PREDICTION MODELING FOR DESIGN SPACE EXPLORATION IN OPTICAL NETWORK ON CHIP

SARA KARIMI

A Thesis

in

The Department

Of

Electrical and Computer Engineering

Presented in Partial Fulfillment of the Requirements for the

Degree of Master of Applied Science (Electrical and Computer Engineering)

at

Concordia University

Montréal, Québec, Canada

November 2016

CONCORDIA UNIVERSITY
SCHOOL OF GRADUATE STUDIES

This is to certify that the thesis prepared

By: Sara Karimi

Entitled: "Prediction Modeling for Design Space Exploration in Optical Network on Chip"

and submitted in partial fulfillment of the requirements for the degree of

Master of Applied Science (Electrical and Computer Engineering)

Complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the final examining committee:

_____ Dr. M. Mehmet-Ali _____ Chair

_____ Dr. C. Assi (CIISE) _____ Examiner, External
To the Program

_____ Dr. W. Hamou-Lhadj _____ Examiner

_____ Dr. J. Trajkovic _____ Supervisor

Approved by: _____
Dr. W. E. Lynch, Chair
Department of Electrical and Computer Engineering

_____ _____

Dean, Faculty of Engineering and
Computer Science

ABSTRACT

Prediction Modeling for Design Space Exploration in Optical Network on Chip

Sara Karimi

In at least a decade chip multiprocessors (CMP) have been dominating new commercial releases due to computational advantages of parallel computing cores on a single chip. Network on Chip (NoC) has emerged as an interconnection network of CMPs. But significant bandwidth that is required for multicore chips is becoming a bottleneck in the traditional (electrical) network on chip, due to delays caused by long wires in the electric NoC. Integration of photonic links with traditional electronic interconnects proposes a promising solution for this challenge. Since there are numerous design parameters for opto-electrical network architectures, an accurate evaluation is needed to study the impact of each design parameter on network performance, and to provide the most suitable network for a given set of applications, a power or a performance goal. In this thesis, we present a prediction modeling technique for design space exploration of an opto-electrical network on chip. Our proposed model accurately predicts *delay* (includes network packet latency and network contention delay) and *energy* (includes static and dynamic energy consumption) of the network. Specifically, this work addresses the fundamental challenge of accurate estimation of desired metrics without having to incur high simulation cost of numerous configurations of the optical network on chip architecture. We reduce the number of required simulations by accurately selecting the parameters that have the most impact on the network. Furthermore, we sparsely and randomly sample the designs build using these parameters from an Optical Network on Chip (ONoC) design space, and simulate only the sampled designs. We validate our model with three different applications executing on a large set of network configurations in a large optical network on chip design space. We achieve average error rates (root relative squared error) as low as 5.5% for the delay and 3.05% for the energy consumption.

ACKNOWLEDGEMENTS

I would like to take the opportunity to thank and express my gratitude to my supervisor, Dr. Jelena Trajkovic. She does not only give me a tremendous amount of experience and knowledge in our research but also teaches me valuable lessons while providing guidance and helpful insight about living as a new international student in the completely unfamiliar country. I also want to thank her because she has been more than a supervisor to me, she was my first friend in Canada, and she taught me always to be positive and confident. Most importantly she trusts me and encourages me to attack the problem no matter how big it seems at first glance. I am truly honored to work with her and have her as my supervisor.

I also want to thank my friends and colleagues in our Lab: Umair, Kamil, Evan, Shafigh, Aryan, Hari, Amine and Ehsan for their help, comments and suggestions in my research. We had many memorable days that we worked, talked and laughed.

Last but not least, I would like to thank my family and close friends for their support during my study. I especially want to thank my beloved mother for helping me get through this chapter in my life like she always did in the previous ones.

List of Content

List of Figures

List of Tables

List of Equations

List of Listing

List of Abbreviations

NoC	Network on Chip
ONoC	Optical Network on Chip
CMP	Chip Multi Processor
WDM	Wavelength Division Multiplexing

List of Writing Conventions

Features	We use feature to refer to attributes, predictors, or input variables of prediction model
Delay	We use delay to refer to network latency and contention delay of network
Energy	We use energy to refer to static and dynamic energy consumption of network

CHAPTER 1

1. Introduction

1.1. Motivation

By scaling down the technology node, we can have a larger number of cores on a single chip. If current trends continue, there will be hundreds of cores on a chip in five to ten years [1]. When the number of the cores grows to tens and hundreds on a chip, traditional shared medium network like shared bus, cannot support the required high bandwidth, performance and power. For addressing this challenge in modern system architecture, Network on Chip (NoC) had been proposed as the interconnection network for Chip Multiprocessors (CMP). Although electrical NoC is a good solution for replacing traditional shared medium network, it cannot scale well in performance and power when the number of cores grows to hundreds or thousands. Optical Network on Chip (ONoC) is a promising solution for solving power and performance bottleneck of the on-chip network. Some of the advantages of using ONoC are its high bandwidth, bit-rate transparency for the power consumption in switches and waveguides which means power consumption of these devices are not dependent on the number of transmitted bits or distance, scalability and low loss in the optical waveguide that cause power consumption be completely independent of transmission distance. In designing ONoC, there are many parameters and design alternatives that can be considered even for a single network architecture. These several alternatives create a significant design space that a designer can choose its design from it, based on the preferences. Evaluation by simulation of all of these design space points required a lot of time and effort. In this thesis, we proposed a prediction model that we derive based on sampled simulations. This prediction modeling technique that we present in this work can address a fundamental challenge in simulation cost of design space exploration. Our prediction model has been derived from a set of data that we obtain from simulating of numerous design alternatives. Although we simulated a large number of design alternatives, these simulations are still a small part of entire design space points (i.e. all possible design alternatives). Our derived model can be used for predicting the behavior (i.e. delay and energy) of these simulated design alternatives as well as other design alternatives from entire design space that we do not simulate. This work

provides a rapid design space exploration for a designer who wants to capture the tradeoffs between many alternative designs in an ONoC architecture.

1.2. Problem statement

A designer has many possibilities to design an optical network on the chip as an interconnection network for CMPs. This design variety comes from network configuration parameters that provide a large amount of alternatives for each aspect of designing an optical network on chip. For finding a design configuration that can fulfill our desired outcome in latency and delay of the network as well as network energy consumption, a designer needs to do a massive search and exploration among all the design possibilities in the entire design space. Exploring the full cross-product of many design parameters requires a large number of long-running simulations. Moreover, efficient design space exploration is constrained by the significant computational costs of simulators. Long simulation times may cause designers to limit design studies and consider only small subsets of the full design space. These limitations and constraints will make results and conclusions that cannot generalize to the other alternatives in larger design space. Therefore designers need a method to explore among large design space without spending too much time to test and evaluate all the design alternatives.

1.3. Proposed Solution (Methodology)

In this thesis, we come up with a prediction model that can be used by the designer to find out about delay and energy of network without simulating all the design alternatives independently. In this technique, at first, we sampled different design configurations from a full ONoC design space and simulated these designs in our simulator to find out delay and energy consumption of these architectures. In this way, we obtain a dataset from some design points of the entire design space. Then by using statistical inference and machine learning methods, we find several prediction models that can predict outputs (i.e. delay and energy) of the different design alternatives in entire design space with very good accuracy. These prediction models have been built based on the dataset that we gather in the first step. We use this dataset as an input for a learning process that formulates a prediction model. The derived prediction model can predict

output's values for the other design configurations in design space that we do not use in the first step to build our model. In other words, we gather a limited number of samples from entire design space and train a model for predicting the outputs of entire design space. After obtaining the prediction model, we validate it on new samples, and we measure the accuracy of the prediction. In chapter 3 we will go through the theory of several prediction models that we explore in this thesis. In chapter 4 we discuss our different design configurations that generate our dataset, and then we introduce the required process on the dataset for improving models. In chapter 5 we present the experimental results and evaluate our models to find out how well these models can predict other design configurations in design space.

1.4. Related Work

There has been significant work in optimizing the ONoC simulators to reduce required time for design space exploration, and also fundamental work for performance modeling of multiprocessors systems. Our work is unique in its domain Optical Network on Chip because we consider both architecture and application specific parameters of the design to obtain accurate delay and energy prediction. Therefore, to the best of our knowledge up to date, there has not been any work that tackled the same problem. Hence we will provide background and compare our work to the other works that have been done in these three categories: (i) Optical Network on Chip Architectures and Design (ii) Design Space Exploration of Network on Chip (iii) Regression Modeling for Design Space Exploration.

1.4.1. Optical Network on Chip Architectures and Design

Corona [2] is an optical Network on Chip (NoC) architecture that uses an optical crossbar with optical token ring arbitration to permit a node to send data. It uses a Multiple-Writer-Single-Reader (MWSR) [3] optical bus. When the number of nodes increases, both the waiting time for receiving a token and the network diameter increase (i.e. longest distance between every two nodes of the network that represents the size of the network). Corona implementation uses a lot of wavelengths (64) to modulate data packets to improve the latency; however, this requires an impractically a large number of microring resonators resulting in a large area, and moreover high power consumption which rapidly increases with the network size. In order to reduce the number of waveguide and wavelength [4] proposes an Optical Ring Network on Chip (ORNoC)

architecture. The major advantage of ORNoC is that the same wavelengths can be reused to realize multiple communications on the same waveguide, at the same time, with no arbitration required. By using Wavelength Division Multiplexing (WDM) that allows multiple signals to be transmitted simultaneously, and wavelengths reuse method, ORNoC architecture provide a sufficient number of rings to support a contention free architecture. In ORNoC, (wavelength, waveguide) pair is statically determined based on an input matrix that determined all possible connections among clusters. It also has no waveguide crossing, because waveguide is forming a ring that only has a few bends. Its serpentine layout also causes smaller power loss and occupied smaller area [5]. It uses on-chip laser and can be applied to 2D and 3D designs (using a set of stacked electrical layers and one optical layer). CHAMELEON [6] extends the ORNoC with a reconfiguration layer that can open point to point communication channels at a run time. Therefore it causes better utilization of bandwidth according to the application traffic. The configuration process is managed by a control network implemented in an electrical layer on top of the optical network.

In ATAC [7] cores are connected via an electrical and an optical network. The optical network is used for global broadcasting. It uses a ring-based architecture similar to ORNoC, and it is also contention free, based on WDM. However, it does not support simultaneous communication between the source and multiple destinations unless it is a broadcast of the same message. ATAC implements "distance-based" routing algorithm to decide whether to send a unicast over the optical network or just use the electrical network (broadcast messages always use the optical network).

ATAC+ [1] has the similar architecture of ATAC, but it uses the athermal ring resonators [8] to compensate the temperature shift, on-chip Germanium Lasers and an adaptive "single writer multiple readers" (SWMR) optical link that improves power consumption. Detail explanation of these components is in section 0. [1] studied the effect of some system architectural parameters such as routing strategy, cache coherence protocol, etc. on delay and energy individually. However, there is no study to monitor the effect of a comprehensive set of design parameters of the network together in design. A comprehensive study of design alternative with different parameters needs an exhaustive simulation that has not been done for any of the mentioned architectures due to simulation cost. In our work we go through a massive design space points

and use a simulation free model for delay and energy prediction. Our proposed technique generates a model that can evaluate the effect of all configuration parameters together in design.

1.4.2. Design Space Exploration and Modeling of Network on Chip

ORION [9] simulator proposed a NoC power and area model used for design space exploration. It provides parametrized power and area models for routers and links. It does not model any optical components, and also supports only limited repeated links. ORION has incomplete architectural models and timing for the router (i.e. lack of delay model for router components) and uses a fixed set of technology parameters. All links are optimized for minimum delay. It also has very low accuracy for modern technologies: 3x power overestimate for 65 nm technology node, 400 MHz x power and 2x area overestimate for 45 nm technology node, 1 GHz, and more than 5x power overestimates for links.

PhoenixSim [10] is a Photonic NoC Simulator that model Optical component in NoC. PhoenixSim lacks electrical models and relying instead on ORION for all electrical routers and links. PhoenixSim uses fixed numbers for energy estimations for electrical interface circuitry, such as modulator drivers, receivers, and thermal tuning, losing many of the interesting dynamics when transistor technology, data-rate, and tuning scenarios vary. It also uses a basic Photonic NoC [11] that use the electrical network for control flow and optical network for data transmission. The electrical signal is sent first to reserve the path for optical transmission. In this architecture, contention delay may be high. Because the entire path between a source and a destination is reserved ahead of time and it may happen that no free optical path available. Thus the new communication needs to be delayed till the previous ones free up the required resources. In our work we use DSENT [12] for delay and power calculation and Graphite as our simulator that solves the problem of lacking electrical model, and electrical interface for optical devices. Moreover, our network model ATAC does not make any reservation for acquiring the path. Therefore there is no more contention delay from this side.

DSENT [12] is a tool for area and power evaluation of opto-electrical on-chip interconnects. DSENT provides models for optical devices and the electrical backend circuitry including modulator driver, receiver, and ring tuning circuits. It also models the interface between electronic parts and photonic parts. DSENT had been used in Graphite [13], the network-on-chip simulator, which provides both delay and power modeling. Moreover, previous studies like

PhoenixSim and Corona have been used coarse, higher level models and unrealistic traffic patterns like synthetic workloads or captured traces, but Graphite uses real application workloads to evaluate the design architecture. Although this simulator has been modeled electrical and optical components with low errors, and therefore it can calculate power and delay with good accuracy, studying all the possible design needs a lot of time. For example, one simulation of a network with the size of 4x4 cores takes 10 minutes on Intel Core i7 host machine running Linux, but 256x256 cores take more than 1 hours depend on the running application. Due to long simulation time, exploring a large design space is nearly impossible and impractical for large scale industrial designs. Our work proposes a simulation free modeling by introducing a statistical inference into simulation framework and a machine learning technique for prediction of delay and power.

1.4.3. Regression Modeling for Design Space Exploration

The authors in [14] propose regression modeling for predicting performance and power for various applications executing on any microprocessor configuration in a large microarchitectural design space. Although their model has less than 4.5% error rate in its prediction, their work requires numerically solving and evaluating linear systems to find out an efficient formulation for the models. Moreover, for finding the parameters that have a significant impact on performance and power, they use domain specific knowledge of microarchitecture to specify nonlinear effects and interactions between parameters. Therefore their model relies on designer domain knowledge instead of a stepwise procedure, and also it makes their work impractical for others domains.

In [15] they propose a regression-based application specific performance model for GPU design space exploration. It can predict program run time for any point in the design space. Although in our work we applied the same approach for finding the prediction model, our problem is completely in a different area. Therefore the difference between GPU and ONoC design space and their limitations, characteristics and features make their approach unsuitable for ONoC. Moreover, in this thesis, we present not only a regression model but also several other prediction models like decision tree and neural network and compare them together. And finally, we proposed a Regression Model Tree that shows more accurate result compare to the other

models. Another advantage of using model tree is that its training speed is very fast compared to the other models.

1.5. Contribution

This thesis proposed a prediction model for design space exploration in the optical network on chip through the following main contributions:

1- Presents the first prediction modeling methodology for design space exploration of the optical network on chip. This methodology results in a prediction model that can accurately predict delay and energy for any design configuration without simulation. We also compare several prediction models and study their pros and cons. Moreover, we show that by sampling from a big dataset we will have a smaller dataset that can be used for deriving a prediction model with acceptable accuracy.

2- Performs the first comprehensive delay and energy consumption analysis for the optical network on chip architecture using numerous simulated designs. We simulate more than thirteen thousand different design configurations from three different application benchmarks and evaluate delay and energy consumption of them using our prediction model.

3- By analyzing the impact of each design parameters on delay and energy consumption of the network, we provide a guideline for future ONoC research that can be used by designers to study the different tradeoffs in their network design. We identify the parameters and architectural decision that has the most effect on delay and energy consumption of network. In the last chapter, we capture the interaction between design parameters and delay and energy consumption that can help the designer to efficiently choose the design with the best payoff without going through extensive simulations and explorations among many design alternatives.

CHAPTER 2

2. Photonic Network on Chip Architecture and Interconnects

In this chapter, we will present an overview of NoC and ONoC architecture and also give a brief explanation of used nanophotonic devices. It provides a necessary background in ONoC and photonic devices.

2.1. Network on Chip for Multicore Systems

As Moore's law states the number of transistors doubles in an integrated system every 18 months. It causes that number of cores in multi-core architectures expected to double every 18 months [16]. We need an efficient communication approach to exploit as much as we can from computational resources available in the system on chip. Traditionally a shared medium network, such as dedicated point to point signal wires, shared buses or segmented buses with bridges had been widely used. These shared mediums are a great choice for a system with maximum five cores and ten bus masters. But when the number of cores grows, scalability becomes a big issue. Moreover, in such a case, the system becomes inefficient regarding energy due to congestion frequently happening in communication architecture. There are some attempts at addressing communication challenges on chip, from transistor redesigning to 3D-stacking [16], but none of them give a promising answer because they are a costly procedure. Therefore by increasing the number of on-chip cores, a scalable and high bandwidth communication fabric becomes critically important. As a result, network on chip rapidly replaces traditional buses. Network-on-Chip uses the concept of routers and links, from traditional networking to replace a shared medium such as a bus. Network on chip has routers at every node that connected to the neighbor's router via electrical interconnects (wire) called links [17]. Therefore, network on chip improves the scalability of the multicore system on a chip with better performance compared to traditional shared buses.

2.2. Optical Network on Chip

As we discuss in the previous section, in modern microprocessor architectures multicore systems are the dominant trend. By scaling down the technology node, we can have a larger number of cores on a single chip. If the current trend continues, there will be hundreds of cores on a chip in five to ten years [1]. When the number of cores grows to tens and hundreds on a chip, memory bandwidth for supporting parallel computation increase dramatically. Since the number of memory pins cannot increase as fast as the number of cores [18], and also pin data rate increases slowly [16] memory bandwidth becomes a significant bottleneck in multicore systems. Supplying enough bandwidth with electrical links cause a remarkable increase in overall power dissipation [3]. Moreover, it results in thermal issue with current chip packaging technology [18]. Therefore electrical interconnect cannot provide required high bandwidth, while keeping performance, power and area in an acceptable range. Studies in [19] indicate that almost 50% of the power on the chip is caused by the interconnections. [19] has been shown that more than half of the dynamic power is dissipated by interconnections.

One promising solution for solving power and performance bottleneck of on-chip interconnection is Optical Network on Chip (ONoC) that use silicon photonics to connect ever increasing number of cores on a single chip. Optical components like light sources, waveguides, modulators, detectors are needed for creating such a network that will explain in details in next section. Hybrid opto-electrical NoC microarchitecture that uses optical links integrated with the electrical network was shown to have good scalability and performance. More details about opto-electrical NoC will be provided in section 2.4. The current technology is mature enough to allow the photonic integration on a chip by using these CMOS-compatible optical components [4]. However, these optical components are vulnerable to fabrication non-uniformity (i.e. process variation) that may cause decreasing of performance or even system failure [20]. As we said before, electrical wire and buses scale poorly with technology shrinking. However, studies presented in [21] and [22] show that it is possible for optics to entirely replace electrical connections on chip by using an optical device that can build with standard CMOS processes. Low level implementation details issues have already been started to be researched, and some preliminary mitigation techniques have been proposed.

In this project, we will use an optical NoC architecture called ATAC [23]. ATAC proposed an optical interconnect that provide between 1.8x and 4.8x better energy-delay product than conventional electrical-only interconnects. This architecture will be discussed in section 0.

One of the main advantages of using optical links is that optical medium has bit rate transparency. The power consumption of electrical network depends on the number of bits that are transmitted and also the distance between source and destination, therefore, dynamic power scales with bit rate. In contrast, a photonic system that is made by photonic switches and waveguides is bit-rate transparent, which means power consumption of these devices are not dependent on the number of transmitted bits or distance. [24] However overall energy cost for modulator driver is based on data rate [12], photonic switches on and off once per message and their energy consumption do not depend on the bit rate. Moreover, since waveguides are bit rate transparent dynamic energy consumption is independent of the receiver place [18].

Optical NoC can reduce power consumption while keep required high bandwidth for CMPs. That's because in electrical NoC, when a packet is sent to the network, it needs to buffer, regenerate and then transmit in routers and links of the network for multiple times till it reaches its destination. These switching and regenerating the packet in CMOS technology consume dynamic power that grows with data rate. However in photonic NoC, due to low loss optical waveguide, the data is transmitted from source to destination without any need to repeat, regenerate or buffer the packet. Furthermore, because of bit-rate transparency no extra dynamic power is consumed for routing packet with the optical network [3] Although in some Photonic NoC architecture like Corona, it needs to reserve the optical path before any photonic transition by sending an electrical signal through the network and it consumes power. In some architectures like ATAC, there is no need for reservation of the optical path.

There is two type of transmissions: serial and parallel. Using serial transmission has this advantage that it needs less number of waveguides, but it needs another component such as serializer and deserializer to make the data transmission possible. On the other hand in parallel transmission, there is no need for these extra components, but it wants more waveguides and therefore larger area to make data transmission possible. Wavelength Division Multiplexing (WDM) is a technique that uses multiple wavelengths to transmit multiple instances of optical data in a single optical waveguide (a unit that its responsibility is transferring light on chip.), at

the same time. By using WDM in the photonic system, we can achieve high bandwidth density. WDM uses a multiplexer at sender side and a demultiplexer at the receiver side of the connection. Advances in optical domain provide even denser wavelength division multiplexing for even higher bandwidth [2].

2.3. Optical Components and Modules

In this section, we give a quick overview of main optical components required for an optical NoC system. The set of components consists of single or multi-wavelength waveguides for routing the optical signals on-chip, the laser source, modulators for imprinting bit streams onto wavelengths, resonant rings for wavelength selective filtering, and photodetectors/receivers for converting the optical signal back into the electrical domain. These are some explanation about main optical components:

Waveguides: The unit that carries and transport light on the chip is a waveguide. The main characteristic of the waveguide is its effective index that indicates how light propagates through the waveguide. Waveguides are also characterized by the optical loss that they introduce into the system. The loss that the waveguide introduces largely depends on its layout because every loss is defined for each straight line segment, bend, branch, crossing or waveguide combiner. Recent work has shown that waveguides can have loss down to 0.3 dB/cm [25]. There is also low-loss silicon nitride waveguide that has been shown loss down to 0.1 dB/cm [26]. By using Wave Division Multiplexing technique, a waveguide can carry multiple frequencies of light, and each of them has its information to travel simultaneously and in the same waveguide.

Laser: Laser is a source that produces the optical signal that is coupled into the waveguide. There is two type of lasers that can be used for on-chip communication, on-chip laser, and off-chip laser. The on-chip laser has complex integration, and it generates high heat. [27] Presents Germanium Laser that is on the chip and has fast throttling feature (throttled laser will be discussed in section 2.4). On-chip lasers have this advantage that they avoid coupling and distribution losses (suffered when bringing in light from an off-chip source) and are close enough to be shut down and restarted very quickly. In contrast, the off-chip laser has easier manufacturing, but it has a high loss. It is also difficult to turn on/off due to the communication latency from the sender to the laser as well as the energy needed to communicate off-chip.

Moreover, a laser source can generate a fixed wavelength or be tunable to generate multi-wavelength based on wavelengths of operation and maximum output power of the laser [11].

Micro-Ring Resonator: A Microring Resonator (MR) is a small ring structure, with typically between 1 and 10 μm in radius [28] made of the curved waveguide that is placed adjacent to the waveguide. The typical distance between a ring and waveguide is between 0.4 and 0.6 μm [28] and it defines the way the ring and the waveguide are coupled. Input optical signal couples to MR with cross-over coupling coefficient and then couple to the adjacent waveguide with same coupling coefficient. Uncoupled light continuous its path with straight-through coefficient [20]. Radius of each ring is indirectly proportional to its resonance wavelength (λ_{MR}). It causes that MR be highly sensitive to fabrication and process mismatches. The resonant wavelength of MR can be controlled by adjusting the ring radius or the index of refraction [29]. Index of refraction strongly depends to temperature, as result resonance wavelength of MR is highly sensitive to temperature. One of the challenges of MR-based designs is that these MR devices are extremely sensitive to the temperature. Even 1◦C change in temperature causes dramatic shift in the resonant frequency. Typically the wavelength shift is between 0.07 nm/◦C to 0.32 nm/◦C [30]. Hence MR needs active thermal tuning [29] .

To overcome this challenge, a technique that calls “thermal trimming” usually is used. Here a small electrical heater is placed below the microring to stabilize the ring resonances. For example, a shift of 3.6 nm could be reached at the heater power of around 180 mW [31]. Another challenge is highly sensitiveness to manufacturing variation of ring devices. The rings are particularly sensitive because of their small overall dimensions. For example, 1nm of variation in width and height leads to 1nm to 2nm shift in resonance wavelength of the ring [31].

When an optical signal with a wavelength λ_s travels on a waveguide, upon encountering an MR the following scenarios are possible:

- When wavelength of traveling optical signal is equal to resonance wavelength of MR (i.e. $\lambda_s = \lambda_{MR}$) the signal will couple into the MR with the same wavelength as the signal. And then couple out into the perpendicular waveguide.
- When wavelength of traveling optical signal is different from resonance wavelength of MR (i.e. $\lambda_s \neq \lambda_{MR}$) the signal will propagate along the waveguide.

- When optical signal wants to transmit on the network, the signal coupled into an appropriate MR with the same wavelength and them couple out to the waveguide.

Therefore, an MR can be used to select an appropriate signal on reception (filter), and to couple in the modulated optical signal into the network (modulator), to perform switching within the routing network (router or a switch) [4]. Accordingly, we define these three components that work based on MR, here:

- **Modulator**: Modulation means encoding digital data (i.e. logic '0' and logic '1') to an optical signal with different optical power. Microring Modulator does both "modulating" the signal and "inserting" it into the network. (Figure 1 part a)

- **Filter**: This function selects one wavelength from a WDM signal traveling on a waveguide. The selected signal is extracted from the incoming waveguide and coupled into the photodetector as shown in Figure 1.b.

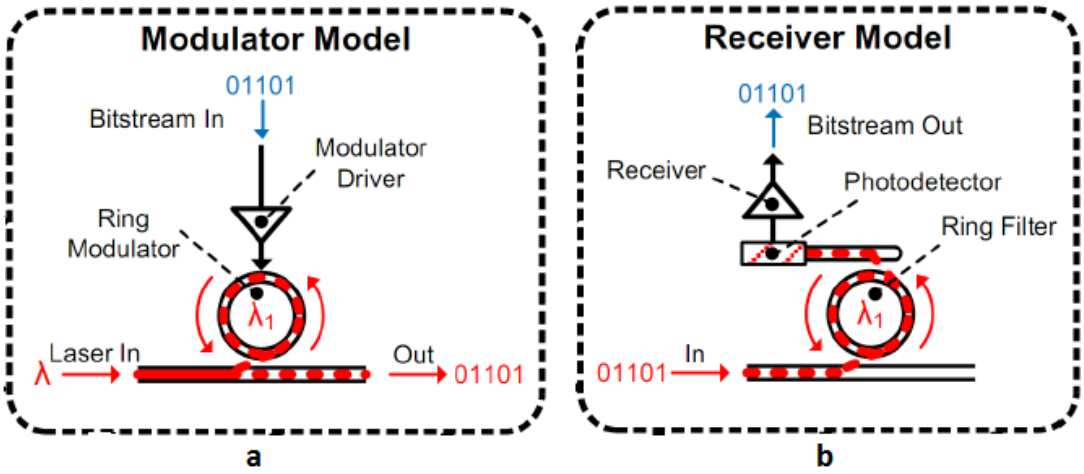


Figure 1: a) Modulator Model b) Receiver Model (Filter and Photo Detector) [32]

- **Switch**: Microring resonator can be used for designing routing element, i.e. switch. The switch consists of one, two or more number of MR. These optical switches are composed of optical waveguides and MR(s) and operate as a classic electrical switch. The signal goes into the switch from one of the input ports, and then based on its wavelength, it selects one of the two different output ports. If the wavelength of signal (λ_i and λ_j) is the same as resonance wavelength of switch (λ_i) the signal takes the opposite waveguide

(straight state, Figure 2 part a) and if they are not the same ($\lambda_j \neq \lambda_i$), the signal goes diagonal (Figure 2 part b).

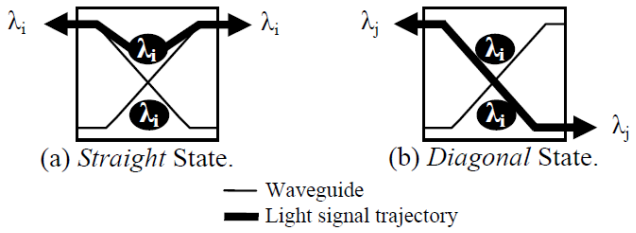


Figure 2: Optical Router (Switch) [33]

Photo Detector: Also known as a photodiode, the photodetector is used for absorbing light and producing electrical current. The main characteristic of the photodetector is its responsivity that indicate the minimum optical power necessary to detect a logic "1" reliably. The responsivity of a photodiode is a crucial design parameter: the received optical power needs to be greater than the threshold specified by the responsivity. The received optical power is given by the following equation:

$$P^{dBm}_{\min\_receiver} = P^{dBm}_{\min\_laser} - L^{dB}_{wc}$$

Equation 1: Minimum required power for photodetector [31]

Where $P^{dBm}_{\min\_receiver}$, is the minimum optical power received by the detector in dBm, $P^{dBm}_{\min\_laser}$ is the minimum laser output power in dBm and L^{dB}_{wc} is the worst case losses in the optical path in dB. Therefore the designer has to make sure that the worst case optical losses (from laser source to photo detector) do not attenuate the optical power received by the detector beyond the threshold needed for correct identification of the transmitted value.

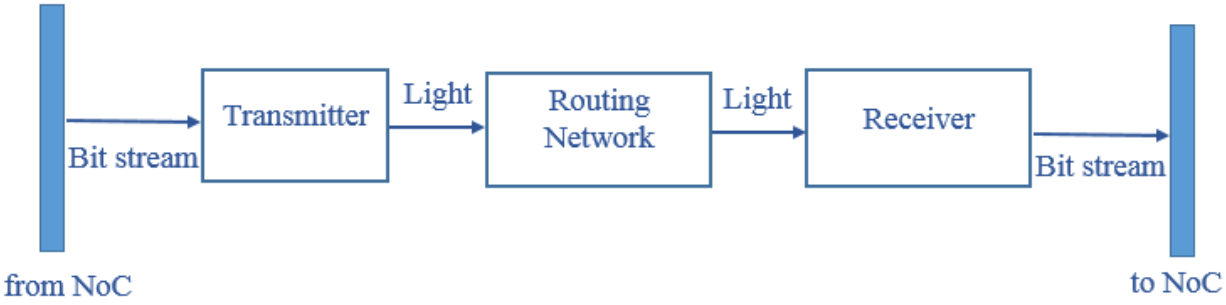


Figure 3: Optical Network Overview

We will describe how a photonic network works in general. Figure 3 shows a general overview of photonic communication. This structure can divide to three main parts: Transmitter, Routing Network, and Receiver.

Sender puts its data to the link, and in transmitter part, the serializer is responsible for up-converting the input data from a bus data rate to the higher sterilized data rate. Serializer does this by combining multiple incoming data stream (i.e. wire). [18] Then a Modulator Driver converts electrical signal to optical signal. This high-speed conversion also can combine data on a single wavelength channel that can be combined with other optical signals in other wavelengths to form a dense wavelength-parallel optical signal (through WDM).

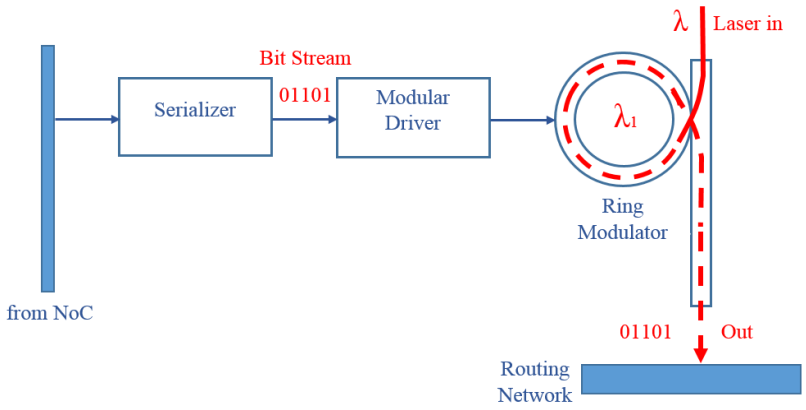


Figure 4: Transmitter Part

In routing network part, switches and routers direct packets to the destination using an active or passive router. In receiver part, appropriate wavelengths filter out and then convert to the electrical current by a photodetector which able to absorb light and generating electrical current. The wavelengths that are not filtered out and detect continue their path in the network. For the detected wavelength, the current that generate in photodetector will use in a receiver to convert to a voltage. This voltage amplifies as much as it needs to reach the certain level that digital circuits can work with it. Then a deserializer that has the inverse functionality of a serializer down-convert the high-speed optical data rate to an electronic bus data rate [18]. Figure 5 shows receiver part.

