-aware mapping.** In Chapter 4, we showed that thermal-aware mapping algorithms can improve the chip temperature. To capture temperature variations during runtime, online task mapping or migration should be used. Therefore, one future research direction can be designing an online thermal-aware mapping algorithm, which utilizes application characteristics such as threads’ communication requirements and their power consumptions. Reinforcement learning [123] can be used to implement that mapping algorithm.

- **Extending the critical path predictor.** Our proposed critical path predictor employs a machine learning classification to improve criticality prediction at barriers. However, the ML algorithm requires a table for its sampling data. To remove the need for that table and improve the prediction accuracy, a reinforcement learning algorithm, which is able to adjust its decisions on-the-fly, can be used. Moreover, the proposed predictor requires barriers in an application. To remove its dependency on barriers, we can use time-based predictor instead of barrier-based predictor for the non-barrier codes to collect the statistics and predict the thread criticality. In a time-based predictor, critical thread predictions will be made at regular time intervals instead of at barriers. However, the accuracy of the predictions will depend on the length of the time interval. The length of the time interval can be chosen by an application-aware algorithm.

This thesis utilizes different technologies, and investigates opportunities to improve the communication architectures. Research like this thesis is vital to the furtherance of future many-core systems, which may have hundreds of cores and greatly demand energy efficient communication architectures.
Bibliography


[68] James W Joyner, Payman Zarkesh-Ha, and James D Meindl. A stochastic global net-length distribution for a three-dimensional system-on-a-chip (3D-SoC). In *ASIC/SOC*


[78] Pranay Koka, Michael O. McCracken, Herb Schwetman, Xuezhe Zheng, Ron Ho, and Ashok V. Krishnamoorthy. Silicon-photonic network architectures for scalable, power-