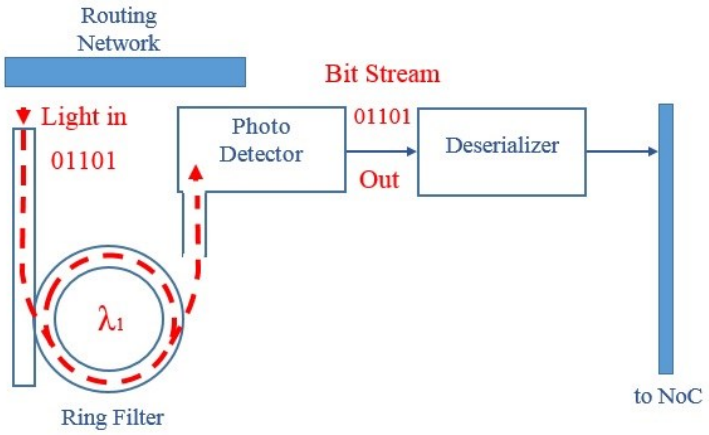


Figure 5: Receiver Part

2.4. Network architecture of ATAC

In this section, we present an overview of "ATAC" [1]: a hybrid electro-optical network architecture used for on-chip communication. This architecture combined the electrical interconnects with cutting-edge on-chip optical communication networks and can scale to 1000-core processors at the 11 nm technology node. We use ATAC because it provides a wide range of design alternatives for network architecture (e.g. several routing policies, several components for an optical receiver, etc.). Moreover, ATAC fully integrates all levels of the system from electrical and optical device models to applications.

ATAC consists of several networks that are different in architecture and implantation and also have a different use. They are ENet, ONet, BNet, and StarNet. The first network called "ENet, " and it is an electrical mesh network that connects processing cores via point to point connections. It is like network on common multicore processors such as Intel Teraflop [33]. ENet is used for short range communication. Communication is considered to be a short range communication if the packet needs to travel among a few number of adjacent cores usually placed in the same cluster. The optimal value for considering a communication in a short range may change as the load of network increased. But a careful study in [1] shows the optimal value changes from 5 to 15 to 25 as network offered load increased. 5 to 15 to 25. The ENet is shown in green in Figure 6.

In addition to this electrical network, there is an optical network called “ONet”. ONet is ideal for long distance communications because it provides low latency and energy efficient global interconnect. Processing cores in ATAC architecture are divided into clusters. Each cluster consists of a given number of cores that is decided in design time. A cluster has an ONet endpoint called “hub”. Hub is an interface between optical modules in the ONet and electrical modules of the ENet inside of each cluster. In the case of sending a packet, if packet wants to use ONet to go to the destination, it will use ENet to reach to the hub in the cluster. In the case of receiving a packet, it will use an electrical network that connects each core in the cluster to the hub of that cluster. A designer can choose one of two available networks for this purpose: a point to point electrical network called “StarNet,” or a small electrical broadcast network “BNet”. StarNet and BNet are used exclusively by cores within the given cluster for receiving packets from the Hub, and therefore from ONet. StarNet has a star topology and has a designated connection between each core and the corresponding hub. BNet is simply a fan-out tree with no router or crossbar that connect each core to its hub.

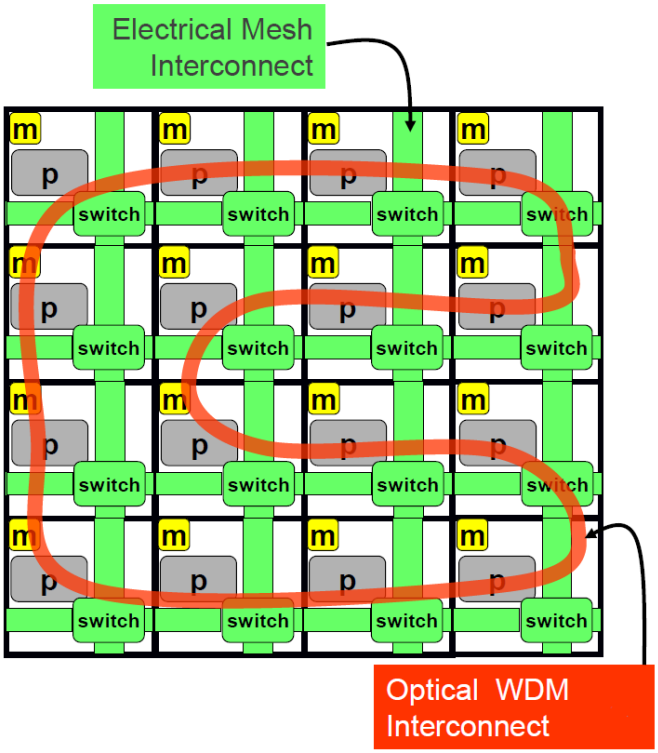


Figure 6: Interconnect in ATAC [1]

The StarNet is a 1 to 16 demultiplexer that is connected to 16 point-to-point links for forwarding the data from the hub to the cores in the receiving cluster. For a unicast communication, StarNet uses only one of its link while for a broadcast it uses all 16 links. In contrast, BNet always forwards data to all cores in receiving cluster, and if it is a unicast message only a single core which is the destination of the message processes it, and other cores immediately drop the message without any process. Because of this different behavior in unicast and broadcast StarNet consumes lower dynamic energy compare to BNet (~ $1/8^{th}$) [1]. In the rest of the thesis, we will use StarNet due to its better performance.

The hub in each cluster connects to the other hubs from another cluster by low latency and contention free network ONet. This interconnect consists of waveguides that pass through every hub and loop around the chip to make a ring connection like what has been shown in Figure 6. ATAC supports parallel communication and therefore for each data bit there is a separate waveguide: e.g. for transmitting 32-bit data, there are 32 parallel waveguides. Also note that even though this is a ring network, its layout is serpentine rather than circular (as mentioned in section 1.4.1). For transmitting data via waveguides, a hub needs to have circuits for electro-optical and opto-electrical conversion, namely optical driver and a modulator for electro-optical conversion; and an optical filter and a photodetector for opto-electrical. We discuss it, in details in section 2.3.

As Figure 6 shows waveguides form a loop, therefore a signal sent by any hub in the network will be reached to all of the other hubs before it filters out by one of them. The ONet uses wavelength division multiplexing (WDM) to avoid contention. Each hub's modulators are tuned to send on a unique wavelength. Each hub also contains receive filters tuned to all of the other wavelengths. These filters eliminate the need for arbitration in the optical network. Filters of each hub tuned to $1/N^{th}$ of the signal (where N is the number of destinations) and the rest will pass through the hub. It causes that the ONet functionality to be similar to a broadcast bus, but without any bus contention. In contrast, electrical network and non-serpentine optical topologies should send multiple unicast messages for supporting broadcast in the network. Therefore compare to the electrical network and other optical topologies, ONet transmission is faster (signal reaches all hubs in less than 2ns). Hence, broadcast communication takes place over the optical network (ONet). Moreover, it is possible to switch between broadcast and unicast modes

by using integrated on-chip lasers. This laser that called “throttled laser” can dynamically adjust laser output power and turn off when the link is idle. Therefore it saves power when the link is idle and on unicast. Traditional standard laser always works in full power even when it is used for unicast or when the link is idle. Based on the current trend in laser technology it is estimated that throttled laser can turn on and off in 1 ns. For switching to unicast mode, the laser will be turned on and set up its power for one receiver and then a notification send to a receiver to tune in. For switching to broadcast mode, laser power will be adjusted all receivers supply and then a notification sent to all receivers to tune in.

The dynamic energy consumption in ONet is constant for communication over ONet and it is independent of place of the receiver on the chip. In contrast, dynamic energy consumption is directly related to the number of hops between sender and receiver for communication over ENet. As we mentioned earlier, in the case of broadcast ONet is optimized for both energy and performance. However, in the case of unicast, we should choose between two different routing policies: “cluster_based” and “distance_based” routing. In cluster_based routing scheme packets will be sent over ENet if source and destination of the packet be in the same cluster, and otherwise, the packet sends over ONet. In distance_based routing scheme, there is a parameter called distance threshold r_{thres} that set by designer in design time, and a packet that its distance between source and destination is less than r_{thres} will be sent over ENet and for distance at r_{thres} and above it will be sent over ONet. Having these two different routing policies may be helpful to use network resources uniformly. For example if an application have low network demands we can send all unicast over the ONet, but in the case of application with high network demands we should use ONet and ENet in balance to maximize the throughput of the network. Fining optimal routing policy needs a careful study in both energy and performance of the network that we try to do in chapter 5.

2.4.1. Studied Outputs

In our simulations, we evaluate our network models in two aspects: delay and energy consumption. In delay, we consider two outputs which are contention delay and packet latency. Contention delay computes the queuing delay when message wants to access a shared object, like network links, off-chip memory and shared cache. Packet latency is network latency of sending a packet from one tile to another. In energy consumption, we consider two outputs which

are static and dynamic energy. Static energy is non-data dependent energy consumption comes from optical components like the laser and thermal tuning. Dynamic energy is data dependent energy consumption comes from components like the router, electrical link, receiver, modulator, serializer and deserializer.

CHAPTER 3

3. Regression Theory

In this chapter, we introduce basic concepts of regression theory and regression model tree. We apply a regression tree modeling technique to estimate delay and power on ONoC system efficiently. The initial idea for predicting delay and power was applying a linear regression model to the dataset to estimate delay and energy (we may refer to them as outputs in this thesis). This linear regression model is a sum of weighted variables that is used as predictors to calculate an output value. This basic linear regression for estimating the outputs do not perform well with acceptable accuracy. Therefore for improving the prediction, we considered using "Tree". As it will be shown in section 3.2, this technique performs uniformly better compared to simple regression modeling, yet with not satisfying accuracy. Finally, we use a technique that combines regression and tree, which is referred to as "Regression Tree Modeling" to estimate the outputs with acceptable accuracy. To obtain this model, we also consider the nonlinear relationship between predictor variables and outputs. At the end, we employ a standard technique to evaluate the accuracy and effectiveness of the proposed model.

For predicting delay and energy, we gather a large data set from different alternative configurations that are possible for designing an ONoC system. The approach for gathering the alternative configurations from all "design space" will be discussed in detail in chapter 4. A single configuration consists of many variables that indicate the behavior of the designed system. In this thesis, we are interested to know about system behavior in two aspects: delay and energy of the system. We consider delay and energy as our desired outputs. By delay, we mean packet latency and contention delay of the network, and by energy, we mean static energy and dynamic energy consumption of network. Details about the definition of these four outputs are available in section 0. The variables that affect the outputs are used as inputs to predict the value of the observed outputs. In the literature, these input variables are referred to using multiple names such as "predictors", "attributes" and "features". From now on, in this thesis, we will use "features" to refer to these input variables. One important step for obtaining a representative model that can

predict outputs with high accuracy is feature selection. We should select features with significant impact on outputs. In section 3.7 we will discuss feature selection.

3.1. Linear Regression

In general, features can have numeric values or nominal values. In our dataset, some of the features have numeric values like "number of cores" (e.g. values 4, 16, 64…), and some of them have nominal values like "routing strategy" (e.g. "distance-based", "cluster-based"). The outputs have numeric values too. Our first two outputs are "average packet latency" and "average contention delay" had been reported in a nanosecond, and second two outputs "static energy" and "dynamic energy" had been reported in joule. One of the classic ways of solving this kind of problems that deal with continuous prediction (i.e. predicting an output that has continuous value) is to write the output as a linear sum of the features values with appropriate weights. This is called a "regression equation" that has been explained in Equation 2 in section 3.1.1, and the process of determining the weights in this equation is called "regression", and it is a well-known procedure in statistics [34].

In Figure 7 we display a general dataset with "n" of inputs and an output. Each configuration in design space includes several features that affect the value of output. By simulating thousands of experiments with different configurations we create a dataset. This dataset gathered "m" numbers of different sampled configurations. Each configuration consists of "n" number of features as inputs (i.e. $f_1, f_2, \ldots, f_n$) and one output (i.e. y). A subscript in f indicates feature number and a superscript indicates row number in the dataset (i.e. a sample configuration number). For example, $f_2^{(3)}$ indicates feature number two in sample configuration number three. $f_1^{(3)}$, $f_2^{(3)}$, …, $f_n^{(3)}$ are all features in sample configuration number three and $y^{(3)}$ is output for this sample.

Features				output
f_1	f_2	...	f_n	y
$f_1^{(1)}$	$f_2^{(1)}$	...	$f_n^{(1)}$	$y^{(1)}$
$f_1^{(2)}$	$f_2^{(2)}$	...	$f_n^{(3)}$	$y^{(2)}$
$f_1^{(3)}$	$f_2^{(3)}$	...	$f_n^{(3)}$	$y^{(3)}$
...	...	...	...	...
$f_1^{(m)}$	$f_2^{(m)}$	...	$f_n^{(m)}$	$y^{(m)}$

m = number of gathered configurations

n = number of features

Figure 7: Dataset

3.1.1. Model representation and formulation

Linear regression is a basic method in the statistic that is usually used when the output has a numeric value, and all the features have numeric values. However, some of the features in this work have nominal values. To convert features with nominal values into features with numeric values we employ "one hot encoding" technique. For example, originally "routing strategy" can be "cluster-based" or "distance-based". When we employ "one hot encoding" the "routing strategy" is (10) for the value "cluster-based", and (01) for "distance-based". Generally, a nominal feature with k values is converted into k binary features.

The objective of this model is to express the output as a linear combination of the features, with predetermined weights:

$$h_w(f) = w_0 + w_1 f_1 + w_2 f_2 + \cdots + w_k f_k$$

Equation 2: Regression Equation

where $h_w(f)$ is hypothesis function that will predict output value; f_1, f_2, …, f_k are features values; and w_0, w_1, …, w_k are weights.

There are several algorithms that can be used to find the weight values. One of them is the gradient decent algorithm that will be discussed in section 3.1.3. After applying this algorithm, the result is a set that has weight values. If the data exhibits a nonlinear dependency, the best-

fitting straight line should be found, where "best" is interpreted as the least mean-squared error [34]. Figure 8 shows the regression procedure. It starts with gathering a dataset from our system; then we indicate features that affect the output of the system. By applying a learning algorithm to the dataset based on selected features, we will find a hypothesis function that can estimate the output of the system.

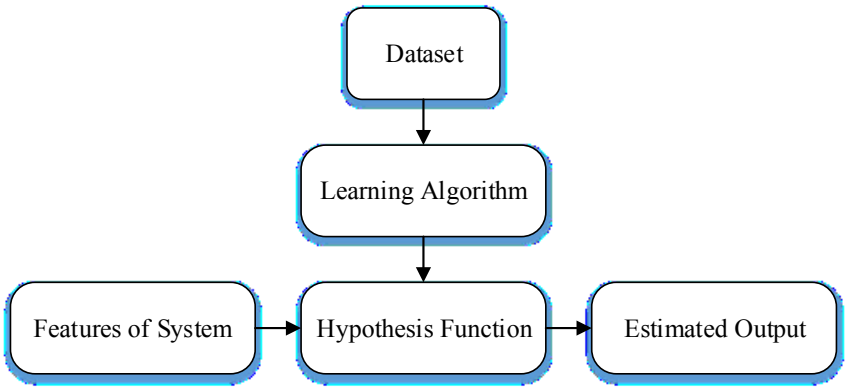


Figure 8: Regression Procedure

3.1.2. Cost function

In Equation 2, a hypothesis function $h_w(f)$ may have different value for different given weights (i.e. w_i for $i = 0, 1, \dots, k$). The objective in this step is to choose the value for w_i so that $h_w(f)$ is close to y, where y is actual value of output in dataset. In order to do so, we define the following equation:

$$j(w) = \sum_{i=1}^{m} (h_w(f^i) - y^i)^2$$

Equation 3: Cost Function

Where $j(w)$ is cost function and f^i indicates all features in row i in dataset and y^i is actual value for output in row i in the dataset. The cost function is a function of weights "w", and hypothesis function ($h_w(f)$) is a function of features "f" for fixed weight w. Different value of w's gives different value of $h_w(f)$. Then we calculate difference between $h_w(f)$ and y which is $j(w)$. The objective is to minimize the value of $j(w)$. We should choose best w that give minimum value for $j(w)$. It means $h_w(f)$ is the best fit for y in this particular value of w.

3.1.3. Gradient Descent

Gradient descent is a general algorithm for minimizing a function. We will use gradient decent algorithm to minimize the function value for a given function $j(w)$, and therefore to find the best fit to the actual data value. In order to employ this algorithm, we start with a random value for weights (w) and we keep changing weights to reduce $j(w)$ until we end up at a minimum point of $j(w)$. However, the Algorithm may not converge to the minimum and even diverge. It depends to alpha (learning rate). I explain it in next paragraph, and Figure 9 part a. The pseudocode of gradient descent algorithm used to change a value of each weight w is presented in Listing 1.

```
repeat until convergence {
        w_j := w_j − α ∂/∂ w_j j(w)
}
```

Listing 1: Gradient Descent Algorithm

Where α is learning rate and $\frac{\partial}{\partial w_j} j(w)$ is derivative. Learning rate will be used to indicate the size of the step needed to be taken for changing w's. If α is too small, gradient decent can be slow to converge and if α is too large gradient decent can overshoot the minimum. It also may fail to converge or even diverge with large value of α like the case in Figure 9 part a. As we approach a local minimum point, gradient decent will automatically take smaller steps. Therefore, there is no need to decrease α over the time. The reason is that $\frac{\partial}{\partial w_j} j(w)$ become smaller over a time and at local minimum point it is equal to zero.

Derivate $\frac{\partial}{\partial w_j} j(w)$ is the slope of the line that is a tangent to the function. With positive slope the derivative causes the value of w decrease like the case in Figure 9 part b, and with negative slope it causes the value of w increase like in Figure 9 part c.

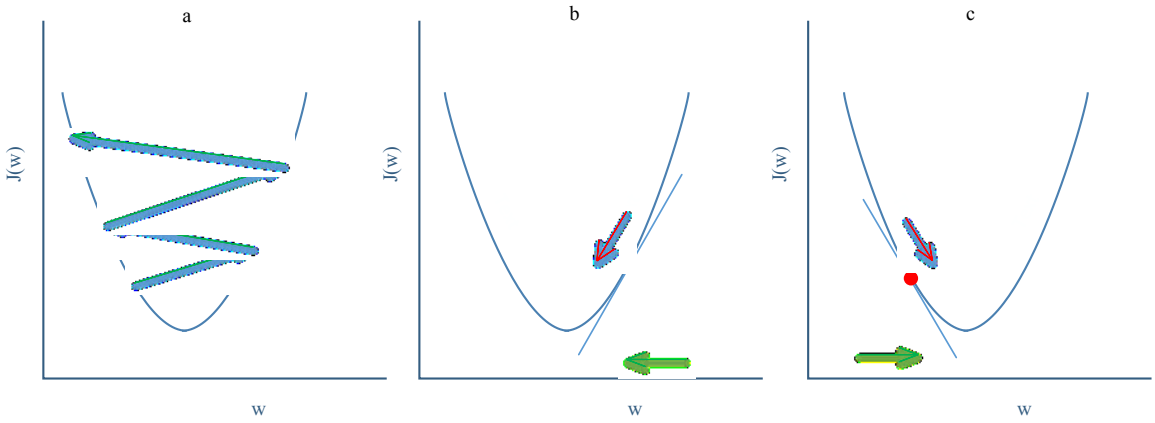


Figure 9: a) Case of divergence, b) Positive slope of tangent, c) Negative slope of tangent

Depending on the initial point that has been set for the gradient decent algorithm, it may reach to the different local minimum point. However, in linear regression and with the gradient decent algorithm, the cost function is a convex quadratic function. Therefore it has only one global optimum point. In general, for example, the regression problems that use higher order polynomials, it is possible that gradient decent get stuck in local minima, and never arrive at an absolute minimum.

3.2. Trees

3.2.1. Decision Tree

Decision tree builds a regression or classification model in the form of a tree structure. It uses a "divide-and-conquer" approach to learn a model from a dataset. It breaks down the dataset into smaller subsets and develops a tree structure. The final result is a tree with leaf nodes. Leaf node represents a "decision" on data that reach to this leaf. In this thesis, the term "decision" refers to the predicted value for output. A decision tree can handle both nominal and numeric data. When the output has nominal value the decision tree called "classification tree". In classification tree, each leaf decides about the class of the instance that reach to this leaf. For example, if the tree wants to decide about the class of a particular picture and determine if it is a picture of a car or a picture of a bicycle. When the output has numeric value the decision tree called "regression tree". In regression tree, each leaf decides about the value of the output (predicted value of

output) for instances that reach to this leaf. For example, if the tree wants to determine the value of energy consumption for a particular configuration that feeds to it and now reach to the specific leaf. For classifying a new data, it is routed down the tree according to the values of the features tested in previous nodes, and when the data reaches to the leaf, it will be classified according to the class assigned to the leaf. A functional tree can have a linear model at the leaf nodes, which are used for predicting the output. [34]

3.2.2. Decision Tree Algorithm

The decision tree algorithm works with "standard deviation reduction" technique. It is based on the decrease in standard deviation after a dataset is split on a particular feature that nominated for splitting the tree. Before presenting the definition of standard deviation reduction, we want to define a few terms that we will use in the explanation. These terms are means, variance and standard deviation of n numbers having values x_i, $i=0, 1, 2 \ldots n-1$.

- Mean: this is the arithmetic average of values that refer to a central tendency value of a numeric set. It is the sum of the values divided by the number of values: $\mu = \frac{\sum x}{n}$
- Variance: It measures how far a set of numbers are spread out from their mean. It subtracts the mean from each value, squares the result, adds them together, and then divide by the number of values: $\sigma^2 = \frac{\sum(x-\mu)^2}{n}$
- Standard deviation: It this is the square root of the variance. It also measures the spread of data from mean like variance, but it has the same unit as original data because of square root function: $S = \sigma = \sqrt{\frac{\sum(x-\mu)^2}{n}}$

For explaining the decision tree algorithm, we use an example to clarify the concept. In Table 1, we have a dataset for our example. It has four features: Distance with values {4, 8, 16}, Cluster size with values {8, 16, 32}, Routing strategy with values {D, C} and number of cores with values {64, 256} and one output that is static energy. Although we come up with imagery numbers for static energy, the example can show the required steps for building a decision tree very well. This algorithm was first presented at [35].

Table 1: Dataset for Decision Tree Example

Distance	Cluster Size	Routing Strategy	#Cores	Static Energy
4	8	D	64	25
4	8	D	256	30
8	8	D	64	46
16	16	D	64	45
16	32	C	64	52
16	32	C	256	23
8	32	C	256	43
4	16	D	64	35
4	32	C	64	38
16	16	C	64	46
4	16	C	256	48
8	16	D	256	52
8	8	C	64	44
16	16	C	256	30

As the first step of this algorithm we should calculate the Standard deviation of the output:

$$S(output) = S(static\ energy) = 9.32$$

In the second step, we should calculate the standard deviation of output for each feature separately. Boxes 1-1 to 1-4 show the standard deviation for each value of each feature that we have in our dataset (e.g. in the first following table we find standard deviation of outputs that have distance 8, 4 and 16 as their first feature in dataset, but different values for all other features)

Box 1- 1

		Standard deviation of static energy
Distance	8	3.49
	4	7.78
	16	10.87

Box 1- 2

		Standard deviation of static energy
Cluster size	32	10.51
	8	8.95
	16	7.65

Box 1- 3

		Standard deviation of static energy
Routing Strategy	D	9.36
	C	8.37

Box 1- 4

		Standard deviation of static energy
#Cores	64	7.87
	256	10.59

Now that we have all the standard deviation values of each value in each feature we should calculate the standard deviation for each feature in overall. For example, if we want to calculate the standard deviation for feature "distance" that has three possible values {4, 8, 16} we should use the following equation:

$$S(output, feature) = \sum_{i \in feature} P(i)S(i)$$

Equation 4: Standard deviation of multiple values

Where $P(i)$ is the probability that i appears as the value of feature. It means we should count how many times a particular value of feature (here i) have been seen in entire dataset and then divide it by total size of dataset.

In our example, if we want to calculate the standard deviation for feature "distance" we will have:

		Standard deviation of static energy	Count
Distance	8	3.49	4
	4	7.78	5
	16	10.87	5

$$S(static\ energy, distance) = P(8) * S(8) + P(4) * S(4) + P(16) * S(16)$$
$$= \left(\frac{4}{14}\right) * 3.49 + \left(\frac{5}{14}\right) * 7.78 + \left(\frac{5}{14}\right) * 10.87 = 7.66$$

Constructing a decision tree is all about finding a feature that returns the highest standard deviation reduction in output. Standard deviation reduction for a feature is obtained when we subtract "standard deviation of output after splitting data based on that particular feature" from "original standard deviation of output" (before any splitting is done):

$$SDR(output, feature) = S(output) - S(output, feature)$$

Equation 5: Standard Deviation Reduction (SDR)

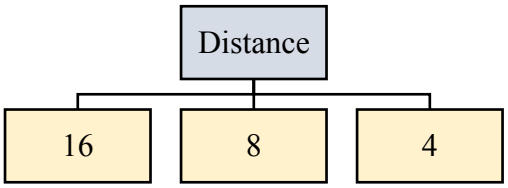
Now we should calculate standard deviation reduction (SDR) for each feature:

$$SDR(static\ energy, distance) = S(static\ energy) - s(static\ energy, distance)$$
$$= 9.32 - 7.66 = 1.66$$

SDR for other features are:

$$SDR(static\ energy, clsuter) = 0.17$$
$$SDR(static\ energy, Routing\ Strategy) = 0.28$$
$$SDR(static\ energy, cores) = 0.29$$

The feature that has largest standard deviation reduction will be chosen as the decision node. In our example, the largest SDR is for feature "distance". Therefore we will choose feature "distance" as the root node and divide dataset based on the values of this feature:



This is our dataset after dividing based on feature “distance”:

Distance	Cluster Size	Routing Strategy	Cores	Static Energy
4	8	D	64	25
4	8	D	256	30
4	16	D	64	35
4	32	C	64	38
4	16	C	256	48
8	8	D	64	46
8	32	C	256	43
8	16	D	256	52
8	8	C	64	44
16	16	D	64	45
16	32	C	64	52
16	32	C	256	23
16	16	C	64	46
16	16	C	256	30

Now we should apply the same procedure to the each node and all of its branches again. A branch that is having a higher standard deviation than a certain threshold requires further splitting. The process is repeated recursively on the non-leaf branches until all data is processed. When the number of data is more than one at a leaf node, the average value of data on that node is considered as the final value for the predicted output. For example, if we choose node “16” for “distance” feature and calculate the standard deviation for static energy again we will have:

16	16	D	64	45
16	32	C	64	52
16	32	C	256	23
16	16	C	64	46
16	16	C	256	30
Standard deviation				10.87

Then we need to calculate the standard deviation of each value for all the other feature and finally, calculate SDR for each feature as follow:

		Standard deviation of static energy
Cluster size	32	14.50
	8	Not appear in this portion of dataset
	16	7.32

$$SDR(static\ energy, clsuter) = 10.87 - \left(\left(\frac{2}{5}\right) * 14.5 + \left(\frac{3}{5}\right) * 7.32\right) = 0.678$$

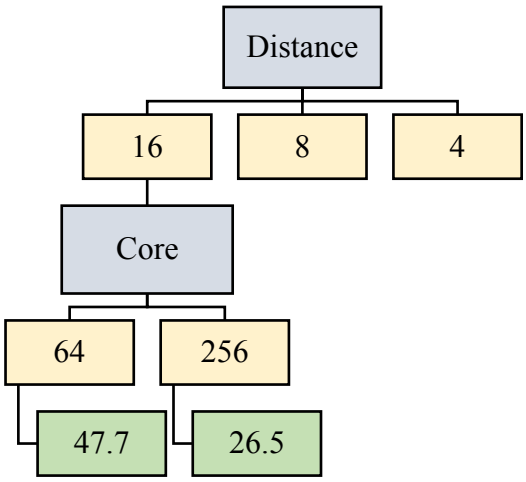
		Standard deviation of static energy
Routing Strategy	Dis	7.50
	Clu	12.50

$$SDR(static\ energy, Routing\ Strategy) = 10.87 - \left(\left(\frac{2}{5}\right) * 7.5 + \left(\frac{3}{5}\right) * 12.5\right) = 0.370$$

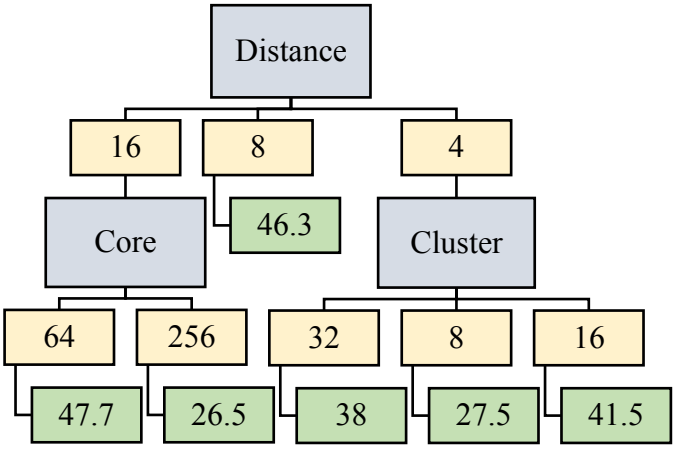
		Standard deviation of static energy
Cores	64	3.09
	256	3.50

$$SDR(static\ energy, cores) = 10.87 - \left(\left(\frac{3}{5}\right) * 3.09 + \left(\frac{2}{5}\right) * 3.5\right) = 7.62$$

Based on the SDR, we will choose feature “number of cores” because it has higher standard deviation reduction.



By repeating these steps, the final tree will be:



Listing 2 displays pseudo code of decision tree algorithm explained above.

```
const integer number_of_features = n
const integer number_of_samples = m
declare feature [number_of_features]
declare dataset[number_of_features][number_of_samples]

StDev_y = standard_deviation (y)
while (StDev_y > threshold){
    for index = 0 to n {
        max_StDev_reduction = 0
        split(dataset,feature[index])
        StDev_branch = standard_deviation (y)
        StDev_reduction = StDev_y – StDev_branch
        if (StDev_reduction > max_StDev_reduction){
             max_StDev_reduction = StDev_reduction
             decision_node = feature[index]
        }
    }
    split(dataset, decision_node)
    StDev_y = standard_deviation (y)
}
```

Listing 2: Decision Tree Algorithm

3.3. Regression Trees and Model Trees

Regression Tree and Model tree are very similar to decision tree with a few changes. Both of them use for numeric output prediction. In “Regression Tree” the leaf nodes give average values of data that reach to the leaf as predicted value. In “Model Tree” instead of calculating the average of data, the leaf nodes give a linear regression model of the data that reach to the leaf. It means that Model Tree consists of several linear regression models that placed in each leaf of the

tree and formulate the linear models of data that reach to that specific leaf. In both cases, the features may have nominal or numeric value.

Regression and model trees are initially constructed by using a same decision tree algorithm. The general idea is to fit a regression model to the output using each of the features. Therefore the dataset is split at each feature like what we explained in the previous section. These trees also calculate the prediction "error" before and after splitting. It means that for splitting on each feature the "error" between the predicted value and the actual values calculated and then they are added together and squared to get a "Sum of Squared Errors" (SSE). The split errors for the features are compared, and the feature with the lowest SSE is chosen as the root node. This process is recursively continued until all features process. The only difference between regression tree and model tree is that, for the model tree, each leaf is replaced by a regression model of data reach to that leaf node, instead of a constant value that is an average of data reach to that leaf node. [34]

In our model, we use two types of trees. One is REPTree, and another one is M5P. REPTree is Reduced Error Pruning Tree which is a fast learning decision tree that builds a decision tree based on the variance reduction technique (very similar to decision tree). Once the basic tree has been formed, consideration is given to pruning the tree back from each leaf. Pruning the tree with this algorithm makes a better prediction because it only prunes a branch if this pruning reduces the prediction error. The primary pruning operation is "subtree replacement". The idea is to select a subtree and replace it with a single leaf. For knowing when we should replace a subtree with a leaf, we consider a different dataset and then estimate the error rate that would be expected if we do the replacement in a tree on that dataset. Finally, by comparing the estimated error before and after that particular replacement we can decide whether we should do the pruning or not [34].

The M5P is a reconstruction of Quinlan's M5 algorithm [36] for building a tree of the regression model. M5P combines a conventional decision tree with the possibility of linear regression functions at the nodes. In this way, it splits the dataset to many smaller sets and then assigns a linear regression model to each set. Therefore the overall prediction error will be reduced because the linear model can fit better in each subset of the original dataset. We will use

the implementation of these algorithms as provided by Weka [34]. In section 4.3.1 we will explain about Weka tools.

3.4. Non Linearity

Linear regression analysis is not always a good model for finding a relation between input variables and output because some problems do not have a linear relation between features and output and our problem is one of them as we will see in section 4.3.4. To obtain a better model, we also consider the nonlinear relationship between predictor variables and outputs. Statistical regression analysis is the study of techniques that relate one dependent variable (i.e. output) to some independent variables (i.e. features). In Equation 6 f_i ($i = 0, \ldots n$) terms are the independent variables. Their changing values cause the dependent variable, $h_w(f)$, to vary as a response.

$$h_w(f) = w_0 + w_1 f_1 + w_2 f_2 + \cdots + w_k f_k$$

Equation 6: Regression Equation

Variables in real problems often have nonlinear dependence that cannot be accurately captured by Equation 2. To handle these cases, transformation functions (F_l and F_n in Equation 7) are applied tof_i. Equation 7 is still a linear regression model because $h_w(f)$ is a function of f with weight of w_i and it is still linear. As a result, techniques in linear regression theory can still be applied to the problem while nonlinearity in independent variables is handled.

$$h_w(f) = w_0 + w_1 f_1 + w_2 f_2 + \cdots + w_k f_k + w_l F_l(f_l) + \cdots w_n F_n(f_n)$$

Equation 7: Linear Regression for handling Nonlinearity

3.5. Feature Transformation

Plotting each of the features against the output may reveal particularly strong associations or identify non-linearity. In section 4.3.4 we will present some of these plots that help us to find transformation needed for each feature as a predictor. As an example after plotting features against average packet latency as an output (section 4.3.4, Figure 14, 15 and 16), these are the relationships between features and output extracted from plots:

For the feature f_1 *"number of receive networks per cluster"* we apply an inverse transformation function on it and it creates a new feature $\frac{1}{f_1}$ where f_1 is an independent variable representing *"number of receive networks per cluster"*. For the feature *"number of access points"* we apply an inverse transformation function on it and it creates a new feature $\frac{-1}{f_2}$ where f_2 is an independent variable *"number of access points"*. For the feature f_3 *"cluster size"* we apply a square root transformation function on it and it creates new feature $\sqrt{f_3}$ that f_3 is independent variable *"cluster size"*.

3.6. Evaluation of fitting

To have a fair evaluation of a model, we need to get a new set of data to test the formulated model learned from the dataset. It is because we want to avoid possible biased estimation that can occur if we use the same dataset for both learning process and testing. Therefore, we will split a full dataset to two separate sets. One is *"training set"* that is used as an input for formulating the model (i.e. for learning). Another set is *"test set"* that is used for evaluating the performance of the model. In this way, the estimation will be more reliable with fair and unbiased accuracy. Training set and test set (i.e. the entire dataset) are generated by simulation of many design alternatives; Section 4.2 will explain about dataset generation in details. And in section 5.1 the experiment results and validation method for testing predicted results are presented.

After formulating a model, we predict values for output based on this model. Predicted output values may not fit the actual value of output in the test set very well. In this case, we have a *underfitting* problem (Figure 10 part a). If we have too many features the formulated model (i.e. corresponding hypothesis function, in case of regression modeling) may fit the training set too well i.e. $j(w) = 0$. But it fails to generalize to new examples (i.e predicting delay and energy for new example). This problem named *overfitting* (Figure 10 part b).

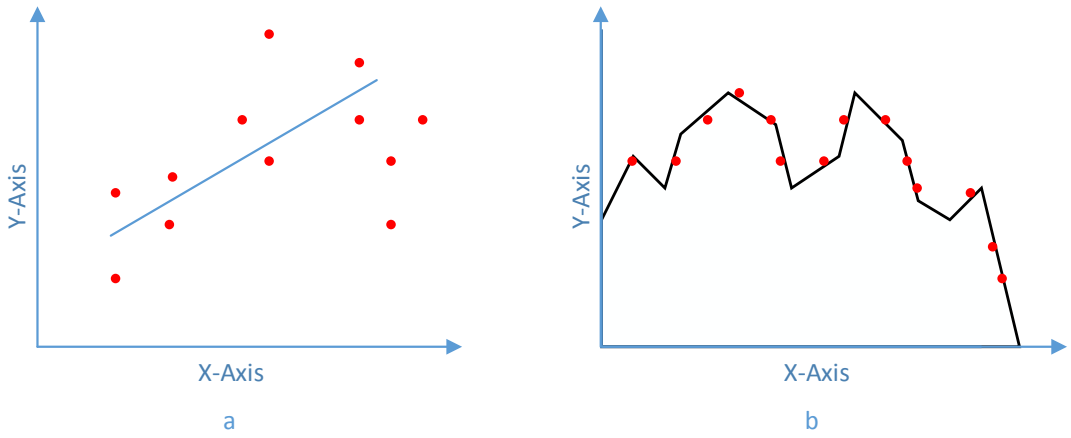


Figure 10: a) Underfitting, b) Overfitting

Overfitting usually occurs when we have so many features and few training data. For addressing overfitting, we can try one of the following options. One solution is reducing the number of features. We should decide which feature we should keep and which one should throw out. Another solution is *Regularization*. In this method we keep all the features, but reduce the magnitude of weight parameters w[34]. This method works well when we have a lot of features and each of them contributes a bit to prediction of output.

3.7. Feature selection

The feature selection is critical for obtaining a model with high accuracy levels. It is important because it can lead us to a successfully and meaningfully modeling of the problem or it can mislead the modeling. Moreover, redundant features can be misleading to modeling algorithms. Keeping irrelevant features in the dataset can result in overfitting. For example, decision tree algorithms attempt making an optimal split of the feature values. Those features that are more correlated with the prediction will split first (the ones that have bigger standard deviation) Deeper in the tree, less relevant and irrelevant feature are used to make prediction decisions. These less relevant or irrelevant features may only be beneficial a little or by chance in the training dataset. This is how we would have overfitting in the case of decision trees. This overfitting of the training data can negatively affect the prediction modeling and cause lower prediction accuracy. Therefore, it is important to remove redundant and irrelevant features from the dataset before evaluating algorithms. This task should be tackled in the step of preprocessing

data, which is one of the primary steps in the machine learning process. In this step, we can also define new feature by finding an appropriate transformation function (like Equation 7) or by combining two or more other features together. We will discuss this step in detail in section 4.3.3 and 4.3.4.

CHAPTER 4

4. Design Space Exploration Technique

In this chapter, we present our methodology for exploring design alternatives and deriving a representative (i.e. accurate) model for delay and energy prediction. For developing a prediction model that can predict delay and energy for a given configuration set in our design space, first of all, we need to gather a dataset from simulating many experiments with different design configurations. This dataset will be our input to a statistical approach that is used for prediction modeling. For the purpose of this thesis, generating the dataset for applying learning algorithms took almost four months. However, in general, the time needed for gathering the dataset can be more or less, and it depends on a number of simulated experiments, the simulator itself and the machine specification that runs the simulator. Nevertheless, the simulation of all possible configurations for an ONoC is an extremely time-consuming process, and efficient exploration of the design space only by using simulation is practically impossible given a tight time to market constraint. Therefore we propose a design space exploration technique that enables efficient evaluation of design alternatives. In this chapter we explicitly cover the exploration technique and its required steps that we took: the procedure for gathering the dataset, processing the dataset, prediction modeling, introducing the simulator framework, the benchmark applications, and statistical analysis tool. And also we discuss configuration parameters and sampling from different configurations among the whole design space. We start with the overview of the proposed technique and follow with the detailed explanation of its steps. As Figure 11 shows, the first step is providing a dataset for our prediction. To do so, we run 13320 simulations with different design configurations of ONoC. Therefore, in the first section of this chapter we talk about generating needed dataset. In this section, we also cover the configuration setting, simulation framework and benchmark applications used for generating the dataset. The second section will talk about data preprocessing, and it includes a statistical analysis of the dataset and needed modifications of the dataset before applying learning algorithm. Data preprocessing consists of feature selection and feature transformation. The last part of this methodology is about building a prediction model based on modified dataset.

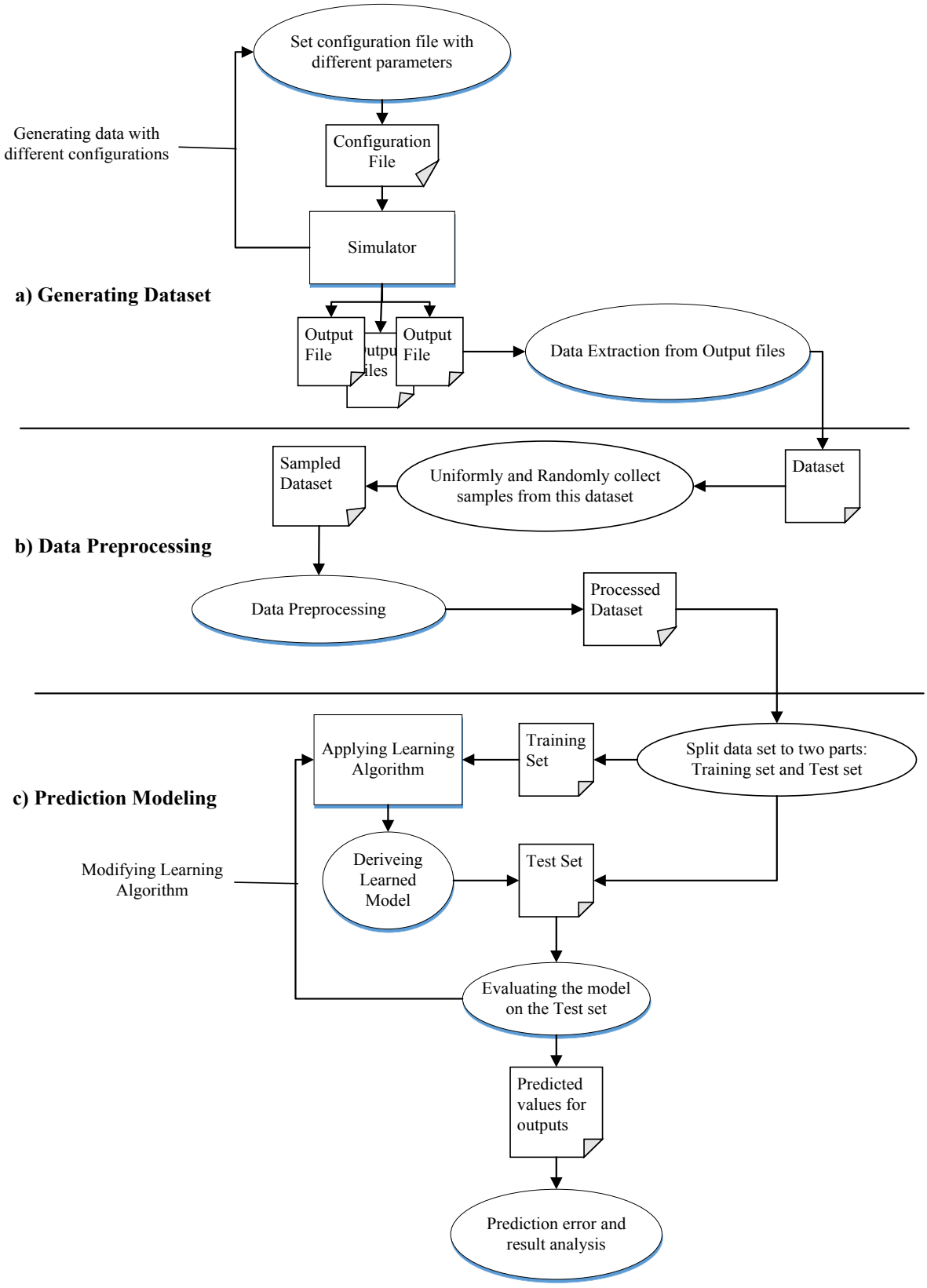


Figure 11: Overview of Methodology

4.1. Overview of Design Space Exploration Technique

The overview of the proposed design space technique is shown in Figure 11. The oval steps show our contributions that are required tasks for design space exploration. They are generating a comprehensive dataset by simulating numerous design alternatives, sampling a smaller dataset from the original one, preprocessing the dataset for having several accurate prediction models, building and evaluating our prediction models and finally analyzing them. The square shapes are the steps taken by existing tools that we will introduce in this chapter. Three main steps are required for applying this technique: a) Generating Dataset, b) Data Preprocessing and c) Prediction Modeling. Part a, generating dataset consists of simulating different ONoC configurations to build a dataset. Part b, the data preprocessing consists of sampling from original dataset and then preparing the dataset to find the prediction model in next step. Part c, the prediction modeling consists of finding the prediction model and evaluating it. In next three sections, we talk about each step in details.

4.2. Generating the Dataset

As shown in Figure 11 part a, the first step in our technique is generating the dataset. Different design configurations of ONoC have different delay and energy consumption. For seeing how delay and energy change with different configurations, we vary the values of the selected parameters to obtain both the configurations and the corresponding delay and energy. For doing so, we wrote a script to go through different configurations and then we used an open source simulator to run these different configurations on different applications. For the purpose of simulation, each configuration has its own "configuration file" as an input that specifies the values of all the parameters needed for the simulations. So in this case, features are configuration parameters that we changed in the "configuration file". The output of this step is a dataset that will be used for the second step which is data preprocessing. Table 2 shows one row from our dataset. The dataset consists of feature values: number of cores, cluster size, number of access points, number of receive networks, laser type, routing strategy and distance, the output values are: average packet latency in nanosecond, average contention delay in nanosecond, static energy in Joule and dynamic energy in Joule.

Table 2: One row of Dataset

Features							Outputs			
f_1	f_2	f_3	f_4	f_5	f_6	f_7	y_1	y_2	y_3	y_4
#Cores	Cluster size	#Access points	#Receive networks	Laser Type	Routing Strategy	Distance	Avg Packet Latency(ns)	Avg Contention Delay(ns)	Static Energy(J)	Dynamic Energy (J)
64	4	1	2	Standard	Distancebase	4	14.797631	1.223670	0.068701	0.000826

4.2.1. Configuration setting

The approach to obtain a dataset from a large ONoC design space is critical to the efficient formulation of prediction models. Table 3 identifies seven features varied in each design configuration, f_1 , . . ., f_7. The range of values considered for each feature is specified by a set of values. The Cartesian Product (*CP*) of these sets which is $= \prod_{i=1}^{7} f_i$, defines the entire design space. Although the range of each feature displays the values that it can get, not all the combinations of the feature values are possible. For example, if we select 8 for the value of "number of access points" in our configuration, the "cluster size" needs to be equal or greater than 8. This is because an access point has to be within a core (to be precise a tile), and the cluster size specifies the number of cores per cluster. Therefore, in order to have 8 access points, we need at least 8 cores per cluster. So all the configurations from the design space that has "number of cores" equal to 8, and the "cluster size" less than 8 are not physically capable to get 8 number of access points, and need to be excluded from the design space. This is the specific limitation that we should consider in the configuration:

$$cluster\ size \geq number\ of\ RecNW \geq number\ of\ AccP$$

Where *cluster size* is the number of cores in a cluster, *number of RecNW* is the number of receiving networks that are responsible for forwarding data from hub to cores in receiving cluster. We use StarNet for our receiving network (see section 0 for more details). And the last one is $number\ of\ AccP$ that indicates number of optical access points per cluster. Not all the tiles in a cluster connect directly to the optical hub of the cluster. In each cluster there is one tile or more that has access point to the optical hub (and so ONet). If the tile is not an access point itself, it should route packet to the nearest access point in its cluster to send packet over ONet. More details about these features are available in section 0.

Moreover, as we said in section 2.4, there is two type of routing strategies for delivering a packet: distance-based and cluster-based. We indicate the routing strategy in feature six. The feature "Distance" refers to distance threshold that used in distance-based routing strategy. For transitions over that distance threshold, packets use ONet. Otherwise, it goes through ENet. However, in cluster-based routing strategy, we do not need this threshold. In this case, we always send a packet over ONet, if source and destination of the packet are not in the same cluster. Otherwise, we send it through ENet. Therefore feature "Distance" is only used for configurations with distance-based routing strategy.

In ATAC ONoC network model, the number of cores should always be perfect squared and power of two. Also, cluster size must be a power of two and number of application tiles should be a multiple of cluster size. This kind of limitations makes the possible design space smaller than what we calculated in *CP*. Based on the range of each feature in Table 3, the cardinality of *CP* is equal to16384. However, the feasible design points that we can have for each application is 4440. Since we employ three different applications, the number of design space points that we simulate in this thesis raise to a total of 13320.

Table 3: Changing Features in Simulations

	Feature's Name	**Measure**	**Range**	$\|f_i\|$	**Explanation**
f_1	Core	Count	64,256	2	Number of Cores
f_2	Cluster	Count	1,2,4,8,16,32,64,128	8	Cluster Size
f_3	AccP	Count	1,2,4,8,16,32,64,128	8	Number of Optical access point per cluster
f_4	RecNW	Count	1,2,4,8,16,32,64,128	8	Number of Receive Network per Cluster
f_5	Laser	Type	Throttled, Standard	2	Laser Type
f_6	Routing	Type	Distance-based, Cluster-based	2	Routing strategy
f_7	Distance	Count	2,4,8,16	4	Distance in distance-based routing strategy

Beside these features listed in Table 3 that varied in different configurations, there are also other parameters that do not change during simulation and have fixed value in all the simulated configurations. These parameters and their values are displayed in Table 4. These are network parameters that remain fixed during all simulations. Note that explanation of these parameters is in section 0.

Table 4: Fixed parameters in configuration of ONoC

Parameter Name	Fix value
Technology node	45 nanometers
Temperature	300 Kelvin
Tile width	1 millimeter
Network model	ATAC
Received Network Type	star

There are also other parameters related to memory configurations that we keep fixed. Here is caches configuration for our simulations:

L1 cache configuration:

Cache line size	64 Bytes
Cache size	16 KB
Associativity	4
Replacement policy	LRU

L2 cache configuration:

Cache line size	64 Bytes
Cache size	512 KB
Associativity	8
Replacement policy	LRU

4.2.2. Simulation Framework

As we explained in section 3.1, our outputs are delay (i.e. packet latency and contention delay) and energy (i.e. static and dynamic energy), and we evaluate delay in nanosecond or clock cycle (because the clock frequency is 1GHz, one clock cycle takes one nanosecond) and energy in Joules. For obtaining these outputs, we used a simulator called *"Graphite"* [13]. Although we used this simulator for gathering the dataset, we do not use any particular feature of the simulator

in our models and believe our approach may generally be applied to any other simulator framework.

Graphite [13] is a distributed, parallel simulator for design space exploration of multicores research. It is an open-source software simulator that can be downloaded from [37]. Graphite models function and performance of following system components: core, on-chip network, and memory. Due to the modular design of Graphite, each module in Graphite can be replaced with an alternative design. Therefore by using a configuration file a designer can simulate different architectures to study its performance and accuracy under Graphite. By using Graphite, we run actual applications on our desire architecture and accurately capture the interplay between applications, multicore hardware architecture, and the underlying electronic and photonic devices. It performs a performance, power and area analysis for an optical multicores architecture like ATAC or ORNoC. Figure 12 shows Graphite architecture. Application threads are mapped to tiles (cores) of the target architecture. These cores are distributed among host processors, and these processors can be distributed to multiple host machines. Scheduling and execution of these threads is the responsibility of host operating system. Threads are automatically distributed by Pin (Intel's dynamic binary translator [38]) that traps application events.

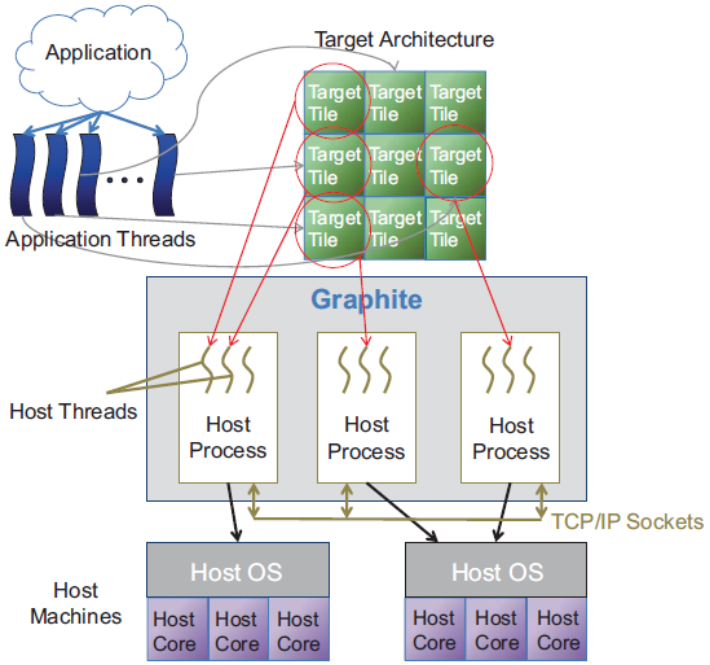


Figure 12: Graphite High-level Architecture [13]

As Figure 13 shows, each tile of target architecture consists of three main components: processor core, network switch and a part of the memory subsystem. Implemented models for each of these components are swappable and can be replaced with an alternative model to explore different architectures. Our focus in this thesis is on the network model. Therefore we keep fixed models for core and memory and just change parameters of the network model in our design space exploration. The network model is responsible for routing the packet over the on-chip network and calculate various delays and energy consumptions on the network.

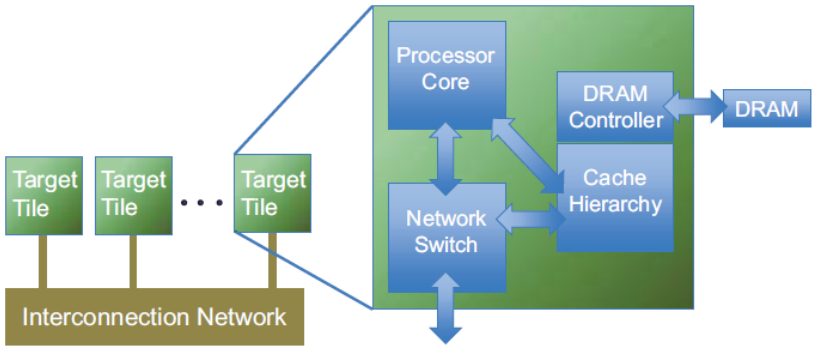


Figure 13: Tile Architecture [13]

Each tile has its individual local clock. Events from core, network and memory subsystem update individual local clocks in each tile. Therefore it is necessary to synchronize clocks for managing the skew in different local clocks. Moreover, threads may run at different speeds, and it causes clocks to deviate. Graphite supports three synchronization schemes that offer different accuracy and performance tradeoffs. They are Lax Synchronization, Lax with Point-to-point Synchronization and Lax with Barrier synchronization. Among these synchronization schemes that Graphite supports, Lax synchronization offers the best performance, and worst accuracy, Lax with Point-to-point Synchronization offers good performance and good accuracy and Lax with Barrier synchronization offers worst performance and best accuracy. In this thesis, we use Lax with Barrier synchronization because it has the best accuracy for timing, and we will briefly explain it here. However, more explanations about the other options are available in [13]. In Lax with Barrier synchronization, all active threads wait on a barrier after a configurable number of cycles (in this thesis it set to 1000 nanoseconds or cycles). If we wait on barrier frequently it can keep cores tightly synchronized and therefore imitates a cycle accurate simulation. Nevertheless, Graphite is not a complete cycle accurate simulator, by supporting several synchronization

strategies graphite present different timing accuracy and performance. Although cycle-accurate simulators provide extremely accurate results, the overhead required for such detailed modeling leads to very slow execution (typically between 1000x to 100,000x slowdown). The results in [13] demonstrate that Graphite provides an accurate estimate of performance and achieves slowdowns as little as 41x over native execution for simulations of SPLASH-2 benchmarks on a 32-tile target.

4.2.3. Benchmarks

For testing the design that we configured for ONoC architecture, we need applications that have significance traffic to evaluate network model. The user can use any application that is suitable for their work, and our methodology can generalize to their application too. We use three application benchmarks from Splach-2 benchmark suite [39] for this purpose. Splash-2 benchmark suite is a set of real application programs for comparing the performance of parallel systems. The selected benchmarks are:

- "Radix": a benchmark with high rates of unicast traffic and low rate of broadcast
- "Barnes": a benchmark with low rate of unicast traffic and high rate of broadcast
- "Ocean": a benchmark with mediator unicast and broadcast traffic

Radix: Radix implements an integer radix sort. The integer radix sort is based on the method described in [39]. The algorithm is iterative, performing one iteration for each radix r digit of the keys. This permutation step requires all-to-all communication, and it causes a lot of unicast traffic between each processor to the other ones. The time complexity of the Radix sort is $O(n)$ [40].

Barnes: Barnes implements the Barnes-Hut hierarchical N-body method to simulate the interaction of a system of bodies (galaxies or particles, for example) in three dimensions over a number of time-steps. The time complexity of the Barnes-Hut method is $O(n\ log\ n)$ [41].

Ocean: It simulates large-scale ocean movements based on eddy and boundary currents. [42]

In Table 5 we present a breakdown of instructions executed for default problem sizes of these three applications on a 32 processor machine. Instructions executed are broken down into total floating point operations (FLOPS), reads and writes. As this table shows the number of instructions related to the shared read and writes is the highest for Barnes that causes high broadcast traffic for this application and is the smallest for Radix that causes low broadcast traffic for this application. It can interpret that due to high broadcast traffic in Barnes it mostly uses ONet and due to the low request of broadcast in Radix it mostly uses ENet. Ocean shows a mediator usage of both ONet and ENet

Table 5: Breakdown of instructions executed for default problem sizes [39]

Application	Problem size	Total instruction (M)	Total FLOPS (M)	Total Reads (M)	Total Writes (M)	Shared Reads (M)	Shared Writes (M)
Radix	1M integers, radix 1024	50.99	---	12.06	7,03	12.06	7.03
Barnes	16K particles	2002.79	239.24	406.85	313.29	225.05	93.23
Ocean	258 x 258 ocean	379.93	101.54	81.89	18.93	80.26	17.27

4.3. Data preprocessing

As Figure 11 part b shows data preprocessing is next step after gathering the data, and it is one of the most important steps in our technique. The input to this step is the original dataset that consists of seven features and four outputs as we discuss in section 3.1. We should carefully analyze and modify our original dataset to get a meaningful result. Otherwise, it will be misleading and caused poor prediction. This process includes dealing with missing value, normalization of data and transformation of data, feature selection, and feature extraction. The output of this step will be the final dataset that we will use for the learning process to derive a prediction model.

4.3.1. Statistical Analysis Tool

To perform data preprocessing, we need to use statistical analysis tools. At first, we developed several python scripts for setting design configuration of the ONoC and then extracting our desired data from output files of the simulator. Then we used Weka [34] for data preprocessing includes feature selection and feature transformation. After preparing the dataset

by applying preprocessing step, Weka used for building the prediction model and statistical analysis. The Weka workbench is a collection of state-of-the-art machine learning algorithms and data preprocessing tools. The system is written in Java and distributed under the terms of the GNU General Public License [34]. Other tools used for this purpose is MATLAB and Microsoft Excel to plot results.

4.3.2. Dealing with Missing Value

This step is about processing data to detect and correct inaccurate records from the dataset. In our dataset, the feature *"routing strategy"* has two possible values. One is "distance based strategy" describes that the network will route packet via different networks based on the distance threshold that was set for it. If the distance between the source and the destination is less than the specified threshold, the packet will be routed only via an electrical network ("ENet" in our case, see section 0). But if the distance between the source and the destination is equal or greater than the threshold, the packet will be routed using the hybrid opto-electrical network (combination of "ENet" and "ONet" described in section 0). Another routing strategy is "cluster-based strategy" where the routing between cores within a cluster is done using an electrical network ("ENet"), and the routing between the cores in the different cluster is done using the hybrid opto-electrical network (combination of "ENet" and "ONet"). Therefore, the feature "distance" that describes the threshold value does not have any meaning for the "cluster-based strategy" and will have a missing value in all configuration instances that correspond to cluster based routing strategy (where feature *"routing strategy"* has the value "cluster-based strategy"). The way that we deal with missing values for this case is to fill them by the maximum distance between cores in a cluster.

4.3.3. Normalization

This step in data preprocessing is useful for adjusting measured values by using normalization. For example, if we have features like memory size and it has value 100 "MB" and another feature like cache size that has value 32 "KB". Although they are two comparable features, they have a different scale (KB vs. MB) and therefore any comparison, addition or subtraction would be unreasonable. Then, the designer should make sure that all the features follow the same scale and are in same range. For doing this, usually, features are dividing by the maximum value encountered on a dataset for that particular feature. In our project, configuration

features that had numerical values count the number of modules. For example number of cores, number of access points, etc. Therefore by their nature, they have the same scale and range because they simply count as 1, 2, 3, etc. and there is no need for normalizing the values. But in general, normalizing the features in one of the required steps is data preprocessing.

4.3.4. Feature Transformation

Plotting each of the features against the delay and energy may reveal particularly strong association or identify the non-linear relation between them. This is useful for finding out a feature transformation function described in section 0. We extract relationships between features and each of four outputs separately based on their plots. We use curve fitting method to find the transformation function that relates our three numeric features (cluster size, number of access points, number of receive network) to each of the outputs. Here we present these plots only for Radix benchmark but the transformation functions for the other benchmarks are the same for each parameter.

1- Features vs. Average Packet Latency

Figure 14 plots how the value of "average packet latency" changes when we vary the number of cores per cluster (i.e. changing cluster size) from 1 to 32 (the range of its values specified in Table 3). The dotted lines represent the value of output in original data obtained from the simulation. Each line is for a set of experiments that have same configurations in all parameters except the cluster size. For example line blue is for all the configurations that have 64 number of cores, 1 number of access point, 1 receiving network, using distance based routing strategy with threshold 4 and has a standard laser. We include other lines that represent other configurations because we want to show that this relation between this feature and output is not an accident.

For the feature f_1 *"cluster size"* based on its plot (Figure 14) we apply a square root transformation function on it and it creates new feature which is $\sqrt{f_1}$ that f_1 is independent variable *"cluster size"*.

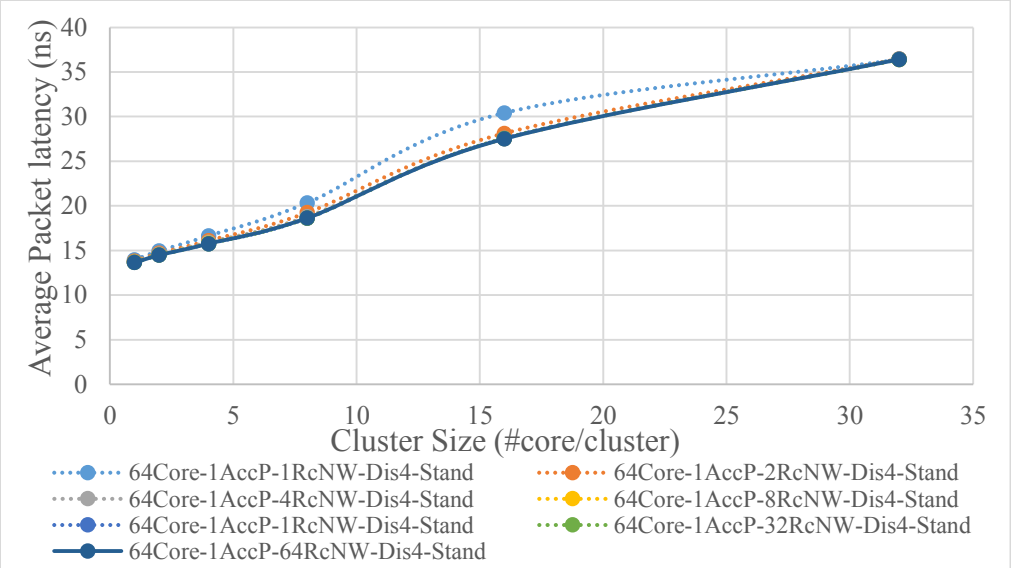


Figure 14: Cluster size (#cores/cluster) vs Average packet latency

For the feature *"number of access points"* based on its plot (Figure 15) we apply an inverse transformation function on it and it creates a new feature which $\frac{-1}{f_2}$ where f_2 is an independent variable *"number of access point"*.

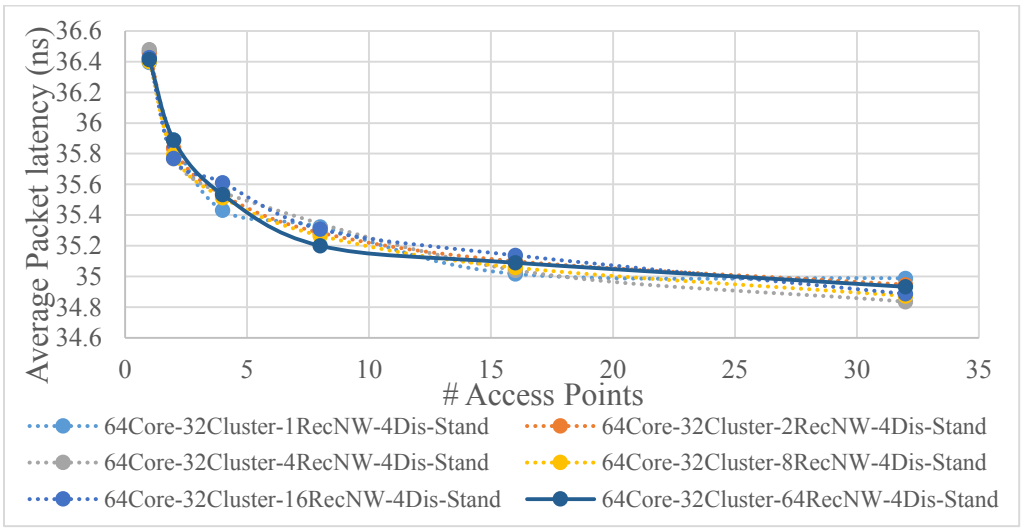


Figure 15: # Access points vs. Average packet latency

For the feature f_3 *"number of received networks"* based on its plot (Figure 16) we apply an inverse transformation function on it and it creates a new feature which is $\frac{1}{f_3}$ where f_3 is an independent variable representing *"number of received network"*.

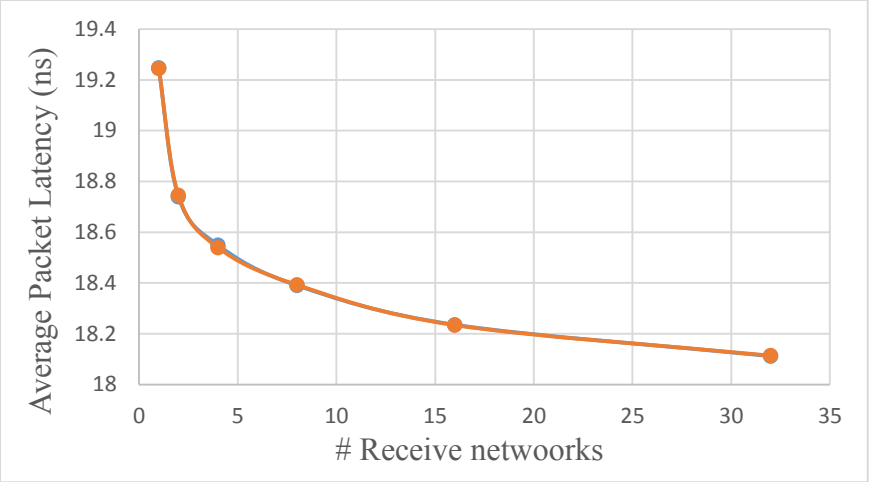


Figure 16: # Receive networks vs. Average packet latency

2- Features vs. Average Contention delay

For the feature f_1 *"cluster size"* based on its plot (Figure 17) we apply a square root transformation function on it and it creates new feature which is $\sqrt{f_1}$ that f_1 is independent variable *"cluster size"*.

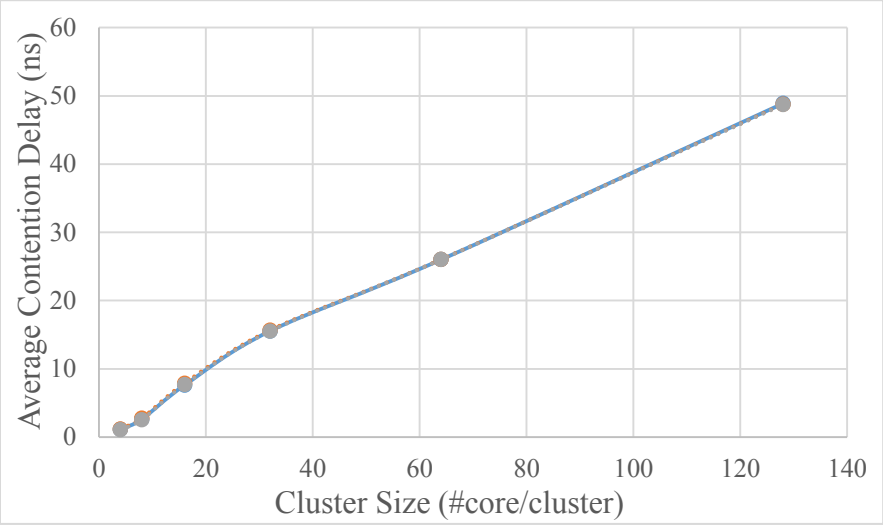


Figure 17: Cluster size (#cores/cluster) vs. Average contention delay

For the feature *"number of access points"* based on its plot (Figure 18) we apply an inverse transformation function on it and it creates a new feature which $\frac{-1}{f_2}$ where f_2 is an independent variable *"number of access point"*.

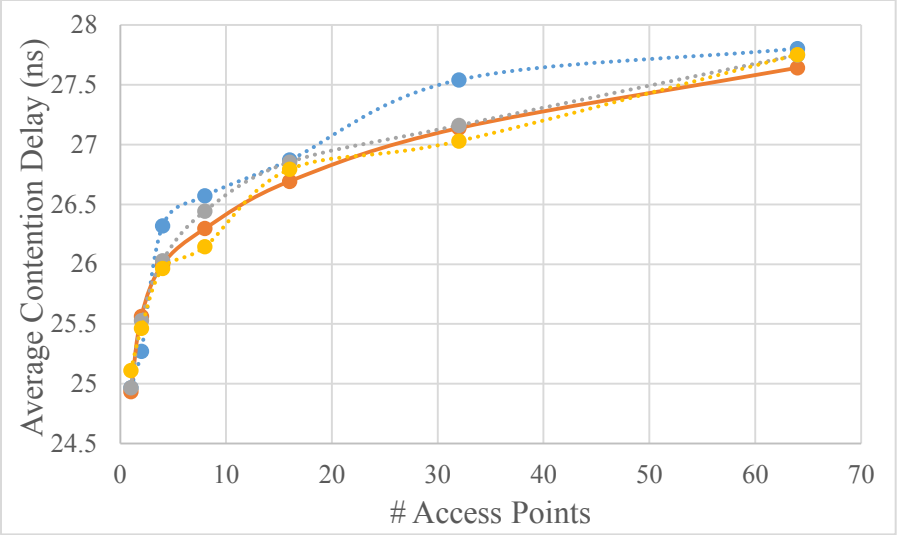


Figure 18: # Access points vs. Average contention delay

For the feature f_3 *"number of received networks"* based on its plot (Figure 19) we apply an inverse transformation function on it and it creates a new feature which is $\frac{1}{f_3}$ where f_3 is an independent variable representing *"number of received network"*.

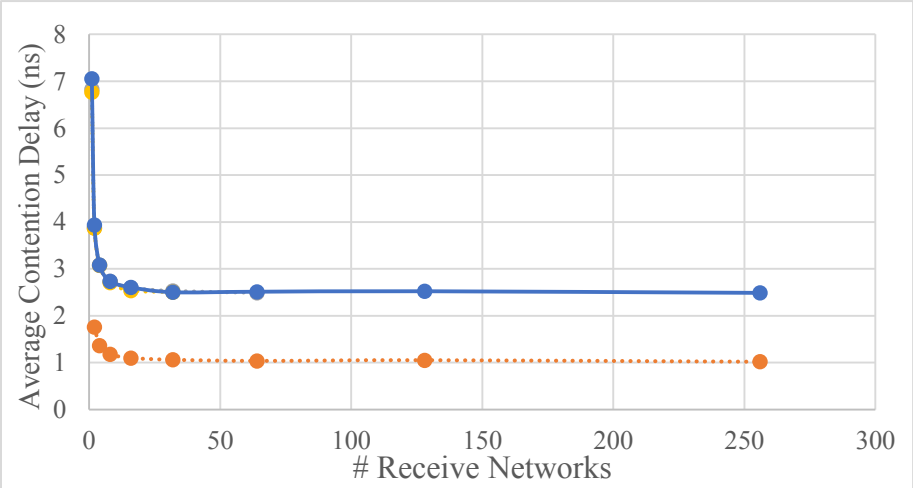


Figure 19: # Receive networks vs. Average contention delay

3- Features vs. Static Power

For the feature f_1 *"cluster size"* based on its plot (Figure 20) we apply an inverse transformation function on it and it creates new feature which is $\frac{1}{f_1}$ that f_1 is independent variable *"cluster size"*.

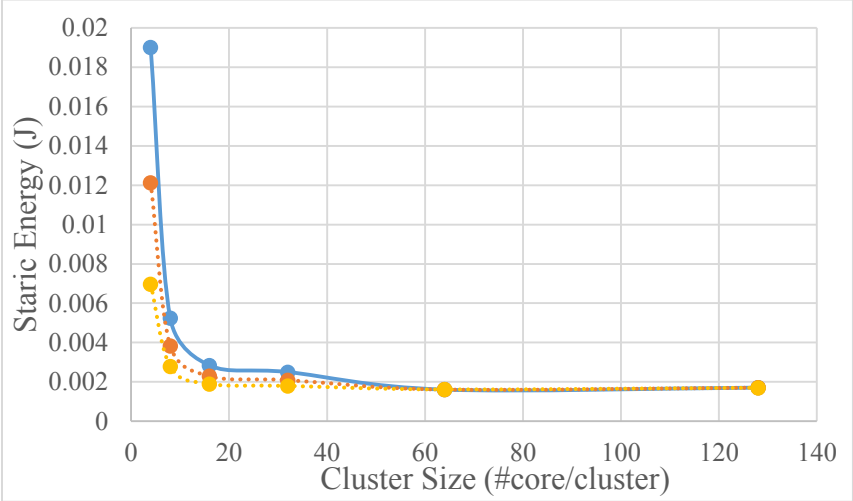


Figure 20: Cluster size (#cores/cluster) vs Static Energy

For the feature *"number of access points"* based on its plot (Figure 21) there is a linear relation between this feature and static energy. Therefore we keep this feature as is, without applying any transformation function on it.

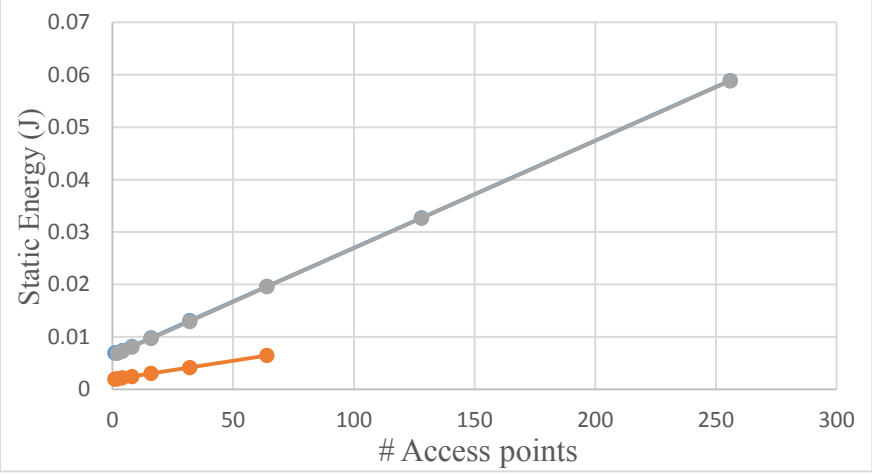


Figure 21: # Access points vs. Static Energy

For the feature *"number of receive networks"* based on its plot (Figure 22), there is a linear relation between this feature and static energy. Therefore we keep this feature as is, without applying any transformation function on it.

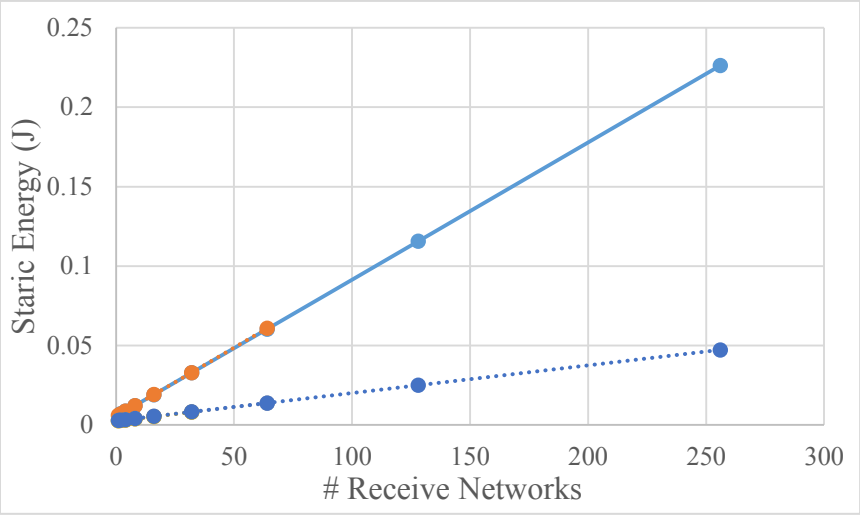


Figure 22: # Receive networks vs. Static Energy

4- Features vs. Dynamic Power

For the feature f_1 *"cluster size"* based on its plot (Figure 23) we apply an inverse transformation function on it and it creates new feature which is $\frac{1}{f_1}$ that f_1 is independent variable *"cluster size"*.

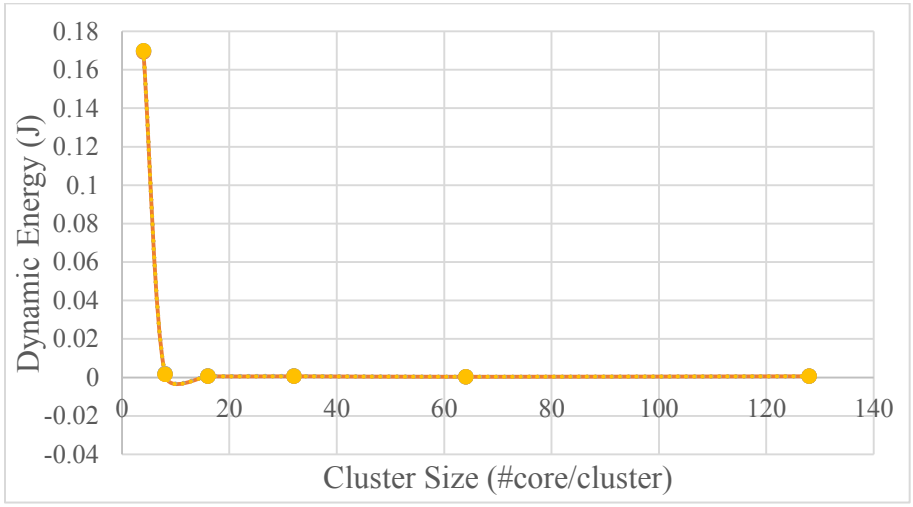


Figure 23: Cluster size (#cores/cluster) vs. Dynamic Energy

For the feature *"number of access points"* based on its plot (Figure 24) there is a linear relation between this feature and static energy. Therefore we keep this feature as is, without applying any transformation function on it.

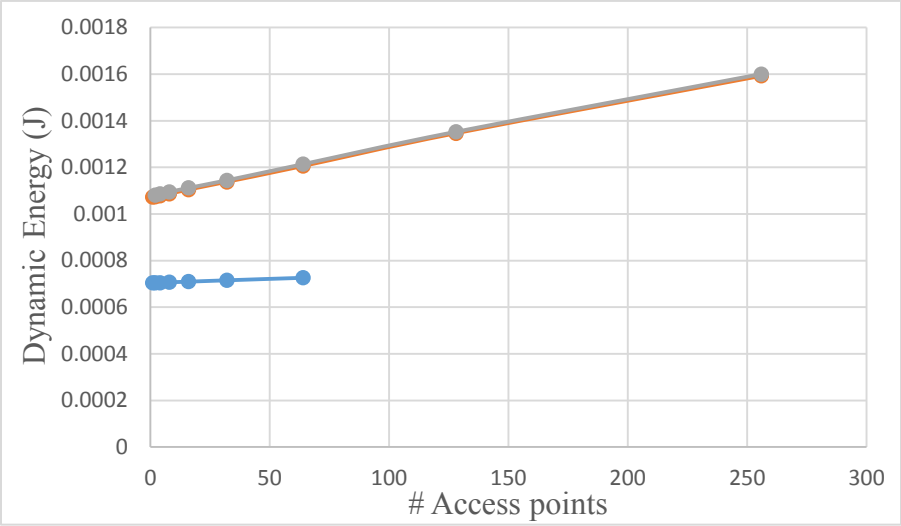


Figure 24: # Access points vs. Dynamic Energy

For the feature *"number of receive networks"* based on its plot (Figure 25), there is a linear relation between this feature and static energy. Therefore we keep this feature as is, without applying any transformation function on it.

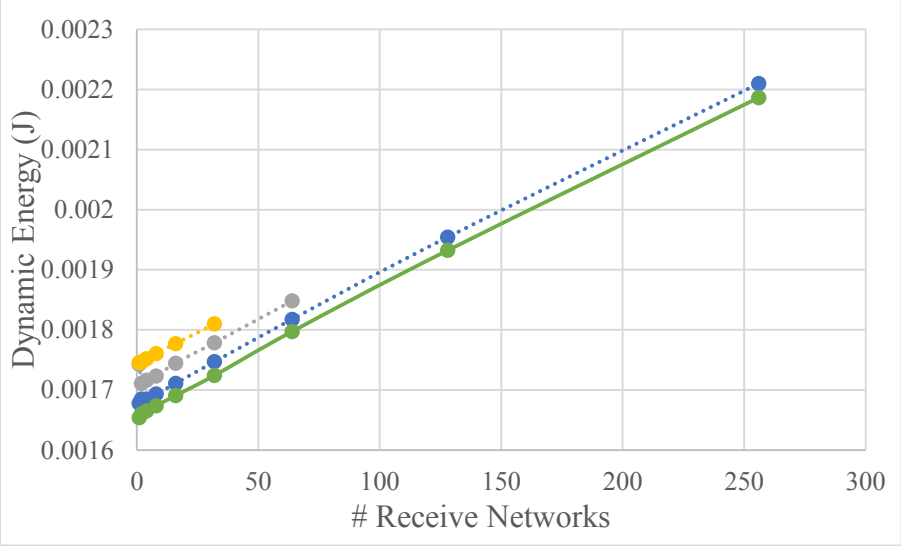


Figure 25: # Receive networks vs. Dynamic Energy

4.3.5. Feature Discretization

One of the essential steps in data preprocessing is feature discretization especially when we want to use a prediction model that can only handle features with nominal values. In this case, we should convert all the numeric values of features to nominal values. However, the prediction models that can handle features with both numeric and nominal values, usually show better prediction accuracy when the features have nominal values. It is because feature discretization can compact many numeric values of features into a finite and limit discrete set of values. Moreover, it helps that dataset will be more understandable and also prediction model work faster [34]. In the case of our dataset, feature discretization can be easily done due to the nature of the observed numerical features that belong to the limit set of district values, not a continuous value range. For example feature "number of cores" can have two numeric values: "64" and "256". We change these numeric values to two nominal values: type one "64 cores" and type two "256 cores". In section 5.6 we will present the prediction accuracy after applying feature discretization and compare it to the prediction accuracy of the original dataset with numeric features, and we will see that discretization technique improves prediction accuracy.

4.4. Prediction Modeling

The third step in our technique is Prediction Modeling. After applying the preprocessing technique on the dataset, we have a dataset ready to use for learning a model to make a prediction. We split the dataset randomly and take a portion of it to use as a training set, and then after deriving the model from it, we test the learned model on the rest of dataset, that it is test set. Section 5.1 will explain about splitting dataset to training and test set and their size. We employ several models for predicting the outputs. The learned models can divide to three separate categories: 1- Regression model, 2- Tree model, 3-Neural network model.

Regression models discuss in section 3.1. For deriving an accurate regression model, it is very important to find all the nonlinear interactions and dependencies between features and output. And that is why developing an accurate regression model is very difficult. Moreover, it is necessary to provide a dataset that all features have numeric values.

Tree models discuss in section 3.2. Tree models are very easy to follow and understand. We can start from the root, and see how features impact the output in each node. Because of this

advantage and their visual representation they are very useful for modeling any dataset with numeric or nominal features. Moreover, the tree can handle nonlinear relationships and interactions between features and output without any need to find them in the dataset before modeling explicitly.

Neural networks that also called “multilayered perceptron” simulates “neurons” that made a couple of layers. In the first layer of neurons, they accept features as an input and then find and apply weighting coefficients to them and send them to next layer. Next layer should also do the same task. This procedure continues until data reach to the output layer. Some layers may send feedback to their previous layers to optimize the resulted output. In this thesis, we do not focus on the neural network as a prediction model although it shows good accuracy in some cases. There are several disadvantages with neural networks compare to other prediction models that we discuss in this work. First of all neural networks are not an easy model to understand and explained to non-technical people. A neural network is like a black box that derives a predicted value without any clarification about how it comes out. The neural network is opposite to tree model that can interpret easily. The required time for building (i.e. training) a neural network is also higher compared to tree and regression models. A neural network can automatically determine any interactions and relationship between features and output. And because of this extreme detection of all possible interactions, a neural network is very prone to overfitting. Moreover, regression and tree models are less complicated to build therefore computational time for training these models are around a couple of seconds for our dataset. But due to high computation that needed for building a neural network, it takes more time to build the model that for our dataset is around a couple of minutes. Although neural networks is not an easy model for explaining and interpreting, it needs less domain knowledge to build, compare to the regression model. They are available software that can easily be used without any need for comprehension of the layer’s structure of neural networks.

In section 5.3, we will compare the prediction accuracy of these three different categories of prediction models. There is no study so far that can show one of these mentioned prediction models always perform better than the others and present more accurate predictions in all the dataset. But in our dataset “M5P tree” and “REP tree” give more accurate predictions almost for

all predicted outputs in three different benchmarks. Therefore we go through these two models for more details in this thesis.

In Figure 26 we can see a regression tree model called "M5P tree" for average packet latency. Each leaf node contains a linear model (LM) that calculate average packet latency. For instance, Equation 8 shows Linear Models for the magnified portion of Figure 26:

Equation 8: Linear Model #6 and #7 for Average Packet Latency

$LM\ num: 6$	$LM\ num: 7$
$avg\_latency\ =$	$avg\_latency\ =$
$-0.0035\ *Cores$	$-0.0075\ *Cores$
$+\ 0.0511\ *\ Cluster$	$+\ 0.0511\ *\ Cluster$
$-\ 0.2887\ *AccP$	$-\ 0.2912\ *AccP$
$-\ 1.2876\ *RecNW$	$-\ 0.0026\ *RecNW$
$+\ 0.3704\ *\ Routing = cluster\_based$	$+\ 0.3704\ *\ Routing = cluster\_based$
$-\ 0.0664\ *distance$	$-\ 0.0664\ *distance$
$+\ 25.8987$	$+\ 23.2418$

As we discuss in section 3.2, each leaf in M5P tree has an equation to calculate the output (here average packet latency) for the configurations that reach to that node on traversing the tree. In Table 6 we indicate the number of leaves (i.e. number of LM) in the tree for each output.

Table 6: Number of Leaves with Linear Model

Output	Number of Linear Model (LM)
Average packet latency	72
Average contention delay	58
Static energy	9
Dynamic energy	10

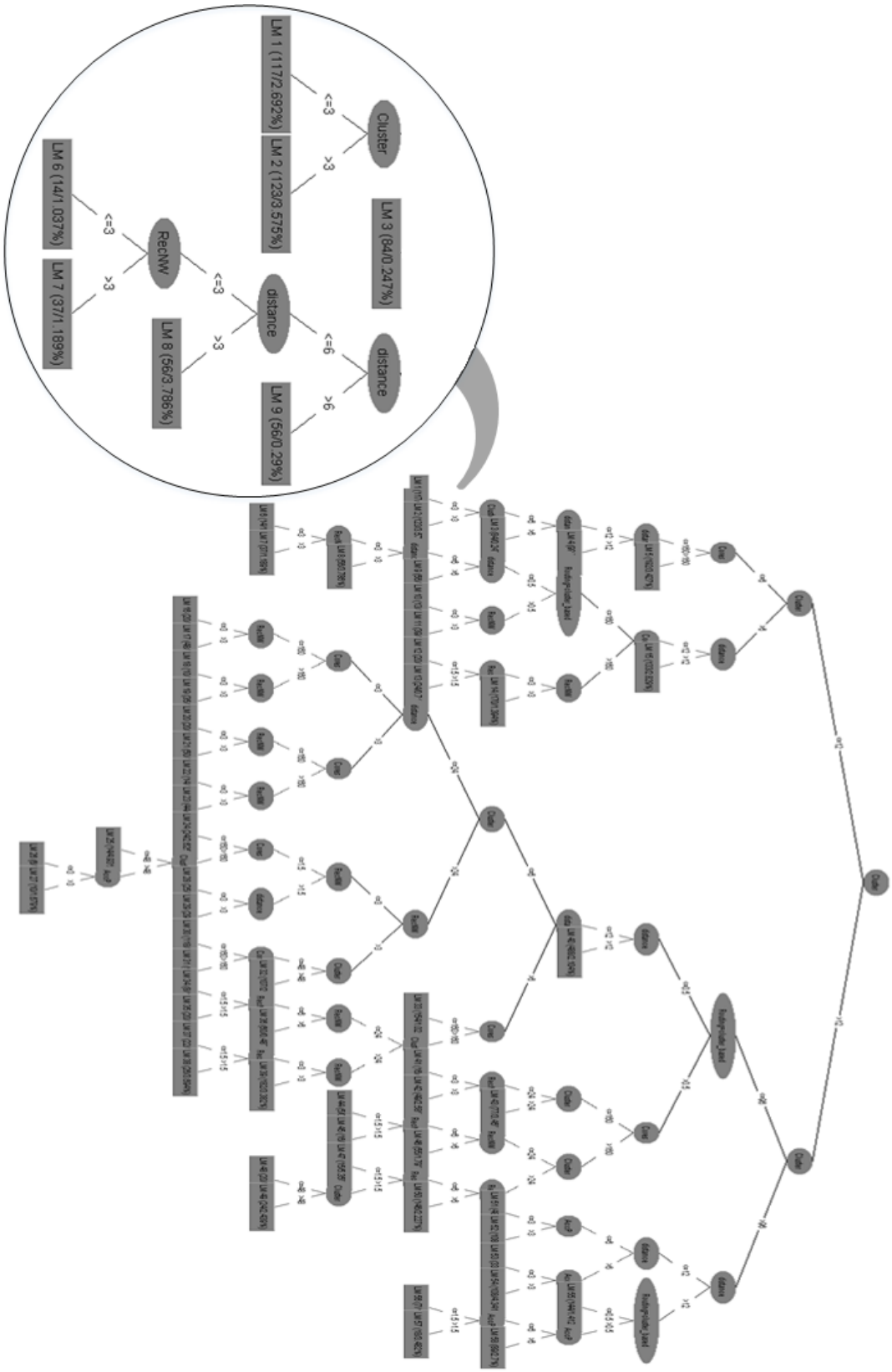


Figure 26: M5P tree for Average packet latency

4.5. Sampling.

In the first step that was generating the dataset, we gather more than 13 thousand simulated configurations to cover different architectures for ONoC. As next chapter will show, building a prediction model based on this amount of data provides a very accurate model. However, it is not always necessary to gather such enormous dataset to predict desired output with acceptable accuracy. In this section, we want to sample a small subset of configurations from original dataset and build our prediction model based on this smaller subset instead of the original large dataset. We test different size of dataset (i.e. different sample size) and formulate prediction models from them. Then we compare the accuracy of these models that build from different dataset sizes to see how better the model get at predicting the outputs when we increase the number of samples configurations that had been used for training the models.

Figure 27 shows learning curve for our prediction problem which is plotting the prediction error vs. the size of the dataset used for training the model. X-Axis is in percentage, for example, 10 in x-axis represents that 10% of original dataset now is used as a new dataset. We uniformly and randomly select 10, 20, 30, …, 80, 90 percent of the original dataset. As this learning curve shows here, by increasing the size of the dataset, prediction error decreases. Moreover only by having 40% of original dataset we can have a prediction model with less than 10% error.

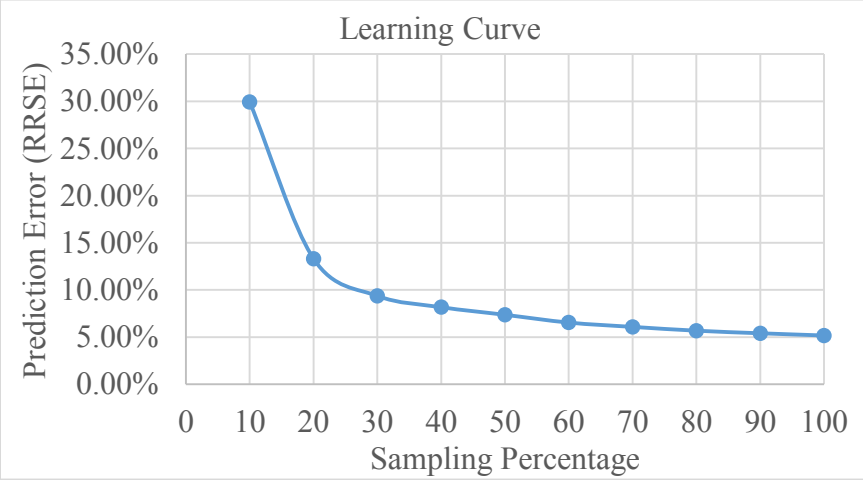


Figure 27: Learning Cure for Barnes Application with REPTree Model

By plotting learning curve, we want to show that it is possible to formulate a prediction model with an acceptable error with the smaller dataset. It is up to the designer to decide about

the size of the dataset based on her acceptable accuracy for the prediction model. In this way, we can reduce the number of simulations, and it solves the challenge of simulating unpractically large number configurations in traditional design space exploration. In section 5.5, we present accuracy of prediction models for different size of dataset.

CHAPTER 5

5. Model Evaluation and Experimental Results

In this chapter we discuss and analyze the prediction results: we observe its accuracy and compare our prediction models on different datasets. We start by presenting our evaluation method for the model and then compare prediction accuracy of different models on different:

1- application benchmarks

2- number of sampled configurations (i.e. size of the dataset)

3- methods for constructing the feature set

5.1. Evaluation Method

There are several techniques for evaluating the performance and accuracy of our model, but before introducing one common technique that we used for evaluation, we need to introduce a new dataset that is needed besides training and a test set. This set is a "*cross-validation set*". As described in section 3.6, the error rate of the training set in not a good indicator of a model for a future prediction on new data. Because the model has been derived from the same training data, and any estimation based on this dataset will be very optimistic. Therefore, for evaluating the model on new data, we need to test our model and its error rate on a dataset that played no part in the formation of the model. This independent dataset is test set. The important point here is that test dataset should not use in any way for creating the model. It may be needed to try out several models that have been derived from the same training dataset. The question is: how do we evaluate those multiple models to choose the most suitable one? If we were to evaluate them using the test set, it would not be fair to use the test set for selecting one of the several models, and after that, again to use the test set for evaluation of the picked model. If we were to do this, our evaluation of the picked model would be biased. The solution for this case is to split the dataset into three parts: the training set, cross-validation set, and test set. In this way, the training set is used for the learning process to create one or more models; the cross-validation set is used to select one of the models as a final, resulting model; and then the test set is used to calculate

the error rate of selected model. These three datasets must be independent, validation set has to be separate from the training set to select an optimum model, and test set must be different from both to get a reliable estimate of the error rate.

One standard way of estimating the error rate of one formulated model is “Ten Fold Cross Validation”. This is a method of evaluating the model during its creation, i.e. during the training phase. The “Ten Fold Cross Validation” flow chart is shown in Figure 28. In this method, the original dataset is randomly partitioned into 10 equal size partitions. Out of those 10 subsets, a single subset is retained as a cross-validation set for testing the model, and the remaining 9 subsets are used as a training set. The cross-validation process is then repeated 10 times with each of the 10 subsets used exactly once as the cross-validation set. The results from all iterations are averaged to generate a single value for estimated error rate of the model. This method is not limited to 10 folds; instead of 10 any number can be used, but extensive tests on numerous different dataset with different learning method have been shown that 10 is the good number to get the best result [34].

For obtaining an accurate error prediction, we repeat the cross-validation process 10 times. Performing 10 times of 10-fold cross validation results in overall 100 repetitions of the learning process using the original dataset. We do this because different 10-fold cross-validation experiments with same dataset and the same learning process might produce a different result due to the effect of the random selection of partitions. The final result of the training process is the obtained prediction model itself and also error rate of the corresponding model.

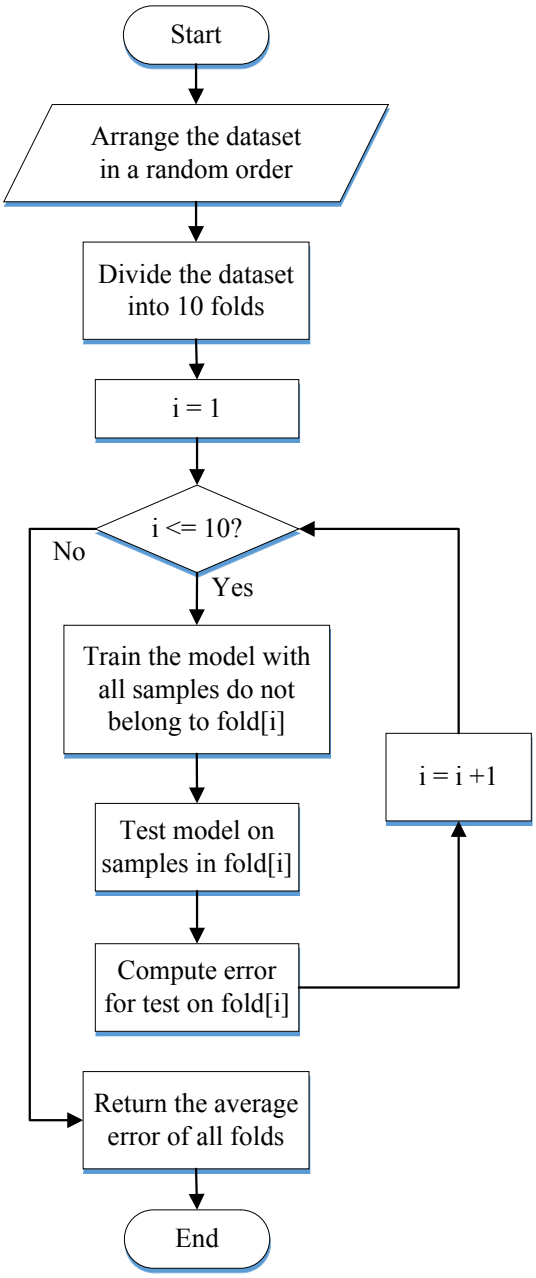


Figure 28: Flow chart for 10-fold cross validation

5.2. Accuracy of prediction

In this section, we present error formulation used for calculating error rate in each prediction model. There are several measures (e.g. one of the presented error formulation) that can be used for evaluating a prediction model.

5.2.1. Error formulation

For presenting the formula for each measure let's denote predicted value on the test instances with p_i and actual value with a_i. Notice that p_i and a_i are referring to the numerical values of the prediction and actual for the i^{th} test instance. [1]

Mean Absolute Error (MAE):

$$\frac{|p_1 - a_1| + |p_2 - a_2| + \cdots + |p_n - a_n|}{n}$$

Equation 9: Mean Absolute Error (MAE)

MAE measures the accuracy of prediction, and it is an average over the differences between prediction values and actual values. MAE measures the average of the errors for prediction without considering their direction (their signs). All the individual errors are weighted equally in the average.

Mean Squared Error (MSE):

$$\frac{(p_1 - a_1)^2 + (p_2 - a_2)^2 + \cdots + (p_n - a_n)^2}{n}$$

Equation 10: Mean Squared Error (MSE)

MSE measures the average of the error, the difference between predicted value and actual value are each squared and then average over all samples. Mean squared error tends to exaggerate the effect of the instances that have the prediction errors larger than the others> However, the mean absolute error (MAE) does not have this effect.

Root Mean Squared Error (RMSE):

$$\sqrt{\frac{(p_1 - a_1)^2 + (p_2 - a_2)^2 + \cdots + (p_n - a_n)^2}{n}}$$

Equation 11: Root Mean Squared Error (RMSE)

In Equation 11RMSE measures the same thing as the mean squared error does, with this difference that the square root of the average will be taken at the end. Since the errors are squared before they are averaged, the RMSE gives a relatively high weight to large errors. This means the RMSE is most useful when large errors are particularly undesirable.

The RMSE will always be larger or equal to the MAE. And that shows if we have the greater difference between MAE and RMSE, the greater the *variance* is in the individual errors in the sample. If the RMSE and MAE are equal, then it means all the errors are of the same magnitude.

MAE, MSE, and RMSE can range from 0 to ∞. They are negatively-oriented scores, and it means that the lower values are better in accuracy.

Relative Absolute Error (RAE):

$$\frac{|p_1 - a_1| + |p_2 - a_2| + \cdots + |p_n - a_n|}{|a_1 - \bar{a}| + |a_2 - \bar{a}| + \cdots + |a_n - \bar{a}|}$$

Equation 12: Relative Absolute Error (RAE)

where $\bar{a}$ is the mean value over the training set.

This is the total absolute error with the normalization. The errors are normalized by dividing the absolute error to average of actual values from training set ($\bar{a}$). The term indicates that how much a_i differs from its mean value, therefore it can tell that how much a_i differs from itself compared to the variance. Because of that the measures are named relative because they give result related to the scale of a.

Sometimes using relative error instead of an absolute error is important. For example, if we have a 10% error, it can be an error in case of predicting 50 instead of 500, or a case of

predicting 0.2 instead of 2. In this scenario, the average of absolute error is not meaningful, and using relative error will be useful.

Relative Squared Error (RSE):

$$\frac{(p_1 - a_1)^2 + (p_1 - a_1)^2 + \cdots + (p_1 - a_1)^2}{(a_1 - \bar{a})^2 + (a_2 - \bar{a})^2 + \cdots + (a_n - \bar{a})^2}$$

Equation 13: Relative Squared Error (RSE)

The RSE measures same thing as RAE, it just takes the total squared error and then normalizes it.

Root Relative Squared Error (RRSE):

$$\sqrt{\frac{(p_1 - a_1)^2 + (p_1 - a_1)^2 + \cdots + (p_1 - a_1)^2}{(a_1 - \bar{a})^2 + (a_2 - \bar{a})^2 + \cdots + (a_n - \bar{a})^2}}$$

Equation 14: Root Relative Squared Error (RRSE)

The root relative squared error measures the same thing as is RSE do, with this difference that the square root of RSE will be taken at the end.

As it explained here, all these error formulations compare actual values to their predicted values but do it in a different way. They all want to show how far away the predicted values are from the actual values. Sometimes square root is used because using square roots instead of absolute values; the extreme values have more influence on the result.

In MSE and RMSE the average difference between these two values (i.e. predicted values and actual values) are measured. And in RAE and RRSE those differences divided by the variation of actual values, therefore they have a scale from 0 to 1, and if you multiply this value by 100, you get similarity in 0-100 scale in percentage. That is how we report RAE and RRSE in percentage.

5.2.2. Correlation Coefficient

A correlation coefficient shows the degree of linear dependence of a's and p's. In other words, the coefficient shows how close two variables lie along a line. For example, if Correlation coefficient is 0.2055 it implies that 20.55% of the variance in the data is explained by the model. If the coefficient is equal to 1 or -1, all the points lie along a line. If the correlation coefficient is equal to zero, there is no linear relation between a and p. However, this does not necessarily mean that there is no relation at all between the two variables. There could be a non-linear relation between them for example.

$$cc = \frac{\frac{\sum_{i=1}^{n}(p_i - \bar{p})(a_i - \bar{a})}{n-1}}{\sqrt{\frac{\sum_{i=1}^{n}(p_i - \bar{p})^2}{n-1}\frac{\sum_{i=1}^{n}(a_i - \bar{a})^2}{n-1}}}$$

Equation 15: Correlation Coefficient

A positive correlation coefficient means that the two variables move in the same direction. That is: the higher value of a corresponds to higher values of p, and vice versa.

A negative correlation coefficient means that the two variables move in the opposite directions. That is: the lower value of a corresponds to higher values of p, and vice versa. Of course, negative values should not occur for reasonable prediction methods.

5.3. Evaluation of different Models

Here we will present different prediction models for three different benchmarks, Radix, Barnes, and Ocean. For each of them, we try several learning techniques based on three different categories that we discussed in section 4.4. These three model categories are Regression models shown in orange color in following tables, Neural Network models are shown in blue color, and Tree models are shown in green color. Finally, we select two models that present best results compared to the others. As we mentioned in section4.4, the best two models are M5P Tree and REP Tree that belongs to Tree models category. M5P tree builds a decision tree with a linear regression model function in each leaf node. REP tree is Reduced Error Pruning tree that builds a decision tree and then prunes some of its branches if this pruning reduces the prediction error. Detail explanation about these two models discussed in section 3.3 and 4.4. Before presenting

the results, we want to remind that as we explained in section 0, CC stands for Correlation Coefficient, MAE stands for Mean Absolute Error, RMSE stands for Root Mean Squared Error, RAE stands for Relative Absolute Error and RRSE stands for Root Relative Squared Error.

1) Radix:

Tables 7 to 10 show prediction accuracy of average packet delay, average contention delay, static energy and dynamic energy consumption, respectively, for Radix benchmark. As Tables 7 to 10 indicates, Regression models that show in orange color have not good accuracy, and average prediction error (RRSE) for them is 75.56%. Although we handle feature with nominal values with one hot encoding technique (we discussed it in section 3.1) and therefore we can use regression model for our prediction, they still do not perform with good accuracy. As we explained in section 4.4, the most important step for having an accurate Regression Model is that all the interactions and relations between features and output, carefully extracted from dataset otherwise it does not perform well as the prediction model. In our dataset, there were a lot of nonlinear relationship between features and outputs that cause a poor prediction for Regression models.

In Tables 7 to 10, blue color rows show Neural Network models. Average prediction error (RRSE) is 58.93% for them. Therefore they do not perform well as prediction models with good accuracy. The reason is that Neural networks are very prone to overfitting as we discussed in section 4.4. They obsessively detect any interactions between features and output, and it causes that they poorly predict in new data, although they show good accuracy in the training set.

Green color rows in the Tables 7 to 10 indicate Tree models. Trees show best results for our dataset. The average prediction error for them is 4.35%. The reason is that they can handle interactions between features and output very well, and it is done automatically by the way that the algorithm builds the tree. Moreover, they do not have the overfitting problem of the neural network because trees do not find all the possible interactions between features and output in training set that make them too much specific that cannot be generalized on a new dataset. Trees can handle both nominal and numeric features very well.

Table 7: Prediction accuracy of average packet latency for different models in Radix

Algorithm	CC	RMAE	MSE	RAE	RRSE
Linear Regression	0.7429	10.6299	13.8193	58.74%	66.92%
Additive Regression	0.8493	9.1956	10.898	50.82%	52.78%
RBF Network	0.3273	16.7789	19.5085	92.73%	94.48%
MultilayerPerceptron	0.9773	3.1819	4.4528	17.58%	21.56%
M5P tree	0.9992	0.6168	0.859	3.41%	4.16%
REP tree	0.9997	0.382	0.525	2.11%	2.54%

Table 8: Prediction accuracy of average contention delay for different models in Radix

Algorithm	CC	RMAE	MSE	RAE	RRSE
Linear Regression	0.7106	11.0282	14.3517	60.62%	70.35%
Additive Regression	0.8468	9.0939	10.8512	49.99%	53.19%
RBF Network	0.2814	17.126	19.5736	94.13%	95.94%
MultilayerPerceptron	0.9774	2.8861	4.3299	15.86%	21.22%
M5P tree	0.9991	0.6839	0.8933	3.76%	4.38%
REP tree	0.9997	0.3622	0.4948	1.99%	2.43%

Table 9: Prediction accuracy of static energy for different models in Radix

Algorithm	CC	RMAE	MSE	RAE	RRSE
Linear Regression	0.3119	6.9724	15.4227	124.70%	94.99%
Additive Regression	0.5445	6.8892	13.6254	123.21%	83.92%
RBF Network	0.1573	5.4222	16.0313	96.97%	98.74%
MultilayerPerceptron	0.9771	1.4916	3.4794	26.68%	21.43%
M5P tree	0.9982	0.4652	1.3035	8.32%	8.03%
REP tree	1.000	0.052	0.1121	0.93%	0.69%

Table 10: Prediction accuracy of dynamic energy for different models in Radix

Algorithm	CC	RMAE	MSE	RAE	RRSE
Linear Regression	0.2833	0.0181	0.0446	124.13%	95.89%
Additive Regression	0.5029	0.0183	0.0402	125.44%	86.44%
RBF Network	0.122	0.0145	0.0462	99.41%	99.24%
MultilayerPerceptron	0.9823	0.0049	0.0088	33.69%	18.86%
M5P tree	0.9966	0.0018	0.0048	12.54%	10.35%
REP tree	0.9997	0.0006	0.001	3.78%	2.24%

2) Barnes:

Second application benchmark that we tested is Barnes. Barnes shows same behaviors as Radix. Therefore we can see in Tables 11 to 14 that Regression Models and Neural Networks models do not perform well because of the same reasons that we discussed in Radix. Tree Models still indicates the best accuracy in Barnes application too.

We should note that although both of the MAE and RMSE have small values, they shouldn't lead us to make this conclusion that prediction models work well. Because the actual values and predicted values for these four outputs are small and therefore the difference between them are also small. This is the case that RAE and RRSE are useful. These relative measures cancel the effect that these small values of outputs may have on our reported error and make it meaningful. RRSE is 3.06% in average for all the four outputs that can show the REPTree model performs good prediction for all three applications.

Table 11: Prediction accuracy of average packet latency for different models in Barnes

Algorithm	CC	RMAE	MSE	RAE	RRSE
Linear Regression	0.87	1.6838	2.0824	45.89%	49.29%
Additive Regression	0.897	1.5115	1.8795	41.19%	44.49%
RBF Network	0.2872	3.4312	4.0459	93.51%	95.77%
MultilayerPerceptron	0.9568	0.9762	1.2422	26.61%	29.40%
M5P tree	0.9972	0.2205	0.3223	6.01%	7.63%
REP tree	0.9986	0.123	0.2236	3.35%	5.29%

Table 12: Prediction accuracy of average contention delay for different models in Barnes

Algorithm	CC	RMAE	MSE	RAE	RRSE
Linear Regression	0.7249	0.93	1.1973	58.45%	68.88%
Additive Regression	0.854	0.7425	0.9055	46.67%	52.09%
RBF Network	0.2423	1.5176	1.6863	95.39%	97.01%
MultilayerPerceptron	0.9627	0.3283	0.4732	20.64%	27.22%
M5P tree	0.9961	0.1046	0.1546	6.58%	8.90%
REP tree	0.9977	0.0768	0.1189	4.83%	6.84%

Table 13: Prediction accuracy of static energy for different models in Barnes

Algorithm	CC	RMAE	MSE	RAE	RRSE
Linear Regression	0.3153	57.0995	122.3058	118.24%	94.88%
Additive Regression	0.5813	54.3542	104.9146	112.55%	81.39%
RBF Network	0.1697	47.4139	127.0112	98.18%	98.53%
MultilayerPerceptron	0.9651	14.0046	34.0322	29.00%	26.40%
M5P tree	0.9987	3.1519	8.5658	6.53%	6.64%
REP tree	1.000	0.4221	0.8748	0.87%	0.68%

Table 14: Prediction accuracy of dynamic energy for different models in Barnes

Algorithm	CC	RMAE	MSE	RAE	RRSE
Linear Regression	0.3067	0.0266	0.0604	124.78%	95.16%
Additive Regression	0.5332	0.026	0.0537	122.08%	84.58%
RBF Network	0.154	0.0208	0.0627	97.70%	98.79%
MultilayerPerceptron	0.9915	0.0044	0.0083	20.65%	13.15%
M5P tree	0.9978	0.0021	0.0054	10.04%	8.44%
REP tree	0.9998	0.0007	0.0014	3.47%	2.13%

3) Ocean:

The third application that we tested is Ocean. The Ocean benchmark shows the same behavior as Radix and Barnes. Therefore we can see in Tables 15 to 18 that Regression and Neural networks models have not good accuracy in their predictions, and Tree models perform well.

Table 15: Prediction accuracy of average packet latency for different models in Ocean

Algorithm	CC	RMAE	MSE	RAE	RRSE
Linear Regression	0.8612	1.922	2.3324	46.40%	50.82%
Additive Regression	0.9057	1.5642	1.9495	37.76%	42.48%
RBF Network	-0.0501	4.1428	4.5903	100.02%	100.02%
MultilayerPerceptron	0.9651	0.9467	1.2061	22.86%	26.28%
M5P tree	0.9964	0.274	0.391	6.62%	8.52%
REP tree	0.9971	0.2252	0.3514	5.44%	7.66%

Table 16: Prediction accuracy of average contention delay for different models in Ocean

Algorithm	CC	RMAE	MSE	RAE	RRSE
Linear Regression	0.7006	1.5048	1.915	62.35%	71.35%
Additive Regression	0.8509	1.1513	1.4117	47.70%	52.60%
RBF Network	-0.0316	2.4139	2.6841	100.02%	100.01%
MultilayerPerceptron	0.9573	0.5911	0.7823	24.49%	29.15%
M5P tree	0.9941	0.1879	0.2924	7.79%	10.89%
REP tree	0.9945	0.1655	0.2805	6.86%	10.45%

Table 17: Prediction accuracy of static energy for different models in Ocean

Algorithm	CC	RMAE	MSE	RAE	RRSE
Linear Regression	0.3167	48.7379	104.883	118.70%	94.84%
Additive Regression	0.5767	46.7532	90.3586	113.86%	81.71%
RBF Network	0.1919	39.5401	108.5204	96.30%	98.13%
MultilayerPerceptron	0.9756	12.1043	24.3782	29.48%	22.04%
M5P tree	0.998	2.7646	8.4791	6.73%	7.67%
REP tree	0.9993	0.6765	4.1426	1.65%	3.75%

Table 18: Prediction accuracy of dynamic energy for different models in Ocean

Algorithm	CC	RMAE	MSE	RAE	RRSE
Linear Regression	0.3045	0.0566	0.1302	125.02%	95.23%
Additive Regression	0.5294	0.0557	0.1159	123.05%	84.82%
RBF Network	0.1692	0.0444	0.1347	98.08%	98.53%
MultilayerPerceptron	0.9887	0.0094	0.0204	20.77%	14.96%
M5P tree	0.9961	0.005	0.0141	11.02%	10.34%
REP tree	0.9978	0.0026	0.0091	5.69%	6.64%

As Tables 7 to 18 indicate, Tree Models (M5P tree and REP tree) have the best prediction accuracy for all the four outputs among all the three different benchmarks. Therefore for the rest of this thesis, we only present results for these two models in all three benchmarks. In next section, we will compare M5P tree and REP tree together.

5.4. Model Evaluation for different Benchmarks

In this section, we present the accuracy of prediction for M5P tree and REP tree models in Radix, Barnes, and Ocean benchmarks. As we explained in section 0, CC stands for Correlation Coefficient, MAE stands for Mean Absolute Error, RMSE stands for Root Mean Squared Error; RAE stands for Relative Absolute Error and RRSE stands for Root Relative Squared Error. In these tables avg_latency stands for average packet latency of the network, and avg_contention represents average contention delay of the network, stat_energy stands for the static energy of network and dynmc_energy stands for the dynamic power of the network. More details and explanations about these four outputs are available in section 2.4.1.

1) Radix:

Table 19: Accuracy of prediction with M5P tree model for Radix

Output	CC	MAE	RMSE	RAE	RRSE
avg_latency	0.9992	0.6168	0.859	3.41%	4.16%
avg_contention	0.9991	0.6839	0.8933	3.76%	4.38%
stat_energy	0.9982	0.4652	1.3035	8.32%	8.03%
dynmc_energy	0.9966	0.0018	0.0048	12.54%	10.35%

Table 20: Accuracy of prediction with REP tree model for Radix

Output	CC	MAE	RMSE	RAE	RRSE
avg_latency	0.9997	0.382	0.525	2.11%	2.54%
avg_contention	0.9997	0.3622	0.4948	1.99%	2.43%
stat_energy	1.000	0.052	0.1121	0.93%	0.69%
dynmc_energy	0.9997	0.0006	0.001	3.78%	2.24%

Tables 19 and 20 show the accuracy of prediction for Radix benchmark with M5P tree and REP tree models, respectively. M5P tree prediction error (RRSE) is 6.73% in average, and REP tree has RRSE of 1.97% in average. REP tree is showing better result because it has this advantage to use “reduce error pruning” technique. Therefore it prunes in the way that causes less error rate on independent data, consequently can generalize better on new data. In section 3.3, we discussed pruning technique in details.

2) Barnes:

Table 21: Accuracy of prediction with M5P tree model for Barnes

Output	CC	MAE	RMSE	RAE	RRSE
avg_latency	0.9972	0.2205	0.3223	6.01%	7.63%
avg_contention	0.9961	0.1046	0.1546	6.58%	8.90%
stat_energy	0.9987	3.1519	8.5658	6.53%	6.64%
dynmc_energy	0.9978	0.0021	0.0054	10.04%	8.44%

Table 22: Accuracy of prediction with REP tree model for Barnes

Output	CC	MAE	RMSE	RAE	RRSE
avg_latency	0.9986	0.123	0.2236	3.35%	5.29%
avg_contention	0.9977	0.0768	0.1189	4.83%	6.84%
stat_energy	1	0.4221	0.8748	0.87%	0.68%
dynmc_energy	0.9998	0.0007	0.0014	3.47%	2.13%

Table 21 and 22 show the accuracy of prediction for Barnes benchmark with M5P and REP tree models respectively. RRSE is 7.90% in average for M5P tree, and 3.73% in average for REP tree. Same as Radix benchmark, REP tree is showing the better result for Barnes application due to reducing error pruning technique that was applied to it.

3) Ocean:

Table 23: Accuracy of prediction with M5P tree model for Ocean

Output	CC	MAE	RMSE	RAE	RRSE
avg_latency	0.9964	0.274	0.391	6.62%	8.52%
avg_contention	0.9941	0.1879	0.2924	7.79%	10.89%
stat_energy	0.998	2.7646	8.4791	6.73%	7.67%
dynmc_energy	0.9961	0.005	0.0141	11.02%	10.34%

Table 24: Accuracy of prediction with REP tree model for Ocean

Output	CC	MAE	RMSE	RAE	RRSE
avg_latency	0.9971	0.2252	0.3514	5.44%	7.66%
avg_contention	0.9945	0.1655	0.2805	6.86%	10.45%
stat_energy	0.9993	0.6765	4.1426	1.65%	3.75%
dynmc_energy	0.9978	0.0026	0.0091	5.69%	6.64%

Table 23 and 24 show the accuracy of prediction for Ocean benchmark with M5P and REP tree models respectively. M5P has RRSE 9.35% in average and REP tree has RRSE 7.12% in average. Same as Radix and Barnes benchmarks, REP tree is showing the better result for Ocean application due to "reduce error pruning" technique that was applied to it.

We should note that although both of the MAE and RMSE have small values, such as 0.22 and 0.35, for predicting average packet latency with REP tree model, in Ocean benchmark they shouldn't lead us to make this conclusion that prediction model works well. Because the actual values and predicted values for these four outputs are small and therefore the difference between them are also small. This is the case that RAE and RRSE are useful. These relative measures cancel the effect that these small values of outputs may have on our reported error and make it meaningful. RRSE is 4.27% in average for all the four outputs in these three benchmarks that can show the REP Tree model performs good prediction for all three applications.

5.5. Evaluation of Model for different Sample size

As we discussed in section 4.5, we can uniformly and randomly sample a small subset of configurations from original dataset, and build our prediction model based on this smaller subset instead of the original large dataset. We test different size of dataset (i.e. different sample size) and formulate prediction models from them. Then we compare the accuracy of these models that build from different dataset sizes. We only present one output (average packet latency) for all three benchmarks to study the effect of sample size on the prediction model accuracy. The other outputs, average contention delay, static energy and dynamic energy show the same behavior as we will discuss here. Therefore we can generalize the conclusion from this output to the other outputs.

Sampling percentage indicates what percentage of the entire dataset was used for building the prediction model. For example, we have 4400 samples with different configurations gathered from Barnes application; we can use 10 % of this dataset (i.e. 440 samples) to build our prediction model and test it. It is exactly like that we have only 440 samples in our dataset, and want to build a prediction model. Therefore this dataset with 440 samples should split to the training set and cross-validation set, to perform 10-fold cross validation method that explained in section 5.1. More details about sampling technique are in section 4.5.

1) Radix:

Table 25: Prediction accuracy of different sample size for Avg packet latency in Radix

Sampling Percentage	CC	MAE	RMSE	RAE	RRSE
10	0.996	1.1233	1.8308	6.23%	8.88%
20	0.9985	0.6529	1.1605	3.54%	5.54%
30	0.999	0.53	0.9287	2.93%	4.48%
40	0.9991	0.4874	0.883	2.76%	4.32%
50	0.9993	0.4471	0.7782	2.54%	3.82%
60	0.9995	0.4207	0.6524	2.37%	3.19%
70	0.9995	0.4077	0.6468	2.30%	3.16%
80	0.9996	0.3927	0.5996	2.20%	2.92%
90	0.9997	0.3753	0.5152	2.10%	2.51%
100	0.9997	0.3708	0.5169	2.09%	2.53%

Table 25, 26, 27 and show when sample size increase the mean absolute error (MAE), root mean squared error (RMSE), relative absolute error (RAE) and root relative squared error (RRSE) are progressively reduced. It is because having more samples from design space helps for a better formulation of the prediction model. However based on those tables the correlation coefficient (CC) increase, when sample size increase. It shows that by increasing dataset size the linear dependency of features and outputs also increase, and it means that our model can formulate the outputs more accurate.

2) Barnes:

Table 26: Prediction accuracy of different sample size for Avg packet latency in Barnes

Sampling Percentage	CC	MAE	RMSE	RAE	RRSE
10	0.9541	0.7811	1.3063	20.12%	29.93%
20	0.9911	0.327	0.5679	8.76%	13.32%
30	0.9956	0.2304	0.3961	6.26%	9.38%
40	0.9967	0.1956	0.346	5.31%	8.18%
50	0.9973	0.174	0.309	4.77%	7.37%
60	0.9979	0.1522	0.2733	4.19%	6.54%
70	0.9981	0.1371	0.2548	3.78%	6.09%
80	0.9984	0.126	0.2371	3.48%	5.68%
90	0.9985	0.1195	0.2256	3.30%	5.41%
100	0.9987	0.1126	0.2163	3.10%	5.18%

3) Ocean:

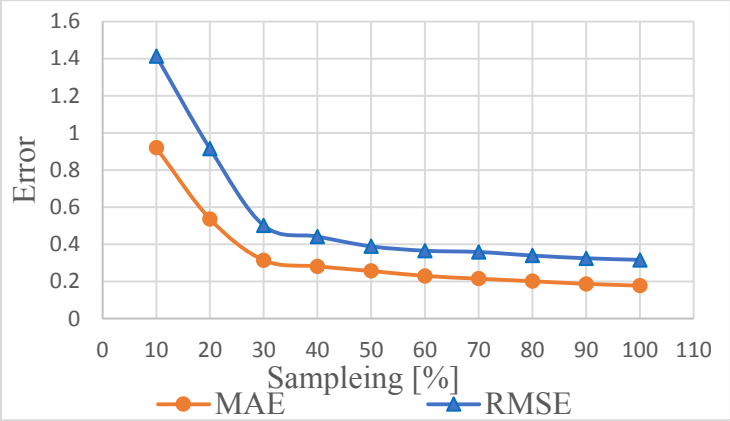
Table 27: Prediction accuracy of different sample size for Avg packet latency in Ocean

Sampling Percentage	CC	MAE	RMSE	RAE	RRSE
10	0.9525	0.9202	1.414	21.70%	30.44%
20	0.98	0.537	0.9157	12.88%	19.92%
30	0.994	0.3151	0.5018	7.60%	10.97%
40	0.9953	0.2818	0.4423	6.78%	9.64%
50	0.9964	0.2568	0.3895	6.20%	8.51%
60	0.9968	0.2303	0.3661	5.58%	8.01%
70	0.9969	0.2154	0.3592	5.22%	7.86%
80	0.9972	0.2018	0.3401	4.91%	7.47%
90	0.9975	0.1875	0.3254	4.55%	7.13%
100	0.9976	0.178	0.3168	4.35%	6.96%

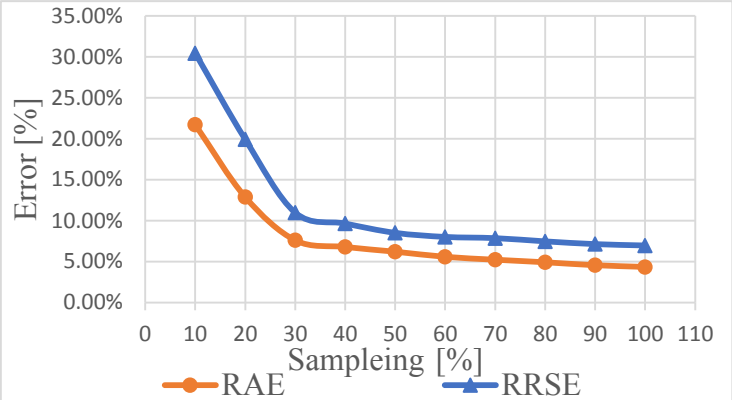
Figure 29, part a, b and c show the learning curve for MAE and RMSE, RAE and RRSE, and CC respectively in the ocean benchmark. The other benchmarks also show similar learning curves. These curves are based on the values in Table 27. By plotting learning curve and presenting these tables, we want to show that it is possible to formulate a prediction model with an acceptable error with the smaller dataset. It is up to the designer to decide about the size of the

dataset based on her acceptable accuracy for the prediction model. In this way, we can reduce the number of simulations, and it solves the challenge of simulating unpractically large number configurations in traditional design space exploration. In section 4.5, we discussed how we do the sampling for our dataset.

a)



b)



c)

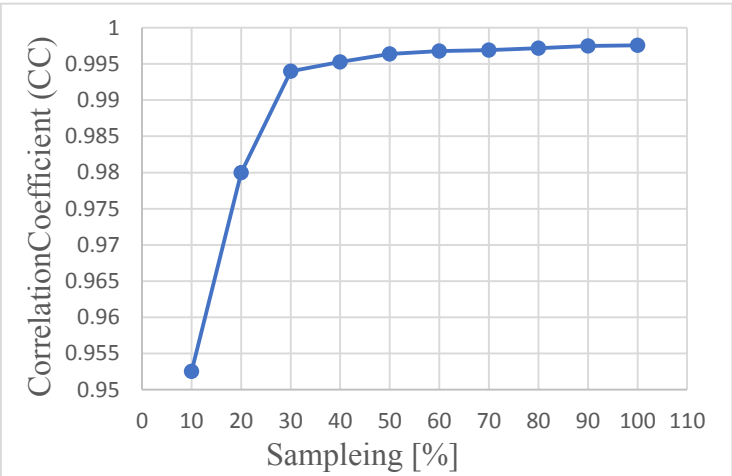


Figure 29: Data sampling percentage vs Error rate

5.6. Evaluation of Models for different Feature sets

In this section, we compare prediction accuracy of models with the original dataset and with the modified dataset, after applying feature transformation and feature discretization.

5.6.1. Prediction accuracy after applying feature transformation

We compare two different feature sets for each output, original feature set and modified feature set after applying transformation function on it, as we described in section 4.3.4.

1) Radix:

Tables 28 to 31 show prediction accuracy of M5P tree model after applying transformation functions on features and compare it with original dataset. There is less than 1% better accuracy after applying transformation function. It shows that our transformation function cannot generalize well to the other new data. In some cases applying transformation function on features causes even poorer prediction compare to the prediction based on the original dataset. These cases have been shown in red boxes in tables.

Table 28: M5P tree before and after applying transformation on dataset in Radix

	After Transformation					Original Numeric				
Output	**CC**	**MAE**	**MSE**	**RAE**	**RSE**	**CC**	**MAE**	**MSE**	**RAE**	**RSE**
avg_latency	0.9994	0.5532	0.7182	3.06%	3.48%	0.9992	0.6168	0.859	3.41%	4.16%
avg_contention	0.9993	0.616	0.7719	3.39%	3.78%	0.9991	0.6839	0.893	3.76%	4.38%
StatEnergy	0.9987	0.4208	1.2146	7.53%	7.48%	0.9982	0.4652	1.304	8.32%	8.03%
DynmcEnergy	0.9969	0.0018	0.0048	12.49%	10.41%	0.9966	0.0018	0.005	12.54%	10.35%

Table 29: REP tree before and after applying transformation on dataset in Radix

	After Transformation					Original Numeric				
Output	**CC**	**MAE**	**MSE**	**RAE**	**RSE**	**CC**	**MAE**	**MSE**	**RAE**	**RSE**
avg_latency	0.9997	0.3819	0.5248	2.11%	2.54%	0.9997	0.382	0.525	2.11%	2.54%
avg_contention	0.9997	0.3621	0.4948	1.99%	2.43%	0.9997	0.3622	0.4948	1.99%	2.43%
StatEnergy	1.000	0.052	0.1121	0.93%	0.69%	1.000	0.052	0.1121	0.93%	0.69%
DynmcEnergy	0.9997	0.0006	0.001	3.78%	2.24%	0.9997	0.0006	0.001	3.78%	2.24%

Table 30: Linear Regression before and after applying transformation on dataset in Radix

	After Transformation					Original Numeric				
Output	**CC**	**MAE**	**MSE**	**RAE**	**RSE**	**CC**	**MAE**	**MSE**	**RAE**	**RSE**
avg_latency	0.7437	10.6431	13.801	58.82%	66.84%	0.7429	10.63	13.819	58.74%	66.92%
avg_contention	0.715	11.0262	14.2599	60.61%	69.90%	0.7106	11.028	14.352	60.62%	70.35%
StatEnergy	0.3948	6.6708	14.9142	119.30%	91.86%	0.3119	6.9724	15.423	124.70%	94.99%
DynmcEnergy	0.338	0.0181	0.0438	124.13%	94.11%	0.2833	0.0181	0.0446	124.13%	95.89%

Table 31: Neural Networks before and after applying transformation on dataset in Radix

	After Transformation					Original Numeric				
Output	**CC**	**MAE**	**MSE**	**RAE**	**RSE**	**CC**	**MAE**	**MSE**	**RAE**	**RSE**
avg_latency	0.9779	3.0679	4.3514	16.95%	21.07%	0.9773	3.1819	4.4528	17.58%	21.56%
avg_contention	0.9788	2.7901	4.2068	15.34%	20.62%	0.9774	2.8861	4.3299	15.86%	21.22%
StatEnergy	0.9995	0.2935	0.494	5.25%	3.04%	0.9771	1.4916	3.4794	26.68%	21.43%
DynmcEnergy	0.9995	0.0009	0.0015	6.41%	3.27%	0.9823	0.0049	0.0088	33.69%	18.86%

2) Barnes:

This is the same behavior for Barnes application too. Transformation functions are not a good representative of the association between features and outputs.

Table 32: M5P tree before and after applying transformation on dataset in Barnes

	After Transformation					Original Numeric				
Output	**CC**	**MAE**	**MSE**	**RAE**	**RSE**	**CC**	**MAE**	**MSE**	**RAE**	**RSE**
avg_latency	0.9982	0.1883	0.2567	5.13%	6.08%	0.9972	0.2205	0.3223	6.01%	7.63%
avg_contention	0.9973	0.0916	0.1288	5.76%	7.41%	0.9961	0.1046	0.1546	6.58%	8.90%
StatEnergy	0.9991	2.9647	8.1383	6.14%	6.31%	0.9987	3.1519	8.5658	6.53%	6.64%
DynmcEnergy	0.9979	0.0022	0.0055	10.20%	8.74%	0.9978	0.0021	0.0054	10.04%	8.44%

3) Ocean:

There is also same behavior for Ocean application too. Transformation functions are not a good representative of the association between features and outputs.

Table 33: M5P tree before and after applying transformation on dataset in Ocean

	After Transformation					Original Numeric				
Output	**CC**	**MAE**	**MSE**	**RAE**	**RSE**	**CC**	**MAE**	**MSE**	**RAE**	**RSE**
avg_latency	0.9972	0.2447	0.346	5.91%	7.54%	0.9964	0.274	0.391	6.62%	8.52%
avg_contention	0.9947	0.1805	0.2771	7.48%	10.33%	0.9941	0.1879	0.2924	7.79%	10.89%
StatEnergy	0.9983	2.5838	8.3885	6.29%	7.59%	0.998	2.7646	8.4791	6.73%	7.67%
DynmcEnergy	0.9961	0.0051	0.0146	11.20%	10.66%	0.9961	0.005	0.0141	11.02%	10.34%

Although Transformation functions do not perform well on M5P tree model, they have very good impact on the prediction accuracy of Neural Network model as it shows in Table 34.

Table 34: Neural network before and after applying transformation functions in Ocean

	After Transformation					Original Numeric				
Output	**CC**	**MAE**	**MSE**	**RAE**	**RSE**	**CC**	**MAE**	**MSE**	**RAE**	**RSE**
avg_latency	0.9779	3.0679	4.3514	16.95%	21.07%	0.9773	3.1819	4.4528	17.58%	21.56%
avg_contention	0.9788	2.7901	4.2068	15.34%	20.62%	0.9774	2.8861	4.3299	15.86%	21.22%
StatEnergy	0.9995	0.2935	0.494	5.25%	3.04%	0.9771	1.4916	3.4794	26.68%	21.43%
DynmcEnergy	0.9995	0.0009	0.0015	6.41%	3.27%	0.9823	0.0049	0.0088	33.69%	18.86%

5.6.2. Prediction accuracy after applying feature discretization

We also try two different ways of formulating the model. First, we consider all the feature values as nominal type, whether their original values were nominal or numeric. In section 4.3.5 we explain this method as feature discretization. Second, we take all the features as a numeric type, whether the original values were numeric or nominal. We discussed this method as one hot encoding technique in section 3.1.1. Now we compare these two different feature sets in three different benchmarks: Radix, Barnes, and Ocean.

1) Radix:

As you can see in Table 35, the accuracy of formulated model with nominal feature set is better to compare to formulated model with the numeric feature set. In the best case, the accuracy is better by 1.88%, and in average by 1.10% in root relative squared error (RRSE) for Radix benchmark.

Table 35: Comparison between Nominal and Numeric dataset in Radix using M5P tree

	Nominal					Numeric				
Output	**CC**	**MAE**	**MSE**	**RAE**	**RSE**	**CC**	**MAE**	**MSE**	**RAE**	**RSE**
avg_latency	0.9995	0.5072	0.6479	2.80%	3.14%	0.9992	0.6168	0.859	3.41%	4.16%
avg_contention	0.9993	0.579	0.7595	3.18%	3.72%	0.9991	0.6839	0.8933	3.76%	4.38%
stat_energy	0.9991	0.3609	0.9992	6.45%	6.15%	0.9982	0.4652	1.3035	8.32%	8.03%
dynmc_energy	0.997	0.0018	0.0044	12.04%	9.49%	0.9966	0.0018	0.0048	12.54%	10.35%

The number of linear models in the leaves is also decreased. It means the model has a fewer number of leaves that have a linear model. Therefore prediction model function that is a piecewise function has a fewer number of sub-functions compare to the model with the numeric dataset, and therefore the model is simpler. The designer prefers less complex decision tree because they are more comprehensible. Furthermore, according to [43], the tree complexity has a crucial effect on its accuracy.

Output	Nominal	Numeric
avg_latency	31	72
avg_contention	32	58
stat_energy	6	9
dynmc_energy	6	10

2) Barnes:

As you can see in Table 36, the accuracy of formulated model with nominal feature set is better to compare to formulated model with the numeric feature set. In the best case, the accuracy is better by 3.25%, and in average by 1.95% in root relative squared error (RRSE) for Barnes benchmark.

Table 36: Comparison between Nominal and Numeric dataset in Barnes using M5P tree

	Nominal					Numeric				
Output	**CC**	**MAE**	**MSE**	**RAE**	**RSE**	**CC**	**MAE**	**MSE**	**RAE**	**RSE**
avg_latency	0.9991	0.1435	0.185	3.91%	4.38%	0.9972	0.2205	0.3223	6.01%	7.63%
avg_contention	0.9977	0.0825	0.1182	5.18%	6.80%	0.9961	0.1046	0.1546	6.58%	8.90%
stat_energy	0.9993	2.4453	6.4427	5.06%	5.00%	0.9987	3.1519	8.5658	6.53%	6.64%
dynmc_energy	0.998	0.002	0.0048	9.56%	7.61%	0.9978	0.0021	0.0054	10.04%	8.44%

3) Ocean:

As you can see in Table 37, the accuracy of formulated model with nominal feature set is better to compare to formulated model with the numeric feature set. In the best case, the accuracy is better by 1.71%, and in average by 1.18% in root relative squared error (RRSE) for Ocean benchmark.

Table 37: Comparison between Nominal and Numeric dataset in Ocean using M5P tree

	Nominal					Numeric				
Output	**CC**	**MAE**	**MSE**	**RAE**	**RSE**	**CC**	**MAE**	**MSE**	**RAE**	**RSE**
avg_latency	0.9977	0.2205	0.3124	5.32%	6.81%	0.9964	0.274	0.391	6.62%	8.52%
avg_contention	0.9952	0.1746	0.2631	7.23%	9.80%	0.9941	0.1879	0.2924	7.79%	10.89%
stat_energy	0.9986	2.2005	6.9993	5.36%	6.33%	0.998	2.7646	8.4791	6.73%	7.67%
dynmc_energy	0.9962	0.0049	0.0133	10.82%	9.75%	0.9961	0.005	0.0141	11.02%	10.34%

Moreover, the nominal type feature set shows better results for 5 models out of 6 different prediction models that we tested as it has shown in Table 38. We only present one output here (average packet latency) to check the effect of different feature sets in the prediction model accuracy. The other outputs, average contention delay, static energy and dynamic energy show the same behavior as we discuss here. Therefore we can generalize the conclusion from this output to the other outputs.

Table 38: Nominal vs. Numeric dataset in Radix for different prediction models

	Nominal					Numeric				
Algorithm	**CC**	**MAE**	**MSE**	**RAE**	**RSE**	**CC**	**MAE**	**MSE**	**RAE**	**RSE**
Linear Regression	0.8909	7.9261	9.3788	43.80%	45.42%	0.7429	10.63	13.82	58.74%	66.92%
Additive Regression	0.8482	9.2086	10.94	50.89%	52.98%	0.8493	9.1956	10.9	50.82%	52.78%
RBF Network	0.3707	16.44	19.174	90.85%	92.86%	0.3273	16.779	19.51	92.73%	94.48%
Multilayer Perceptron	0.9999	0.2077	0.2761	1.15%	1.34%	0.9773	3.1819	4.453	17.58%	21.56%
M5P tree	0.9995	0.5072	0.6479	2.80%	3.14%	0.9992	0.6168	0.859	3.41%	4.16%
REP tree	0.9997	0.3093	0.4936	1.71%	2.39%	0.9997	0.382	0.525	2.11%	2.54%

5.7. Prediction Analysis

In this section, we analyze the errors of prediction model "REP tree" as the most accurate model that we have. At first, we will visualize the errors for the model by plotting the actual value of outputs versus the predicted values for them, then we discuss the distribution of the errors by showing the most frequent error range, maximum and minimum error.

5.7.1. Visualizing Errors

Here we want to look at Absolute accuracy by plotting observed and predicted values of outputs in the dataset. We only present one output, average packet latency. The other outputs, average contention delay, static energy and dynamic energy show the similar result as we discuss here. Therefore we can generalize the conclusion from this output to the other outputs.

1) Radix:

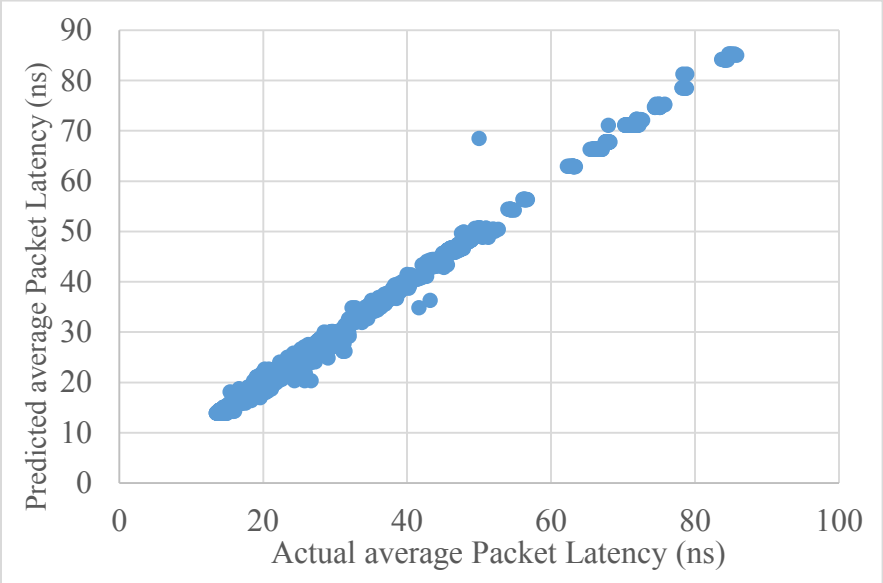


Figure 30: Actual value vs. Predicted value for average packet latency in Radix

Figure 30 shows actual values vs. predicted values for average packet latency in Radix application when the prediction model is REPTree. Each dot indicates one configuration and its actual value that we got for average packet latency from simulation versus its predicted value for average packet latency by our predicted model REP tree. As it indicates the actual value and predicted values (aka dots) lie along a line, and it means the values of these two variables are very close (i.e. $y \approx x$).

Figure 31 show the difference between predicted value and the actual value for each configuration that we had in our dataset (for average packet latency in Radix application when the prediction model is REPTree.). The absolute difference between them is less than one for most of the configurations.

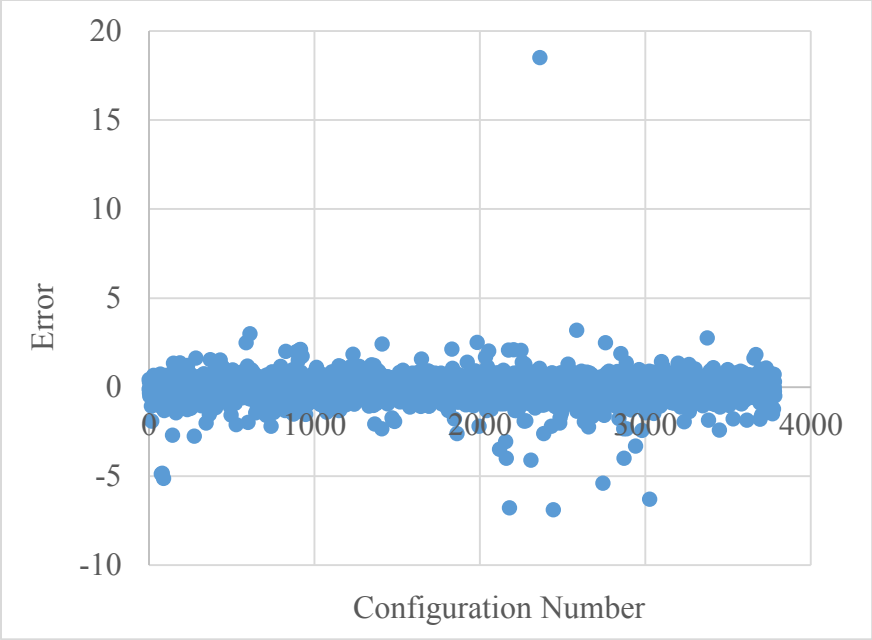


Figure 31: Absolute Error for each configuration in Radix

2) Barnes:

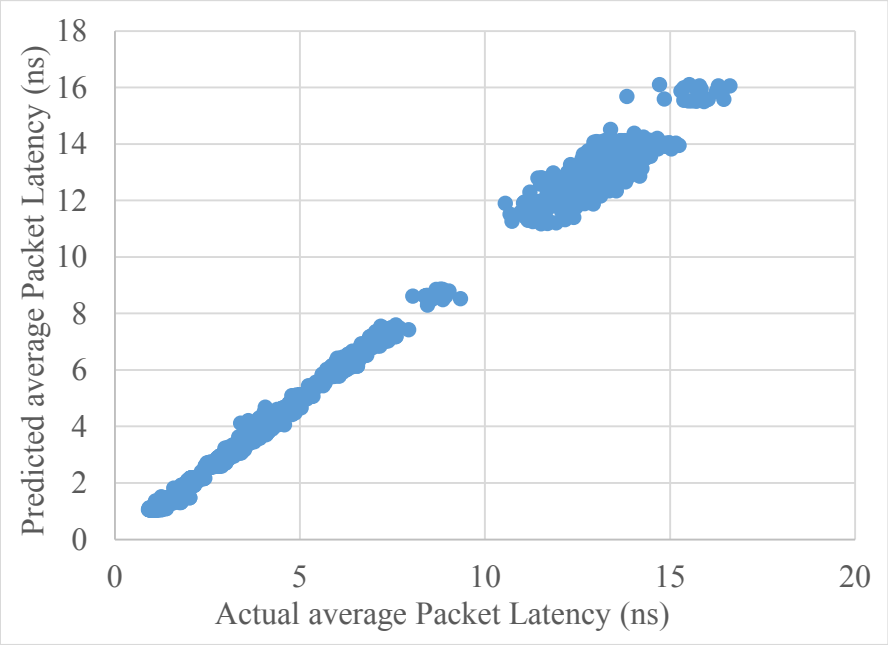


Figure 32: Actual value vs. Predicted value for average packet latency in Barnes

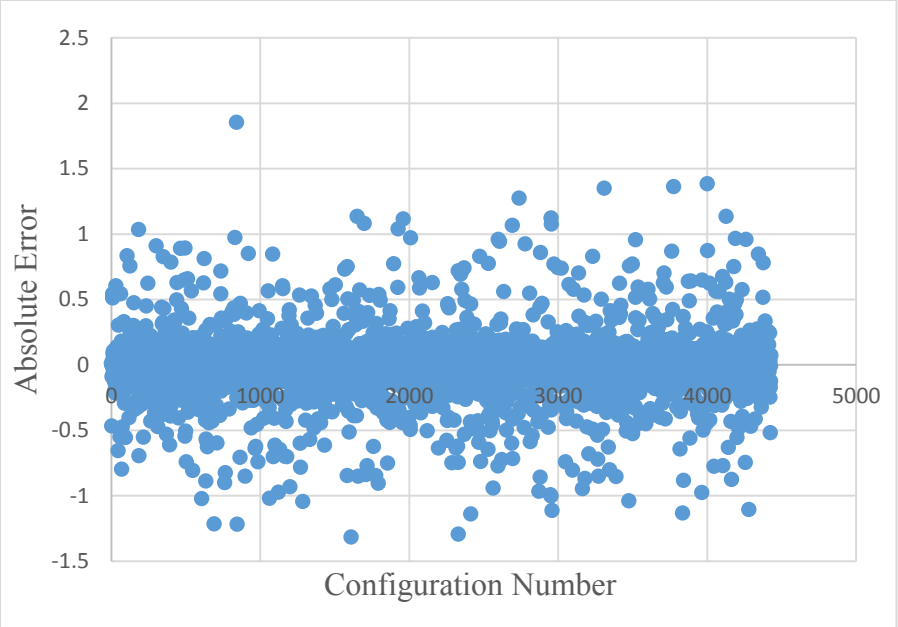


Figure 33: Absolute Error for each configuration in Barnes

Figure 32 shows actual values vs. predicted values for average packet latency in Barnes application when the prediction model is REPTree. As it indicates the actual value and predicted values almost lie along a line, and it means the values of these two variables are very close.

Figure 33 show the difference between predicted value and the actual value for each configuration that we had in our dataset (for average packet latency in Barnes application when the prediction model is REPTree.). The absolute difference between them is less than 1.5 for most of the configurations.

3) Ocean:

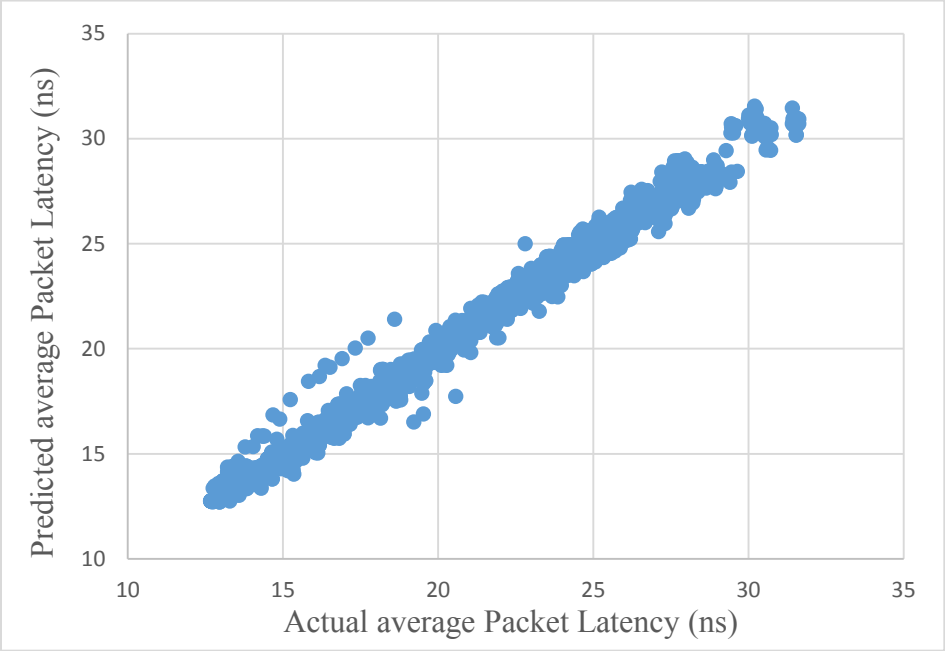


Figure 34: Actual value vs. Predicted value for average packet latency in Ocean

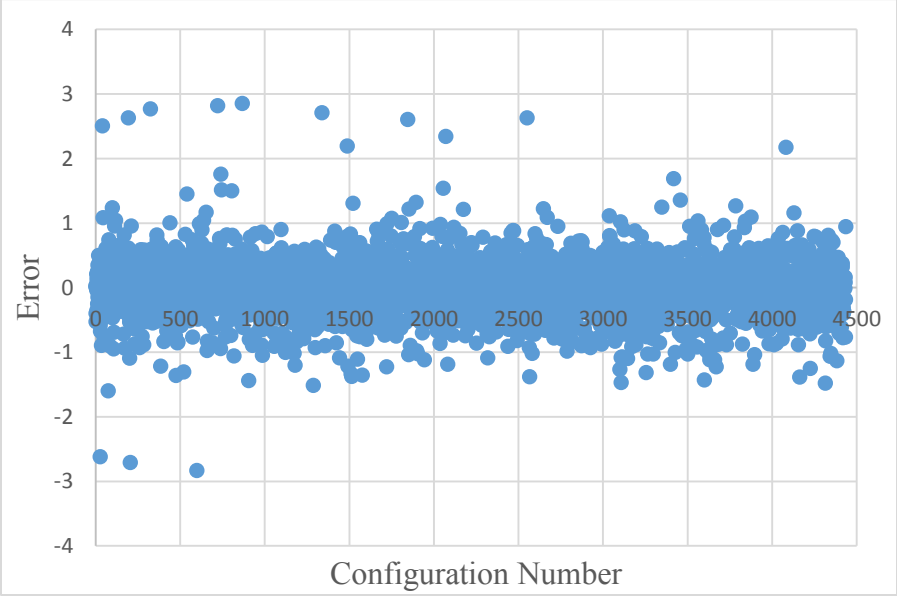


Figure 35: Absolute Error for each configuration in Ocean

Figure 34 shows actual values vs. predicted values for average packet latency in Ocean application when the prediction model is REPTree. As it indicates the actual value and predicted values almost lie along a line, and it means the values of these two variables are very close.

Figure 35 show the difference between predicted value and the actual value for each configuration that we had in our dataset (for average packet latency in Ocean application when the prediction model is REPTree.). The absolute difference between them is less than 3 for almost all the configurations.

5.7.2. Distribution of prediction error

Figures 36, 37, 38 show Frequency of error's values that occur in prediction. Ranges of values are grouped into bins.

1) Radix:

As Figure 36 shows the most frequent error values for average packet latency in Radix benchmark are in the range [-0.05, 0.1].

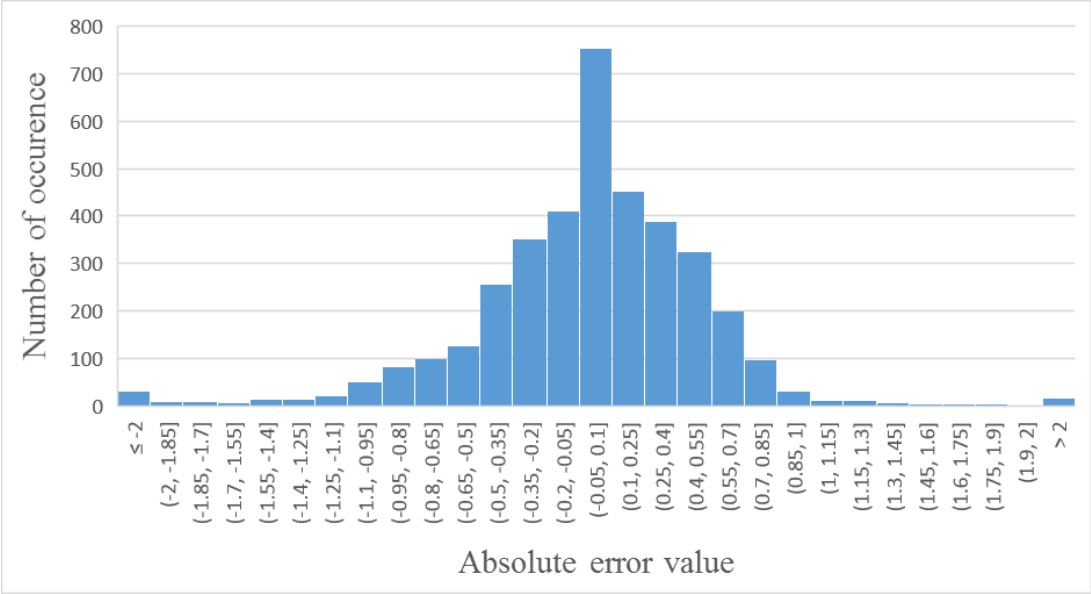


Figure 36: Histogram of Absolute Error for Radix

2) Barnes:

As Figure 37 shows the most frequent error values for average packet latency in Barnes benchmark are in the range [-0.026, 0.02].

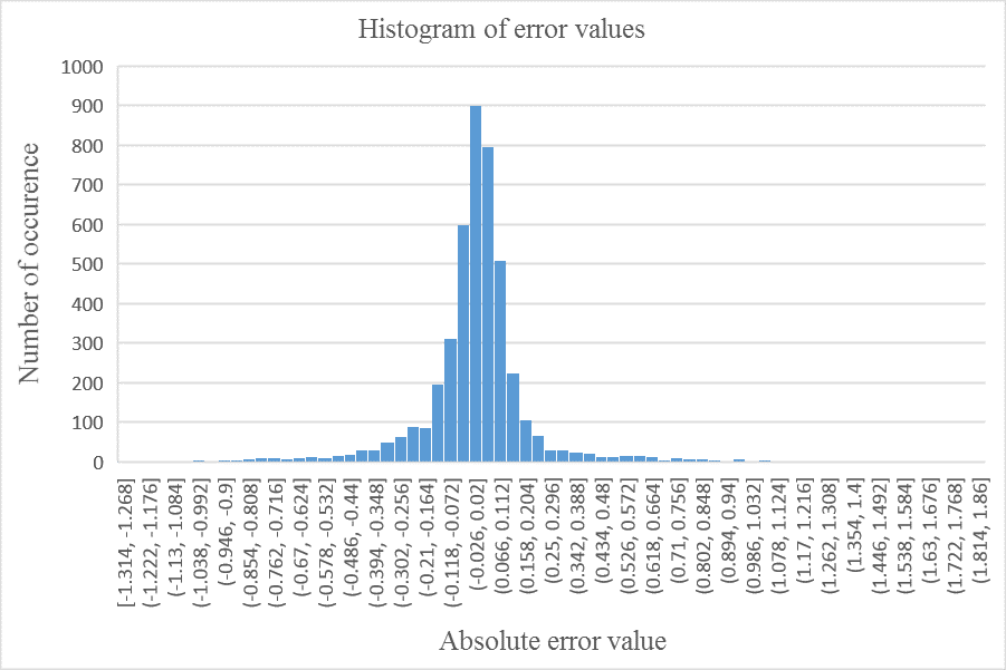


Figure 37: Histogram of Absolute Error for Barnes

3) Ocean:

As Figure 38 shows the most frequent error values for average packet latency in Ocean benchmark are in the range [-0.064, 0.014].

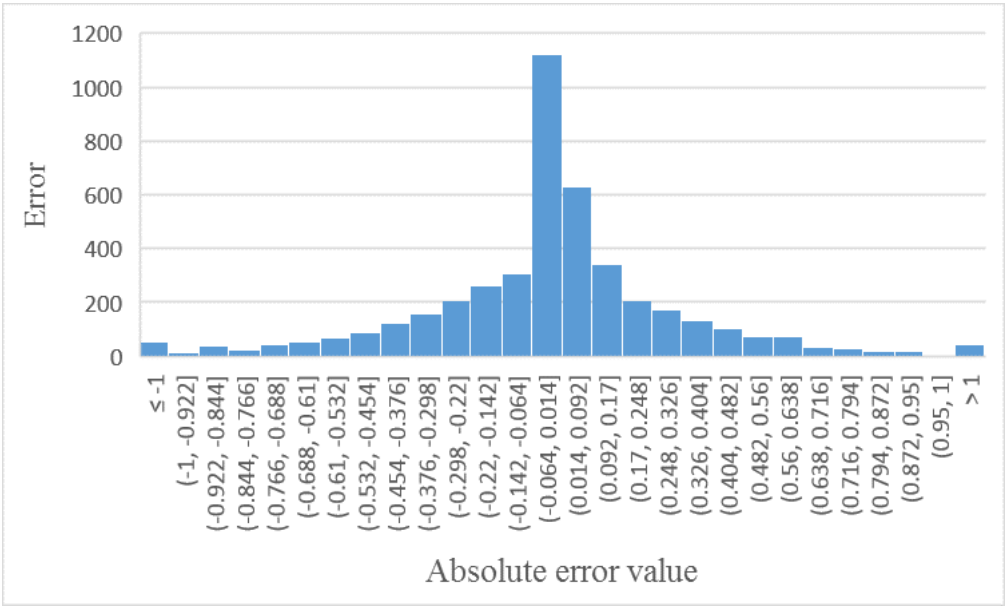


Figure 38: Histogram of Absolute Error for Ocean

5.8. Architecture Comparison

In this section, we want to discuss the impact of each design parameters to the delay and energy consumption of the network. The result of this discussion can be used by the designer as a guideline to be aware of different tradeoffs in design alternatives. We use our massive dataset from simulating of many design alternatives to do this analysis.

5.8.1. Cluster size

In this section, we evaluate the effect of cluster size in our four outputs. As Figure 39 indicates, with having the same total number, by increasing the cluster size in network design, which means having a smaller number of clusters with the larger number of cores inside each, static energy consumption decreases. The reason is that by having bigger cluster size, we will have a fewer number of clusters. It means that the number of receiving networks, optical access points, optical hubs, modulators, photodetectors, serializer, deserializer, etc. that assigned to each cluster decrease, and it causes less energy consumption in overall (both optical and electrical).

Although we show the result of total 64 for the number of cores, same behaviors can be seen in 256 for the number of cores. Moreover, in each experiment we kept all other parameters such as the number of the access point, number of receive network, laser type, etc. fixed, and only change the cluster size.

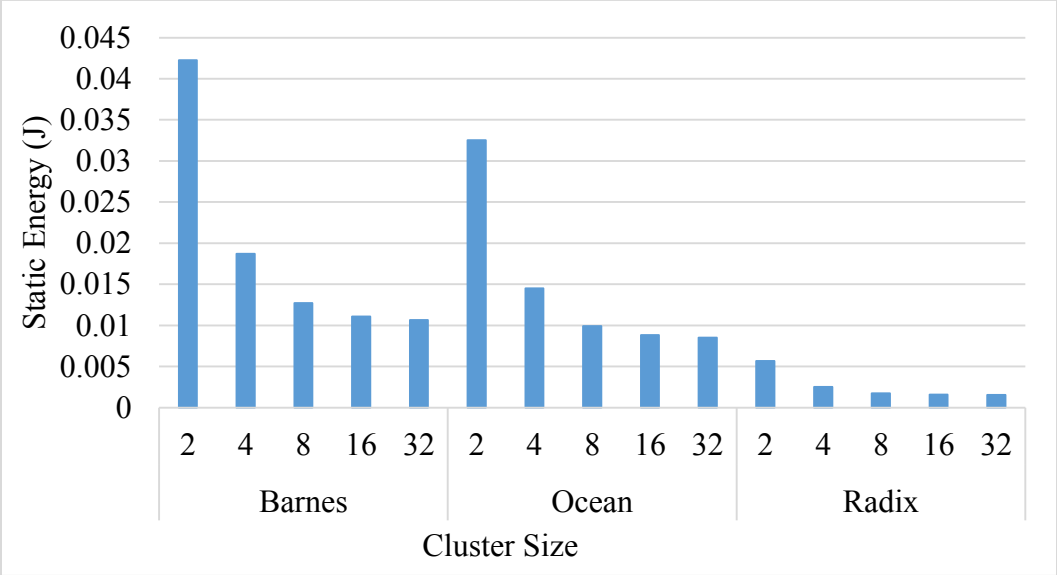


Figure 39: Cluster size vs. Static energy (#cores 64)

Figure 40 shows the trend for dynamic energy consumption. It indicates that for the cluster size of more than 2, the dynamic energy consumption is almost the same. This is because decreasing in the number of mentioned modules has the most impact on the static energy consumption that is data independent, not in dynamic energy consumption which is data dependent and data traffic not change here.

Static energy is in the range of 0.0015 J to 0.042 J, but dynamic energy is in the range of 0.0034 J to 0.0005 J. Therefore static energy consumption is the most influential in the total energy consumption of the network.

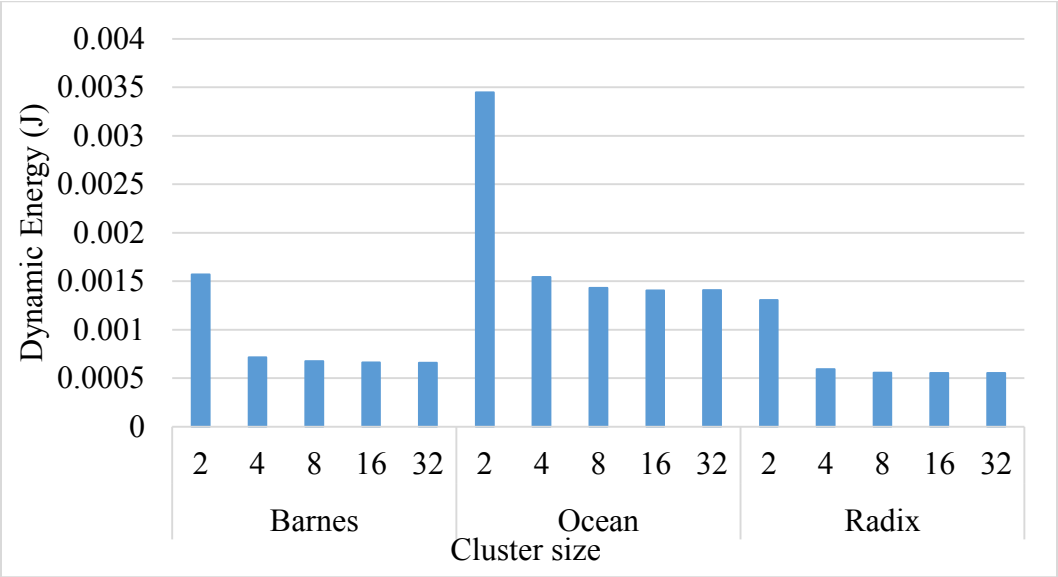


Figure 40: Cluster size vs. Dynamic energy (#cores 64)

As Figure 41 indicates, by increasing the cluster size in network design, which means having a smaller number of clusters with more number of cores in each of them, Average Packet Latency increase. It is because that by increasing the cluster size, the number of cores inside each cluster increased and it causes that all of the cores want to use network components such as access point, receiving networks, an optical hub, shared memory which leads to larger waiting time for accessing these shared components. Figure 42 shows the same result for Average Contention Delay.

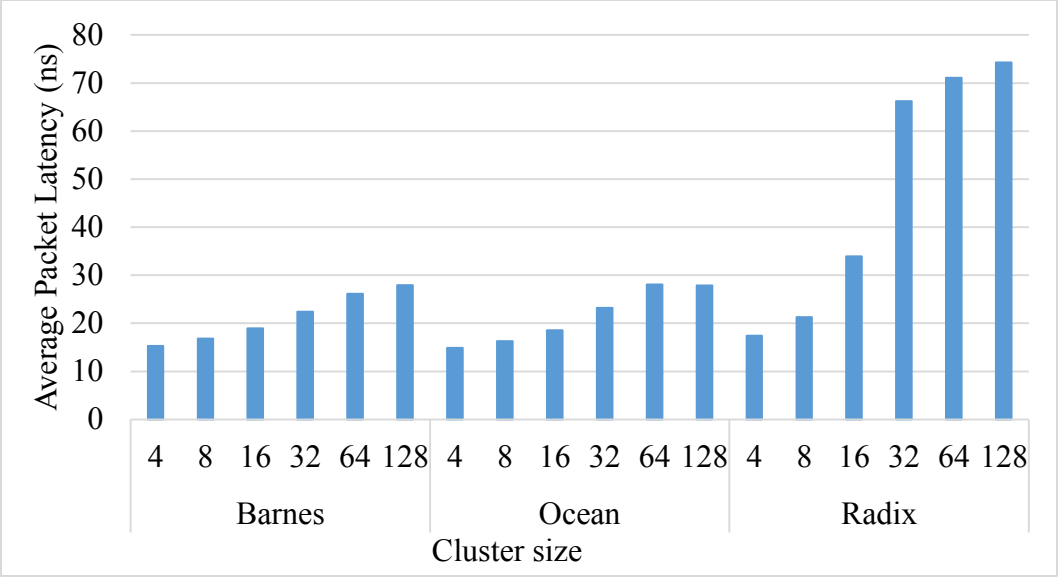


Figure 41: Cluster size vs. average Packet Latency (#cores 256)

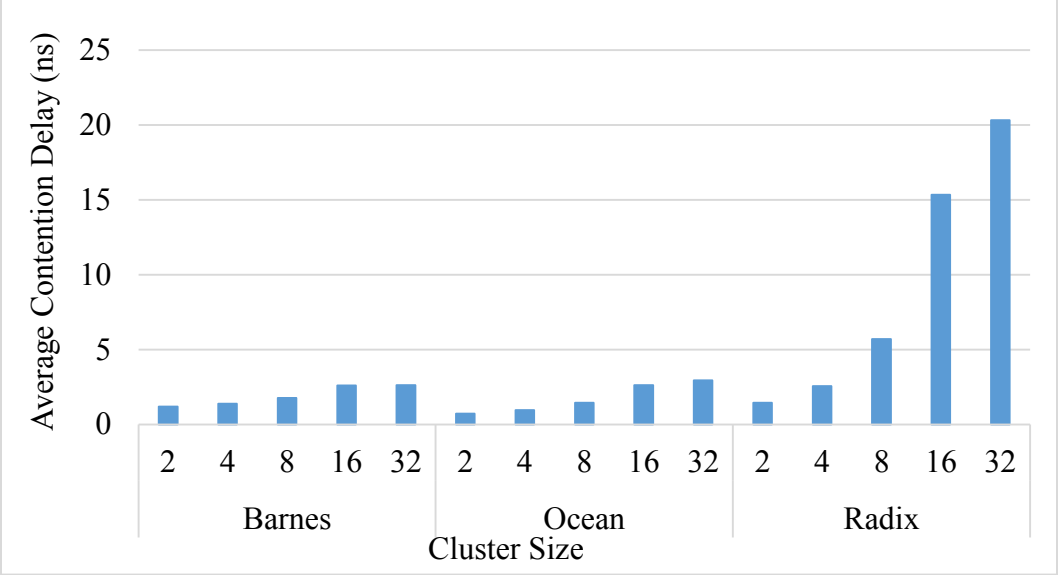


Figure 42: Cluster size vs. average Contention Delay (#cores 64)

5.8.2. Receiving network

As Figure 43 indicates, by increasing the number of receiving networks, static energy consumption increases. This is because, as we discuss in section 2.4, receiving network is a demultiplexer, and by increasing the number of demultiplexers in each cluster (aka increasing number of receiving network), energy consumption increases. Figure 44 shows the trend for dynamic energy consumption, and although the dynamic energy increases by increasing the

number of receiving networks, the range of this increase is smaller than static energy due to data dependency of dynamic energy. In each experiment, we kept all other parameters such as the number of access points, the number of cores, cluster size, laser type, etc. fixed, and only change the number of receiving networks.

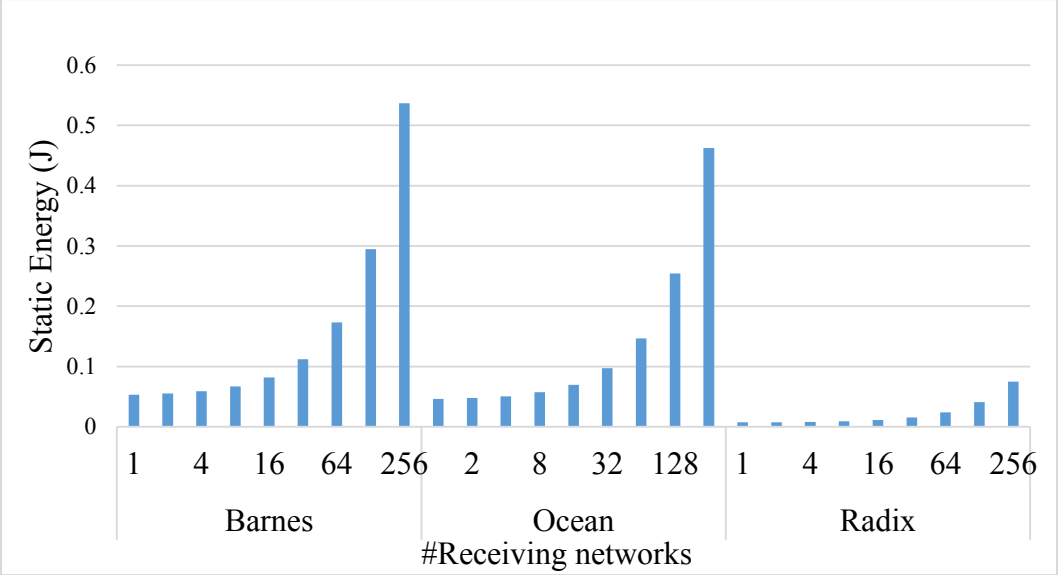


Figure 43: Number of receiving network vs. static energy

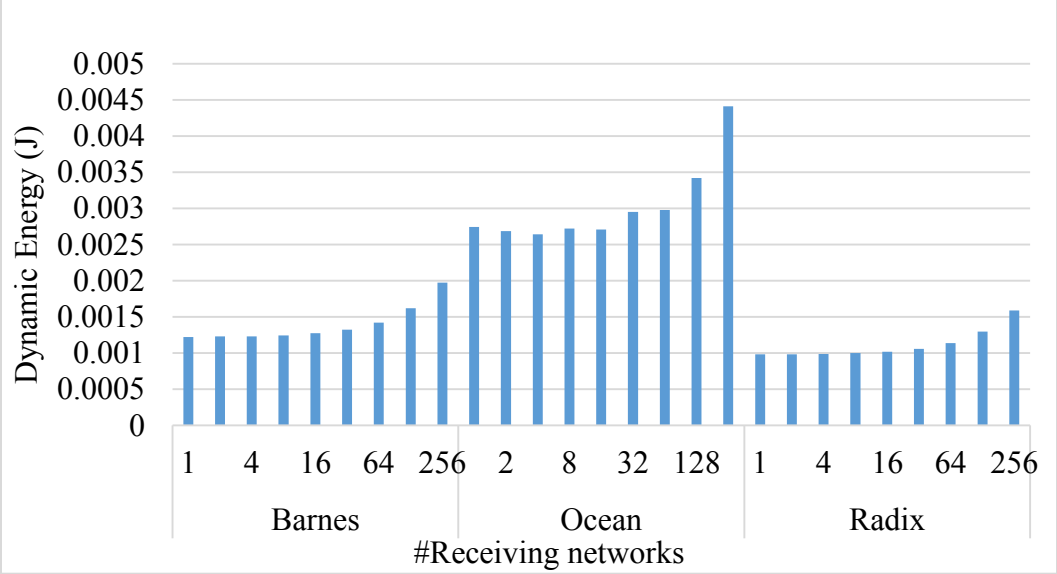


Figure 44: Number of receiving network vs. dynamic energy

Figure 45 and 46 show, by increasing the number of receiving networks, packet latency and contention delay of network decrease, and it was expected because the packets waiting time will decrease. But decreasing in latency and delay is much less compare to increasing in energy consumption. The relative decrease in latency is 5%, but for energy, it is 90%, both compared to 1 receiving network per cluster.

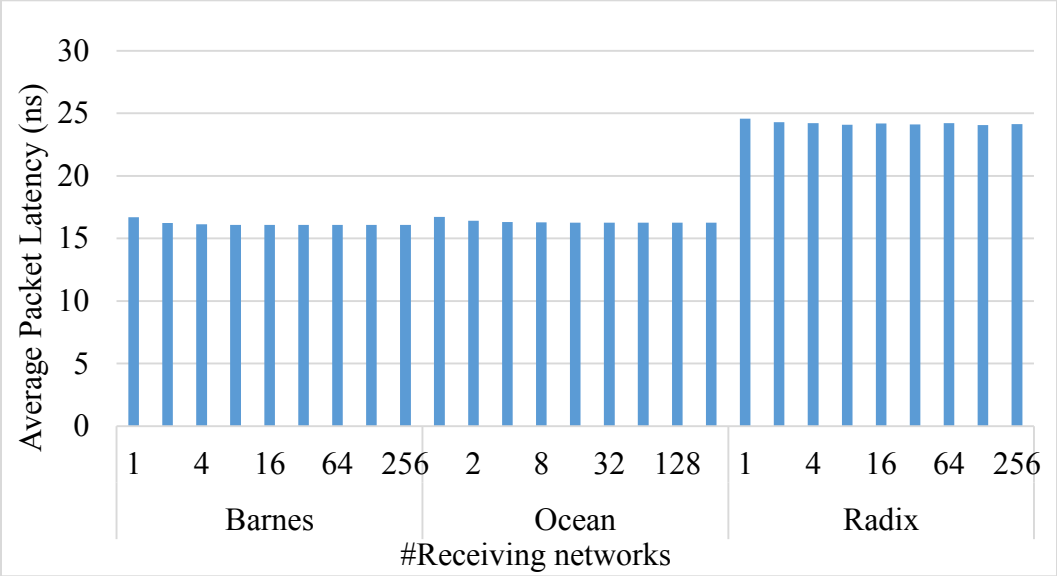


Figure 45: Number of receiving network vs. average packet latency

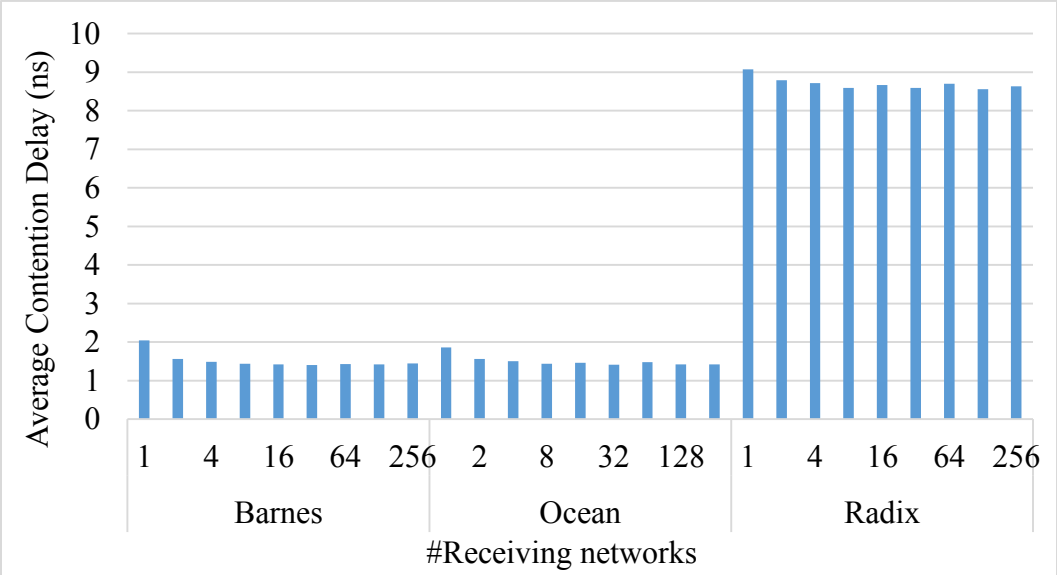


Figure 46: Number of receiving network vs. average contention delay

5.8.3. Access point

As Figure 47 indicates, by increasing the number of optical access points, static energy consumption increase, and it is expected because we added extra modules to each cluster. In each experiment, we kept all other parameters such as the number of cores, the number of receiving networks, cluster size, laser type, etc. fixed, and only change the number of access points.

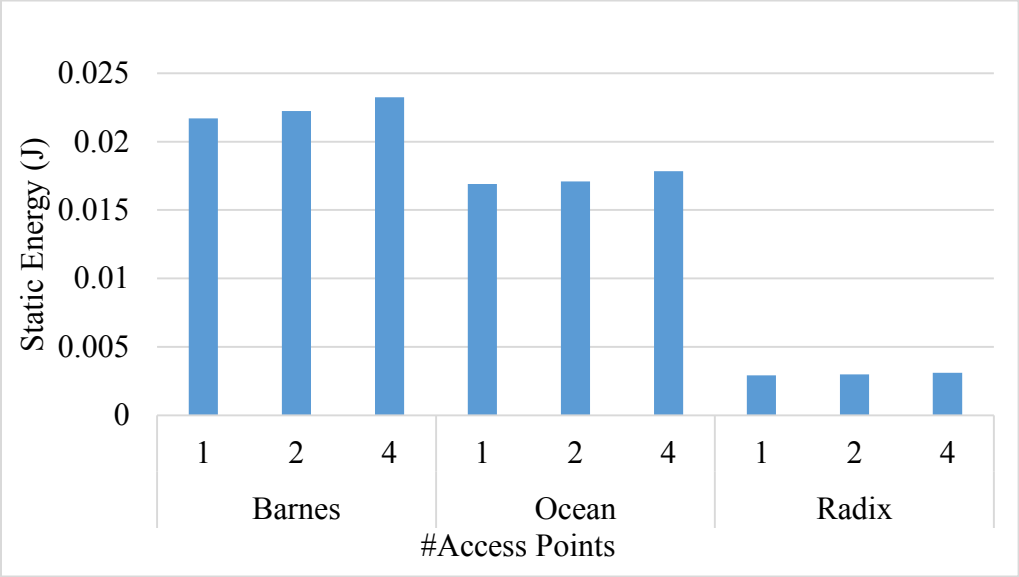


Figure 47: Number of Access points vs. Static energy

As Figure 48 shows, dynamic energy consumption decreases by increasing the number of access points. But this decrease is very negligible.

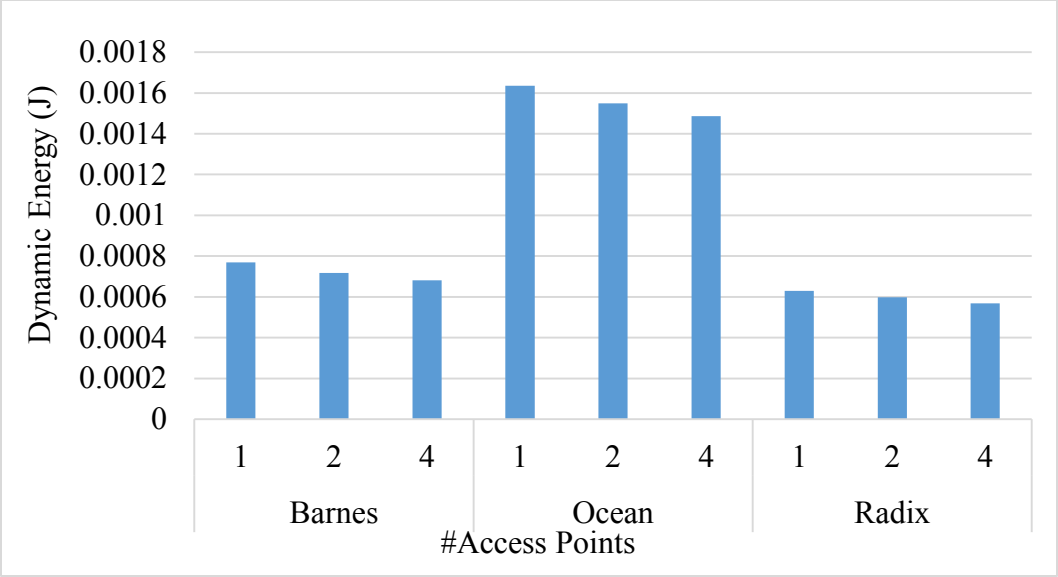


Figure 48: Number of Access Points vs. Dynamic energy

Figure 49 and 50 show, by increasing the number of access points, packet latency and contention delay of network decrease, and it was expected because the packets waiting time will decrease.

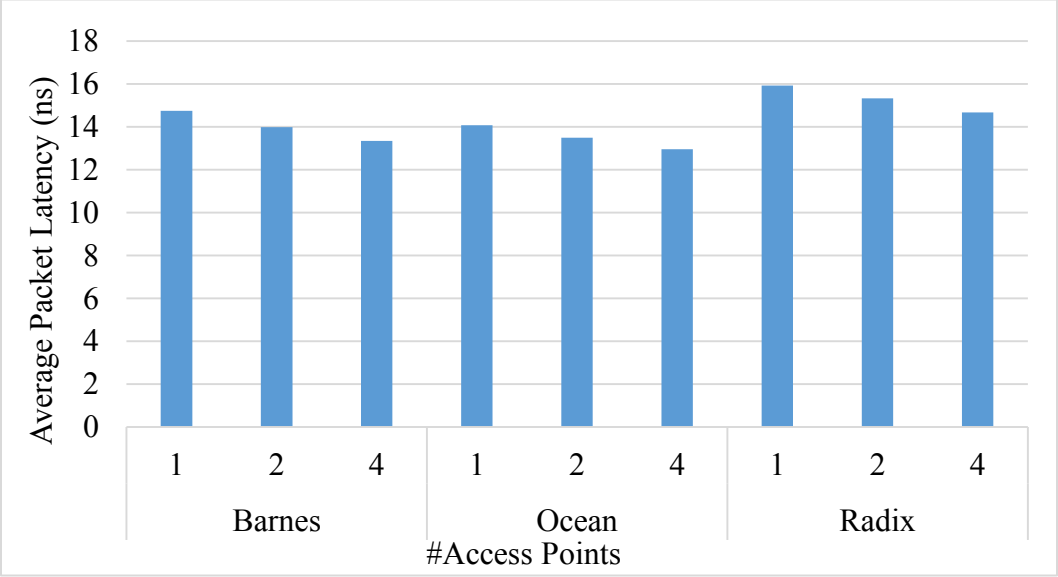


Figure 49: Number of Access Points vs. average packet latency

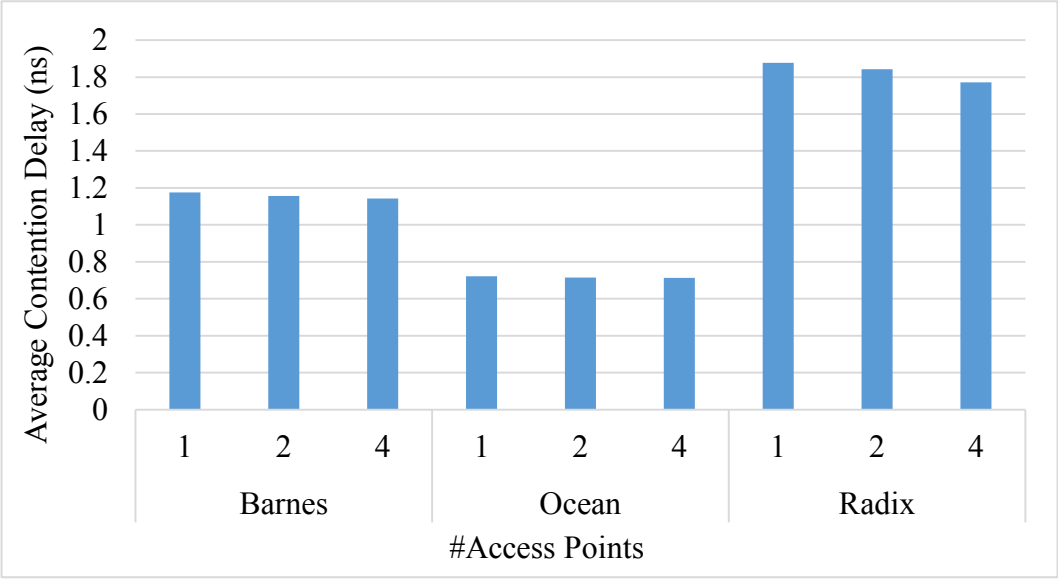


Figure 50: Number of Access Points vs. average contention delay

5.8.4. Laser

For evaluating the effect of Laser in delay and energy consumption of the network, we keep three out of four parameters fixed and change the remained one. For example, we keep these parameters fixed: the number of cores, the number of the optical access point, cluster size, and we just change the number of receiving the network. We also repeat the study for other combinations too, and they show the same result because comparing two type of laser is independent of other design parameters. For example, we keep these parameters fixed: the number of cores, the number of receiving network, cluster size, and we just change the number of the optical access point.

As Figure 51 shows, static energy consumption of standard laser is greater than the throttled laser. We expect to see this result because the advantage of throttled laser to the standard laser is that throttled laser consumes less energy for turning on and off.

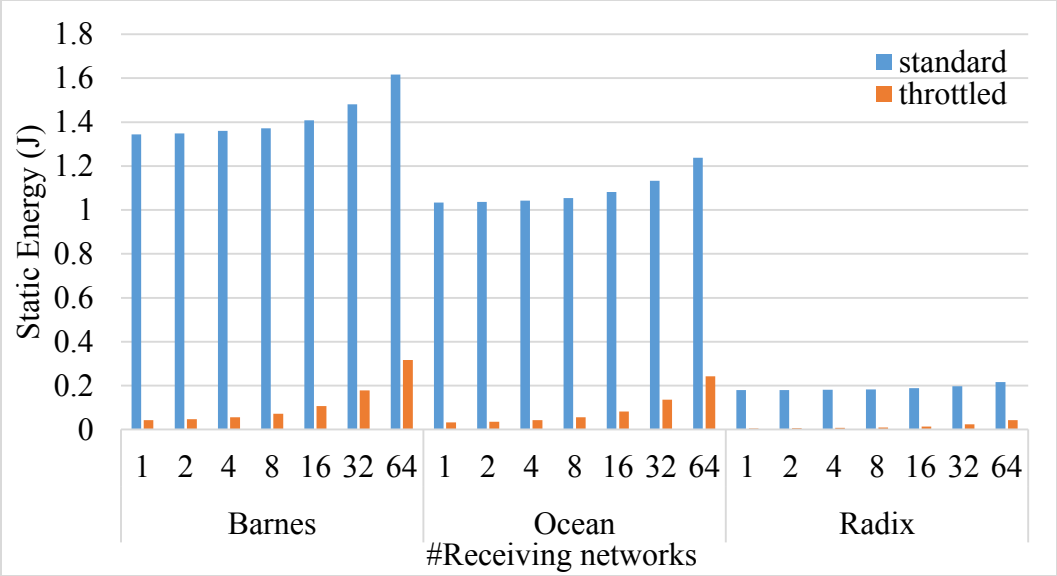


Figure 51: Laser Type vs. Static energy

As Figure 52 shows there is almost no difference between standard laser and throttled laser in dynamic energy consumption because of the laser in a non-data dependent component, therefore, there is no difference between these two lasers in this aspect.

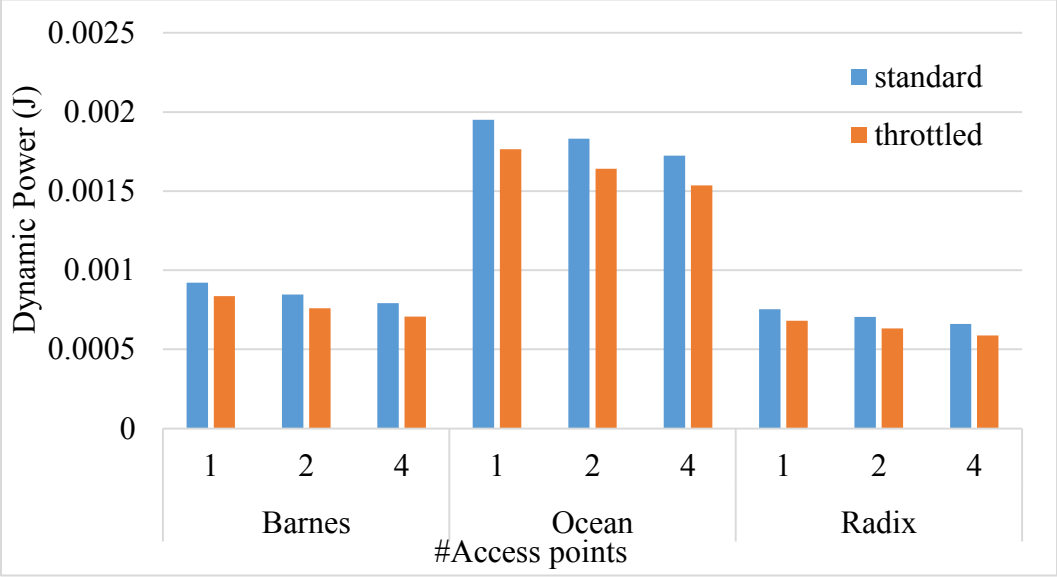


Figure 52: Laser type vs Dynamic energy

As Figure 53 indicates, average packet latency is almost the same for both types of lasers. Figure 54 shows the same result for contention delay too. Because the type of laser does not affect latency and delay on the network.

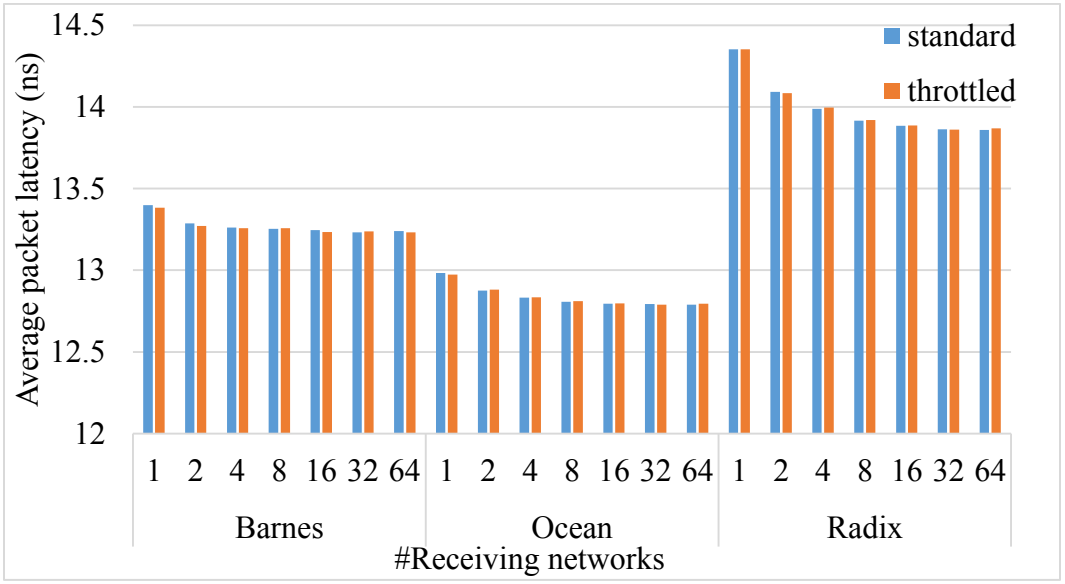


Figure 53: Laser type vs. Average packet latency

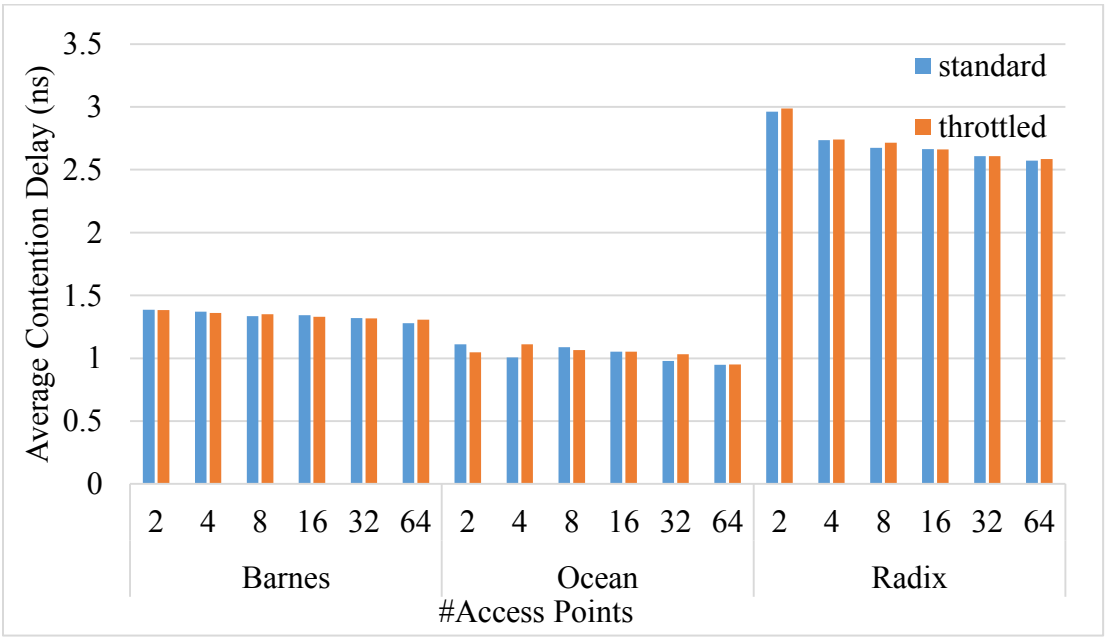


Figure 54: Laser type vs. Average contention Delay

CHAPTER 6

6. Conclusion and Future Work

In this thesis, we have presented a prediction modeling technique for design space exploration of the optical network on chip. Our proposed model accurately predicts *delay* (includes network packet latency and network contention delay) and *energy* (includes static and dynamic energy consumption) of the network with average error rates (root relative squared error) as low as 4.27%. The advantage of our prediction model is that it addresses fundamental challenges of accurately estimating desired metrics without having to incur high simulation cost of the optical network on chip architecture. We reduced the number of required simulations by accurately selecting the parameters that have the most impact on network and then sparsely and randomly sampling the designs build with these parameters from an Optical Network on Chip (ONoC) design space and simulate them. We validated our model with three different applications executing on a large set of network configurations in a large optical network on chip design space. We presented several prediction models include regression models, neural networks models, and tree models and compare their prediction accuracy together. We conclude that tree models have the most accurate prediction in our problem. We also applied several processing techniques in our dataset and demonstrated the improvement they caused in the prediction accuracy. Moreover, we discussed the impact of each design parameters in delay and energy consumption of network. The result of this discussion can be used by the designer as a guideline of different tradeoffs in the design process.

Future direction to improve on prediction modeling technique will focus on providing a full customized regression function that extracts all the interaction among features and output accurately. We can extend our current architecture for the network which is ATAC to the other optical network on chip architectures, including ORNoC and Corona. We can also include memory and core parameters of the system on chip in our design space exploration to fully analyze the system, not only the network portion. Moreover, our next project for extending the current one is to present a recommendation system that can suggest a design configuration to the user based on constraints that user sets for the value of an output.

APPENDIX

A. M5P tree model for Dynamic energy in Ocean benchmark

Figure 55 shows M5P tree structure for Dynamic energy for Ocean benchmark, followed by the tree structure, with level by level representation of the tree structure in text format, as well as the linear model in each leaf. Each tab indicates one level of the tree. The values after ":" indicate the predicted value and values on bracket indicate number of instances that reach that node and its squared error, respectively.

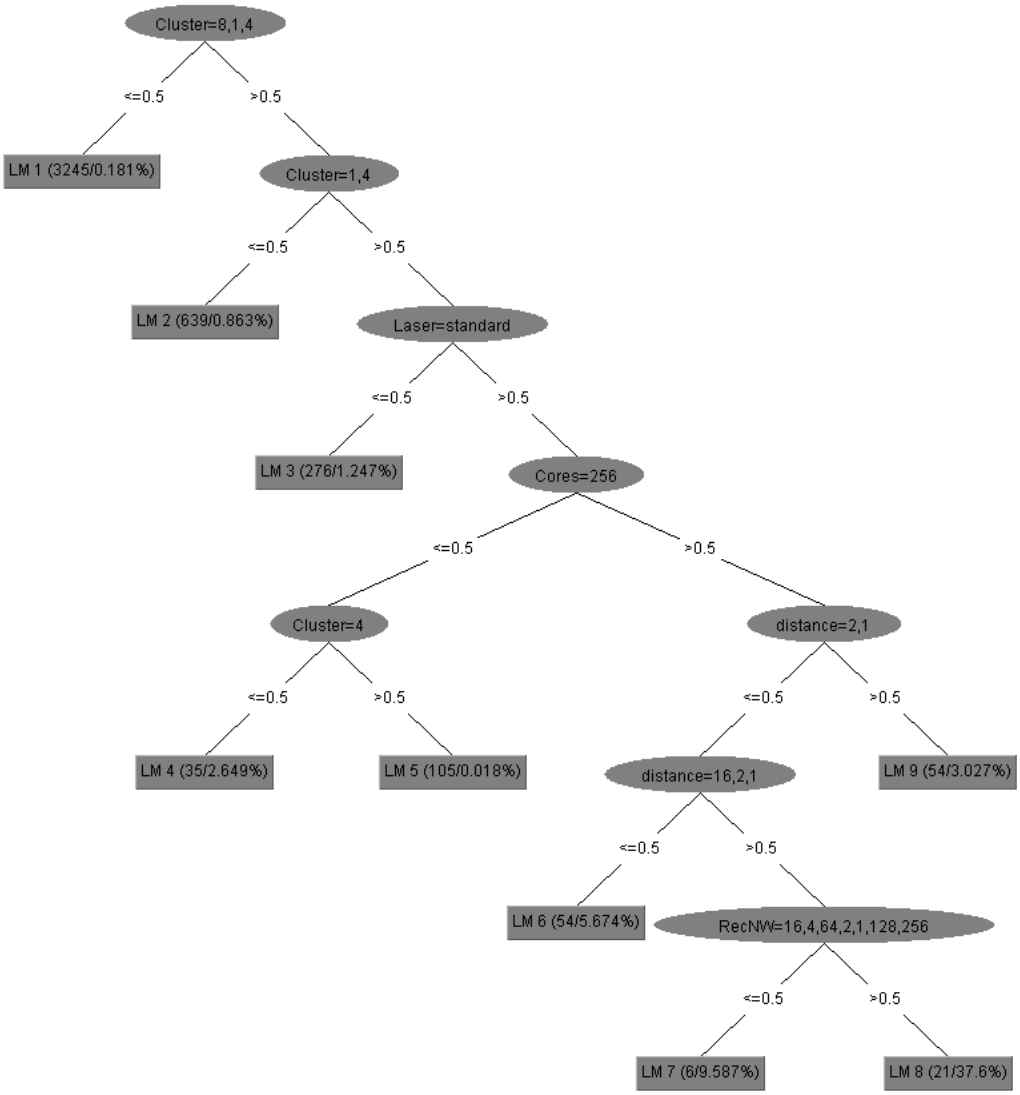


Figure 55: M5P tree for Dynamic energy in Ocean

=== Run information ===

trees.m5PTree

Ocean Nominal dataset

Features:

Cores

Cluster

AccP

RecNW

Laser

Routing

distance

Output:

PowerStat

M5P tree (using smoothed linear models) Structures:

==========

Cluster=8,1,4 <= 0.5 : LM1 (3245/0.181%)

Cluster=8,1,4 > 0.5 :

| Cluster=1,4 <= 0.5 : LM2 (639/0.863%)

| Cluster=1,4 > 0.5 :

| | Laser=standard <= 0.5 : LM3 (276/1.247%)

| | Laser=standard > 0.5 :

| | | Cores=256 <= 0.5 :

| | | | Cluster=4 <= 0.5 : LM4 (35/2.649%)

| | | | Cluster=4 > 0.5 : LM5 (105/0.018%)

| | | Cores=256 > 0.5 :

| | | | distance=2,1 <= 0.5 :

| | | | | distance=16,2,1 <= 0.5 : LM6 (54/5.674%)

| | | | | distance=16,2,1 > 0.5 :

| | | | | | RecNW=16,4,64,2,1,128,256 <= 0.5 : LM7 (6/9.587%)

| | | | | | RecNW=16,4,64,2,1,128,256 > 0.5 : LM8 (21/37.6%)

| | | | distance=2,1 > 0.5 : LM9 (54/3.027%)

LM num: 1

PowerStat =

0.1856 * Cores=256

+ 0.0318 * Cluster=8,1,4

+ 0.7599 * Cluster=1,4

+ 0.0498 * Cluster=4

+ 0.1975 * Laser=standard

- 0.1006

LM num: 2

PowerStat =

3.5778 * Cores=256

+ 0.086 * Cluster=8,1,4

+ 7.0164 * Cluster=1,4

- 0.8306 * Cluster=4

+ 4.1515 * Laser=standard

- 3.4788

LM num: 3

PowerStat =

19.5948 * Cores=256

+ 0.086 * Cluster=8,1,4

+ 7.7877 * Cluster=1,4

- 8.0288 * Cluster=4

+ 21.7151 * Laser=standard

- 5.4426

LM num: 4

PowerStat =

74.9037 * Cores=256

+ 0.086 * Cluster=8,1,4

+ 7.7877 * Cluster=1,4

- 103.8246 * Cluster=4

+ 0.6195 * RecNW=2,1,128,256

+ 21.7738 * Laser=standard

+ 13.1143 * distance=8,16,2,1

+ 3.785 * distance=16,2,1

- 16.2493 * distance=2,1

+ 247.936

LM num: 5

PowerStat =

74.9037 * Cores=256

+ 0.086 * Cluster=8,1,4

+ 7.7877 * Cluster=1,4

- 62.6597 * Cluster=4

+ 0.6195 * RecNW=2,1,128,256

+ 21.7738 * Laser=standard

+ 1.2577 * distance=8,16,2,1

+ 3.785 * distance=16,2,1

- 4.9132 * distance=2,1

+ 47.0621

LM num: 6

PowerStat =

76.7455 * Cores=256

+ 0.086 * Cluster=8,1,4

+ 7.7877 * Cluster=1,4

- 34.0962 * Cluster=4

- 3.7366 * AccP=4,1

+ 10.3337 * AccP=1

+ 5.5243 * RecNW=2,1,128,256

+ 21.7738 * Laser=standard

+ 11.9508 * distance=8,16,2,1

+ 27.6531 * distance=16,2,1

- 16.9606 * distance=2,1

+ 511.14

LM num: 7

PowerStat =

76.7455 * Cores=256

+ 0.086 * Cluster=8,1,4

+ 7.7877 * Cluster=1,4

- 34.0962 * Cluster=4

- 6.4902 * AccP=1

+ 12.9561 * RecNW=16,4,64,2,1,128,256

+ 18.843 * RecNW=2,1,128,256

+ 8.733 * RecNW=1,128,256

- 9.7924 * RecNW=128,256

+ 11.8316 * RecNW=256

+ 21.7738 * Laser=standard

+ 5.8941 * distance=8,16,2,1

+ 35.8949 * distance=16,2,1

- 16.9606 * distance=2,1

+ 533.329

LM num: 8

PowerStat =

76.7455 * Cores=256

+ 0.086 * Cluster=8,1,4

+ 7.7877 * Cluster=1,4

- 34.0962 * Cluster=4

- 3.3842 * AccP=1

+ 7.5577 * RecNW=16,4,64,2,1,128,256

+ 27.1103 * RecNW=2,1,128,256

+ 5.0943 * RecNW=1,128,256

- 5.7122 * RecNW=128,256

+ 6.9018 * RecNW=256

+ 21.7738 * Laser=standard

+ 5.8941 * distance=8,16,2,1

+ 35.8949 * distance=16,2,1

- 16.9606 * distance=2,1

+ 542.8745

LM num: 9

PowerStat =

76.7455 * Cores=256

+ 0.086 * Cluster=8,1,4

+ 7.7877 * Cluster=1,4

- 34.0962 * Cluster=4

- 8.5066 * AccP=4,1

+ 16.3077 * AccP=1

+ 2.9652 * RecNW=2,1,128,256

+ 21.7738 * Laser=standard

+ 3.212 * distance=8,16,2,1

+ 19.106 * distance=16,2,1

- 21.8034 * distance=2,1

+ 514.9184

Number of Linear Models : 9

Time taken to build model: 0.33 seconds

=== Cross-validation summary ==========

Correlation coefficient	0.9986
Mean absolute error	2.2005
Root mean squared error	6.9993
Relative absolute error	5.3591 %
Root relative squared error	6.329 %

B. REP tree model for Dynamic energy in Ocean benchmark

We cannot show REP tree structure here, because it is too big. We only show tree structure level by level in text format. Each tab indicate on level in tree. The values after ":" indicate the predicted value and values on bracket indicate number of instances reach to that node and its squared error, respectively.

=== Run information ===

trees.REPTree

Ocean Nominal dataset

Features:

- #Cores
- Cluster Size
- #AccP
- #RecNW
- Laser Type
- Routing Strategy
- distance

Output:

- PowerStat

REPTree Structure:

```
==========================================

Cluster = 1

|  Laser = throttled : 0.44 (25/0.15)

|  Laser = standard

|  |  distance = 1 : 280.09 (4/0.85)

|  |  distance = 2 : 276.92 (4/0.31)

|  |  distance = 4 : 277.66 (5/0.08)

|  |  distance = 6 : 285.74 (0/0)

|  |  distance = 8 : 292.77 (5/0.39)

|  |  distance = 10 : 285.74 (0/0)

|  |  distance = 14 : 285.74 (0/0)

|  |  distance = 16 : 301.26 (7/0.29)

|  |  distance = 22 : 285.74 (0/0)

Cluster = 2 : 0.61 (95/0.27)

Cluster = 4

|  Cores = 64 : 0.05 (152/0)

|  Cores = 256

|  |  Laser = throttled : 1.37 (76/3.31)

|  |  Laser = standard

|  |  |  distance = 1 : 627.49 (0/0)

|  |  |  distance = 2

|  |  |  |  AccP = 1 : 617.48 (12/7.14)

|  |  |  |  AccP = 2 : 607.43 (12/9.51)

|  |  |  |  AccP = 4 : 594.18 (11/9.11)

|  |  |  |  AccP = 8 : 606.36 (0/0)

|  |  |  |  AccP = 16 : 606.36 (0/0)

|  |  |  |  AccP = 32 : 606.36 (0/0)

|  |  |  |  AccP = 64 : 606.36 (0/0)

|  |  |  |  AccP = 128 : 606.36 (0/0)

|  |  |  distance = 4
```

| | | | AccP = 1 : 614.49 (6/28.03)

| | | | AccP = 2 : 604.08 (6/32.64)

| | | | AccP = 4 : 593.73 (4/11.77)

| | | | AccP = 8 : 604.1 (0/0)

| | | | AccP = 16 : 604.1 (0/0)

| | | | AccP = 32 : 604.1 (0/0)

| | | | AccP = 64 : 604.1 (0/0)

| | | | AccP = 128 : 604.1 (0/0)

| | | distance = 6 : 627.49 (0/0)

| | | distance = 8

| | | | AccP = 1 : 625.78 (4/49.37)

| | | | AccP = 2 : 615.4 (5/30.2)

| | | | AccP = 4 : 611.95 (3/23.34)

| | | | AccP = 8 : 617.71 (0/0)

| | | | AccP = 16 : 617.71 (0/0)

| | | | AccP = 32 : 617.71 (0/0)

| | | | AccP = 64 : 617.71 (0/0)

| | | | AccP = 128 : 617.71 (0/0)

| | | distance = 10 : 627.49 (0/0)

| | | distance = 14 : 627.49 (0/0)

| | | distance = 16 : 702.9 (17/2539.92)

| | | distance = 22 : 627.49 (0/0)

Cluster = 8 : 0.7 (408/0.88)

Cluster = 16 : 0.1 (546/0.02)

Cluster = 32 : 0.07 (639/0.01)

Cluster = 64 : 0.11 (413/0.01)

Cluster = 128 : 0.18 (497/0.04)

==========

Size of the tree : 57

Time taken to build model: 0.03 seconds

=== Cross-validation summary ==========

Correlation coefficient 0.9992

Mean absolute error 0.6749

Root mean squared error 4.5521

Relative absolute error 1.6435 %

Root relative squared error 4.1161 %

C. Prediction accuracy for different sample sizes

1- Radix:

Table 39: Prediction accuracy of different sample size for avg contention delay in Radix

%	CC	MAE	RMSE	RAE	RRSE
10	0.9955	1.1085	1.9403	6.10%	9.54%
20	0.9986	0.5995	1.086	3.23%	5.24%
30	0.9991	0.5037	0.8897	2.76%	4.34%
40	0.9992	0.4565	0.7979	2.54%	3.94%
50	0.9995	0.3987	0.66	2.23%	3.27%
60	0.9996	0.386	0.5896	2.15%	2.91%
70	0.9997	0.3665	0.5267	2.04%	2.60%
80	0.9996	0.3782	0.5964	2.10%	2.94%
90	0.9997	0.3537	0.5073	1.96%	2.50%
100	0.9997	0.3622	0.4948	1.99%	2.43%

Table 40: Prediction accuracy of different sample size for static energy in Radix

%	CC	MAE	MSE	RAE	RSE
10	0.775	1.0562	7.3265	39.85%	63.13%
20	0.9962	0.1276	1.2866	2.71%	8.65%
30	1	0.0573	0.1285	0.97%	0.77%
40	1	0.0571	0.1283	1.05%	0.80%
50	1	0.0555	0.1151	0.95%	0.69%
60	1	0.0533	0.1107	0.95%	0.68%
70	1	0.0532	0.112	0.90%	0.67%
80	1	0.0532	0.1119	0.89%	0.66%
90	1	0.0534	0.1151	0.91%	0.69%
100	1	0.052	0.1121	0.93%	0.69%

Table 41: Prediction accuracy of different sample size for dynamic energy in Radix

%	CC	MAE	MSE	RAE	RSE
10	0.6386	0.0077	0.0346	56.12%	78.23%
20	0.9216	0.0028	0.0167	21.09%	38.79%
30	0.9617	0.002	0.0124	14.38%	27.90%
40	0.988	0.0011	0.0069	8.15%	15.70%
50	0.9808	0.0014	0.0084	10.71%	19.53%
60	0.9933	0.0008	0.0048	6.48%	11.58%
70	0.9977	0.0006	0.0028	5.44%	6.79%
80	0.9965	0.0006	0.0034	5.37%	8.33%
90	0.9971	0.0006	0.0032	5.06%	7.59%
100	0.9986	0.0006	0.0022	4.84%	5.26%

2- Barnes:

Table 42: Prediction accuracy of different sample size for avg contention delay in Barnes

%	CC	MAE	MSE	RAE	RSE
10	0.9815	0.1871	0.3357	11.68%	19.11%
20	0.9938	0.1159	0.1945	7.25%	11.11%
30	0.995	0.1045	0.1745	6.49%	9.96%
40	0.9966	0.0899	0.1436	5.57%	8.18%
50	0.997	0.0865	0.134	5.39%	7.68%
60	0.9973	0.0814	0.1285	5.07%	7.35%
70	0.9976	0.0771	0.1215	4.83%	6.98%
80	0.9976	0.0764	0.1203	4.79%	6.91%
90	0.9979	0.0727	0.1131	4.53%	6.48%
100	0.9979	0.0705	0.1124	4.39%	6.43%

Table 43: Prediction accuracy of different sample size for static energy in Barnes

%	CC	MAE	MSE	RAE	RSE
10	0.9999	0.497	1.3424	1.11%	1.11%
20	1	0.4165	0.869	0.95%	0.72%
30	1	0.4142	0.8549	0.92%	0.70%
40	1	0.4306	0.908	0.90%	0.72%
50	1	0.4234	0.8887	0.97%	0.73%
60	1	0.4296	0.9116	0.89%	0.71%
70	1	0.4459	0.9617	0.89%	0.73%
80	1	0.4393	0.9317	0.90%	0.72%
90	1	0.4454	0.9364	0.90%	0.72%
100	1	0.438	0.9206	0.89%	0.71%

Table 44: Prediction accuracy of different sample size for dynamic energy in Barnes

%	CC	MAE	MSE	RAE	RSE
10	0.8776	0.0072	0.0354	25.26%	47.88%
20	0.9923	0.0014	0.0079	6.42%	12.36%
30	0.9883	0.0017	0.0099	7.62%	15.26%
40	0.9959	0.001	0.0058	4.60%	9.06%
50	0.9963	0.0009	0.0055	4.18%	8.61%
60	0.9962	0.0009	0.0055	4.38%	8.69%
70	0.9985	0.0008	0.0036	3.39%	5.52%
80	0.999	0.0007	0.0029	3.16%	4.47%
90	0.9999	0.0006	0.0009	2.50%	1.31%
100	0.9999	0.0006	0.0008	2.55%	1.28%

3- Ocean:

Table 45: Prediction accuracy of different sample size for avg contention delay in Ocean

%	CC	MAE	MSE	RAE	RSE
10	0.9883	0.2646	0.4222	10.44%	15.20%
20	0.9912	0.2252	0.36	9.18%	13.25%
30	0.9925	0.1997	0.3281	8.24%	12.19%
40	0.993	0.1919	0.3166	7.96%	11.82%
50	0.9937	0.1812	0.2996	7.50%	11.16%
60	0.994	0.1694	0.2938	7.04%	10.98%
70	0.9946	0.1601	0.279	6.60%	10.36%
80	0.9948	0.1526	0.2745	6.27%	10.17%
90	0.9952	0.1418	0.2625	5.86%	9.77%
100	0.9956	0.1334	0.253	5.53%	9.42%

Table 46: Prediction accuracy of different sample size for static energy in Ocean

%	CC	MAE	MSE	RAE	RSE
10	0.9893	3.4395	20.2563	5.57%	14.62%
20	0.9949	1.6856	12.1631	3.51%	10.12%
30	0.9989	0.968	5.5916	2.13%	4.78%
40	0.9991	0.7441	4.8666	1.73%	4.26%
50	0.9993	0.687	4.3508	1.58%	3.78%
60	0.9995	0.5969	3.5875	1.39%	3.16%
70	0.9991	0.7042	4.8892	1.62%	4.28%
80	0.9992	0.7093	4.6595	1.66%	4.10%
90	0.9995	0.5582	3.5543	1.30%	3.13%
100	0.9996	0.526	3.2669	1.23%	2.88%

Table 47: Prediction accuracy of different sample size for dynamic energy in Ocean

%	CC	MAE	MSE	RAE	RSE
10	0.792	0.0171	0.0847	38.95%	63.21%
20	0.656	0.0177	0.1014	41.46%	75.80%
30	0.9938	0.0035	0.0168	6.63%	11.13%
40	0.9941	0.0031	0.0157	6.34%	10.83%
50	0.9947	0.003	0.0149	5.87%	10.25%
60	0.9972	0.0026	0.011	5.06%	7.46%
70	0.9978	0.0024	0.0099	4.65%	6.70%
80	0.9982	0.0022	0.0089	4.14%	5.97%
90	0.9972	0.0024	0.0113	4.54%	7.55%
100	0.9978	0.0024	0.0099	4.46%	6.64%

References

[1] G. Kurian, C. Sun, C. H. O. Chen, J. E. Miller, J. Michel, L. Wei, D. A. Antoniadis, L. S. Peh, L. Kimerling, V. Stojanovic, and A. Agarwal, "Cross-layer energy and performance evaluation of a nanophotonic manycore processor system using real application workloads," *Proc. 2012 IEEE 26th Int. Parallel Distrib. Process. Symp. IPDPS 2012*, pp. 1117–1130, 2012.

[2] D. Vantrease, R. Schreiber, M. Monchiero, M. Mclaren, N. P. Jouppi, M. Fiorentino, A. Davis, N. Binkert, R. G. Beausoleil, and J. H. Ahn, "Corona: System implications of emerging nanophotonic technology," *Proc. - Int. Symp. Comput. Archit.*, pp. 153–164, 2008.

[3] G. Hendry, "Architectures and Design Automation for Photonic Networks On Chip," 2011.

[4] S. Le Beux, J. Trajkovic, I. O'Connor, G. Nicolescu, G. Bois, and P. Paulin, "Optical Ring Network-on-Chip (ORNoC): Architecture and design methodology," *2011 Des. Autom. Test Eur.*, pp. 1–6, 2011.

[5] S. Le Beux, J. Trajkovic, I. O'Connor, and G. Nicolescu, "Layout Guidelines for 3D Architectures including Optical Ring Network-on-Chip (ORNoC)," *2011 IEEE/IFIP 19th Int. Conf. VLSI Syst.*, pp. 242–247, 2011.

[6] L. Beux, H. Li, I. O. Connor, K. Cheshmi, X. Liu, J. Trajkovic, G. Nicolescu, S. Le Beux, H. Li, and I. O. Connor, "CHAMELEON : CHANNEL Efficient Optical Network-on-Chip," *IEEE Int. Conf. Des. Autom. Test Eur.*, 2014.

[7] J. Psota, J. Miller, G. Kurian, H. Hoffman, N. Beckmann, J. Eastep, and A. Agarwal, "ATAC: Improving performance and programmability with on-chip optical networks," *ISCAS 2010 - 2010 IEEE Int. Symp. Circuits Syst. Nano-Bio Circuit Fabr. Syst.*, pp. 3325–3328, 2010.

[8] and L. C. K. V. Raghunathan, J. Hu, W. N. Ye, J. Michel, "Athermal Silicon Ring Resonators," *Integr. Photonics Res. Silicon Nanophotonics Photonics Switch. OSA Tech. Dig.*, 2010.

[9] A. B. Kahng, B. Li, and L. Peh, “ORION 2 . 0 : A Power-Area Simulator for Interconnection Networks,” *Tvlsi*, vol. XX, no. 1, pp. 1–5, 2010.

[10] J. Chan, G. Hendry, A. Biberman, K. Bergman, and L. P. Carloni, “PhoenixSim: A simulator for physical-layer analysis of chip-scale photonic interconnection networks,” *Des. Autom. Test Eur. Conf. Exhib. (DATE), 2010*, pp. 691–696, 2010.

[11] A. Shacham, K. Bergman, and L. P. Carloni, “Photonic Network-on-Chip for Future Generations of Chip Multi-Processors,” *IEEE Trans. Comput.*, vol. 57, no. 9, pp. 1246–1260, 2008.

[12] C. Sun, C. H. O. Chen, G. Kurian, L. Wei, J. Miller, A. Agarwal, L. S. Peh, and V. Stojanovic, “DSENT - A tool connecting emerging photonics with electronics for opto-electronic networks-on-chip modeling,” *Proc. 2012 6th IEEE/ACM Int. Symp. Networks-on-Chip, NoCS 2012*, pp. 201–210, 2012.

[13] J. E. Miller, H. Kasture, G. Kurian, C. Gruenwald, N. Beckmann, C. Celio, J. Eastep, and A. Agarwal, “Graphite: A distributed parallel simulator for multicores,” *HPCA - 16 2010 Sixt. Int. Symp. High-Performance Comput. Archit.*, no. January, pp. 1–12, 2010.

[14] B. C. Lee and D. M. Brooks, “Accurate and Efficient Regression Modeling for Microarchitectural Performance and Power Prediction,” *Proc. 12th Int. Conf. Archit. Support Program. Lang. Oper. Syst.*, no. 1, pp. 185–194, 2006.

[15] W. Jia, K. A. Shaw, and M. Martonosi, “Stargazer: Automated regression-based GPU design space exploration,” *ISPASS 2012 - IEEE Int. Symp. Perform. Anal. Syst. Softw.*, pp. 2–13, 2012.

[16] K. Asanovic, B. C. Catanzaro, D. A. Patterson, K. A. Yelick, R. Bodik, and J. Shalf, “The Landscape of Parallel Computing Research : A View from Berkeley,” *Tech. Rep. No. UCB/EECS-2006-183*, 2006.

[17] S. W. Keckler, K. Olukotun, and H. P. Hofstee, *Multicore Processors and Systems*, 1st ed. Stanford, CA, USA: Springer, 2009.

[18] K. Bergman, L. P. Carloni, J. Chan, and G. Hendry, *Photonic Network- on-Chip Design*. .

[19] N. Magen, A. Kolodny, U. Weiser, and N. Shamir, “Interconnect-Power Dissipation in a Microprocessor.”

[20] M. Nikdast, G. Nicolescu, J. Trajkovic, and O. Liboiron-ladouceur, “Modeling Fabrication Non-Uniformity in Chip-Scale Silicon Photonic Interconnects,” *Autom. Test Eur. Conf. Exhib.*, pp. 115–120, 2016.

[21] J. S. Orcutt, A. Khilo, C. W. Holzwarth, M. A. Popović, H. Li, J. Sun, T. Bonifield, R. Hollingsworth, F. X. Kärtner, H. I. Smith, V. Stojanović, and R. J. Ram, “Nanophotonic integration in state-of-the-art CMOS foundries.,” *Opt. Express*, vol. 19, no. 3, pp. 2335–46, 2011.

[22] “ITRS 2.0: Top-Down System integration,” pp. 16–19, 2015.

[23] G. Kurian, J. E. Miller, J. Psota, J. Eastep, J. Liu, J. Michel, L. C. Kimerling, and A. Agarwal, “ATAC: a 1000-core cache-coherent processor with on-chip optical network,” *Proc. 19th Int. Conf. Parallel Archit. Compil. Tech. - PACT ’10*, no. September, p. 477, 2010.

[24] A. Biberman, K. Preston, G. Hendry, N. Sherwood-Droz, J. Chan, J. S. Levy, M. Lipson, and K. Bergman, “Photonic network-on-chip architectures using multilayer deposited silicon materials for high-performance chip multiprocessors,” *ACM J. Emerg. Technol. Comput. Syst.*, vol. 7, no. 2, pp. 1–25, 2011.

[25] J. Cardenas, C. B. Poitras, J. T. Robinson, K. Preston, L. Chen, and M. Lipson, “Low loss etchless silicon photonic waveguides,” *Opt. Soc. Am.*, vol. 17, no. 6, pp. 4752–4757, 2009.

[26] M. J. Shaw, J. Guo, G. A. Vawter, S. Habermehl, and C. T. Sullivan, “Fabrication techniques for low-loss silicon nitride waveguides Fabrication techniques for low loss silicon nitride waveguides,” *Proc. SPIE - Int. Soc. Opt. Eng. 5720*, no. October 2016, 2005.

[27] J. Liu, X. Sun, R. Camacho-Aguilera, L. C. Kimerling, and J. Michel, “Ge-on-Si laser operating at room temperature.,” *Opt. Lett.*, vol. 35, no. 5, pp. 679–681, 2010.

[28] I. O’Connor, F. Mieyeville, F. Gaffiot, A. Scandurra, and G. Nicolescu, “Reduction methods for adapting optical network on chip topologies to specific routing applications,”

Proc. Des. Circuits Integr. Syst., no. October 2015, pp. 12–14, 2008.

[29] M. Georgas, J. Leu, B. Moss, C. Sun, and V. Stojanovi, "Addressing Link-Level Design Tradeoffs for Integrated Photonic Interconnects," *Cust. Integr. Circuits Conf. (CICC), 2011 IEEE*, 2011.

[30] Y. Ye, S. Member, J. Xu, X. Wu, and S. Member, "System-Level Modeling and Analysis of Thermal Effects in Optical Networks-on-Chip," vol. 21, no. 2, pp. 292–305, 2013.

[31] Y. Xu, "Tolerating Process Variations in Nanophotonic On-chip Networks ∗ Background and Prior Work," vol. 00, no. c, pp. 142–152, 2012.

[32] C. Sun, C. O. Chen, G. Kurian, J. Miller, A. Agarwal, L. Peh, and V. Stojanovic, "DSENT: A Tool Connecting Emerging Photonics with Electronics for Opto-Electronic Networks-on-Chip Modeling," 2012.

[33] Intel, "Intel's Teraflops Research Chip Advancing multi-core technology into the tera-scale era.," *Teraflops Reasearch Chip*. [Online]. Available: http://download.intel.com/pressroom/kits/Teraflops/Teraflops_Research_Chip_Overview.pdf.

[34] I. H. Witten, E. Frank, and M. a. Hall, *Data Mining: Practical Machine Learning Tools and Techniques, Third Edition*, vol. 54, no. 2. 2011.

[35] J. R. Quinlan, *Induction of Decision Trees*. Morgan Kaufmann, 1986.

[36] J. R. Quinlan, "Learning With Continuous Classes," *World Sci.*, vol. 92, pp. 343–348, 1992.

[37] J. E. and A. A. Jason E. Miller, Harshad Kasture, George Kurian, Nathan Beckmann, Christopher Celio, "Graphite Multicore Simulator," 2010. [Online]. Available: https://github.com/mit-carbon/Graphite.

[38] C. Luk, B. C. Ed, F. C. G. Hi, E. D. Q. Rs, A. Tu, R. Cohn, R. Muth, H. Patil, and A. Klauser, "Pin : Building Customized Program Analysis Tools with Dynamic Instrumentation," *PLDI '05 Proc. 2005 ACM SIGPLAN Conf. Program. Lang. Des. implementation,* pp. 190–200, 2005.

[39] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta, “The SPLASH-2 programs,” *ACM SIGARCH Comput. Archit. News*, vol. 23, no. 2, pp. 24–36, 1995.

[40] G. E. Blelloch, C. E. Leiserson, B. M. Maggs, C. G. Plaxton, S. J. Smith, and M. Zogha, “A comparison of sorting algorithms for the connection machine CM-2itle,” *SPAA ’91 Proc. third Annu. ACM Symp. Parallel algorithms Archit.*, pp. 3–16, 1991.

[41] J. L. Hennessy, J. P. Singh, and A. Gupta, “Implications of Hierarchical N-Body Methods for Multiprocessor Architectures,” *ACM Trans. Comput. Syst.*, vol. 13, no. 2, pp. 141–202, 1995.

[42] P. A. L. Singh and J. L. Hennessy, “Finding and Exploiting Parallelism in an Ocean Simulation Experience , Results , and Implications Program :,” *J. parallel Distrib. Comput. 15*, pp. 27–48, 1992.

[43] L. Breiman, J. Friedman, C. J. Stone, and R. . Olshen, *Classification and Regression Trees*. Chapman & Hall/CRC, 1984.