UNIVERSITY OF CALIFORNIA, SAN DIEGO

Accurate Estimators and Optimizers for Networks-on-Chip

A dissertation submitted in partial satisfaction of the
requirements for the degree
Doctor of Philosophy

in

Electrical Engineering (Computer Engineering)

by

Kambiz Samadi

Committee in charge:

Professor Andrew B. Kahng, Chair
Professor Chung-Kuan Cheng
Professor Tara Javidi
Professor Bill Lin
Professor Tajana Simunic Rosing

2010

The dissertation of Kambiz Samadi is approved, and it is acceptable in quality and form for publication on microfilm and electronically:

_____

_____

_____

_____

_____

Chair

University of California, San Diego

2010

DEDICATION

- *To my lovely wife, Haleh, for her love, support and understanding.*
- *To the best sister in the world, Katayoon, for always being there for me.*
- *To my loving parents, Soraya and Hamid, without whose love, encouragement and sacrifices this thesis would not have been possible.*

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

I would like to thank my parents (my mother Soraya Emami and my father Hamid Samadi Bakhtiari) for their unconditional love and sacrifices. For their guidance and wisdom, without which I would not have been what I am. For their encouragement and support, without which I could not have gotten where I am today. I would like to especially thank my younger sister Katayoon Samadi for always being there for me, and for looking up to me for advice and inspiration. Also, I would like to thank my lovely wife Haleh Azartash for her love, understanding and support. For her sacrifices, cooperation, and levelheaded attitude for the past three years. For being my inspiration and the reason to succeed.

I certainly feel privileged and grateful to work under Prof. Andrew B. Kahng's guidance. I especially thank him for a great deal of life lessons that he taught me along the research path. Without doubt, Prof. Kahng has introduced me to new patterns of creative, effective thinking, and it is always inspiring to observe Prof. Kahng's energy. I also thank him for the freedom he gave me in pursing different research ideas. Indeed, Professor Kahng's laboratory is a thriving research environment, and I have no doubt that it would have been impossible to pursue the scope and diversity of work I did with him anywhere else.

I am thankful to my thesis committee members, Prof. Chung-Kuan Cheng, Prof. Tara Javidi, Prof. Bill Lin, and Prof. Tajana Simunic Rosing for taking time out of their schedules to review my research and provide useful feedback. I especially would like to thank Prof. Lin for all the research collaborations, and for all his energetic and inspiring discussions. I am also privileged to have collaborated closely with Prof. Cheng.

I would like to sincerely thank Dr. Shahin Mehdizad Taleie and Dr. Niloufar Reisian for being more than just friends. For their enormous help and support for the last few months that have enabled me to keep my focus on my studies.

I would also like to thank my mentors at Intel Corporation, Dr. Marco A. Escalante, Dr. Michael Kishinevsky, Dr. Umit Y. Ogras, and Dr. Emily Shriver for providing me my most rewarding work experience. In addition, I would also like to thank my mentor at Qualcomm Inc., Dr. Riko Radojcic, for providing a most enjoyable work environment.

I feel privileged to have a great group of friends and collaborators. I would like to thank Kwangok Jeong, Seokhyeong Kang, Jingwei Lu, Tuck-Boon Chan, Rohit Sunkam Ramanujam, Dr. Chul-Hong Park, Dr. Swamy Muddu, Dr. Sherief Reda, Dr. Puneet Sharma, Dr. Alessandro Pinto, Prof. Luca Carloni, Prof. Puneet Gupta, Prof. Li-Shiuan Peh, Dr. Bin Li and Prof. Hailong Yao for their collaboration and excellent research ideas. I would like to especially thank Kwangok Jeong, Seokhyeong Kang, Jingwei Lu, and Tuck-Boon Chan for taking their time to proofread my thesis many times. This thesis could have not come together in its current form without their significant help.

I am also privileged to have had the opportunity of meeting a lot of good friends in San Diego, and would like thank them all for their sincere friendship. I would to especially thank Shervin Sharifi, Kiarash Kiantaj, Ehsan Ardestanizadeh, and Amirali Shayan for all the good times that we had together. I am looking forward to strengthening our friendships further into the future.

I would also like to thank VLSI CAD Laboratory administrator Virginia McIlwain, ECE graduate program coordinator Shana Slebioda, and payroll managers Yuka Nakanishi and MLissa Michelson for their support and cooperation which many times went above and beyond their job responsibilities.

Last, but not least, I would like to express my deepest gratitude to my uncles, Dr. Ahmad Emami and Mr. Houman Emami, and their families, for their constant love and support since I immigrated to United States. My transition could have not been smooth had it not been for their unconditional support.

The material in this thesis is based on the following publications.

- Chapter 3 is based on the following publications:
  - Chung-Kuan Cheng, Andrew B. Kahng, **Kambiz Samadi** and Amirali Shayan, "Worst-case Performance Prediction Under Supply Voltage and Temperature Variation", *Proc. ACM/IEEE International Workshop on System-Level Interconnect Prediction*, 2010, pp. 91-96.
  - Luca Carloni, Andrew B. Kahng, Swamy Muddu, Alessandro Pinto, **Kambiz Samadi** and Puneet Sharma, "Accurate Predictive Interconnect Modeling for System-Level Design", *IEEE Transactions on Very Large Scale Integration Systems* 18(4) (2010), pp. 679-684.

- Kwangok Jeong, Andrew B. Kahng and **Kambiz Samadi**, "Architectural-Level Prediction of Interconnect Wirelength and Fanout", *Proc. IEEE International SOC Conference*, 2009, pp. 53-56.
- Luca Carloni, Andrew B. Kahng, Swamy Muddu, Alessandro Pinto, **Kambiz Samadi** and Puneet Sharma, "Interconnect Modeling for Improved System-Level Design," *Proc. IEEE Asia and South Pacific Design Automation Conference*, 2008, pp. 258-264.

- Chapter 4 is based on the following publications:
  - Andrew B. Kahng, Bin Li, Li-Shiuan Peh and **Kambiz Samadi**, "ORION 2.0: A Power-Area Simulator for Interconnection Networks", to appear in *IEEE Transactions on Very Large Scale Integration Systems*.
  - Kwangok Jeong, Andrew B. Kahng, Bill Lin and **Kambiz Samadi**, "Accurate Machine Learning-Based On-Chip Router Modeling", *IEEE Embedded Systems Letters* 2(3) (2010), pp. 62-66.
  - Andrew B. Kahng, Bin Li, Li-Shiuan Peh and **Kambiz Samadi**, "ORION 2.0: A Fast and Accurate NoC Power and Area Model for Early-Stage Design Space Exploration," *Proc. Design, Automation and Test in Europe*, 2009, pp. 423-428.

- Chapter 5 is based on the following publications:
  - Andrew B. Kahng, Bill Lin, **Kambiz Samadi** and Rohit Sunkam Ramanujam, "Efficient Trace-Driven Metaheuristics for Optimization of Networks-on-Chip Configurations", *Proc. IEEE/ACM International Conference on Computer-Aided Design*, 2010, pp. 256-263.
  - Andrew B. Kahng, Bill Lin, **Kambiz Samadi** and Rohit Sunkam Ramanujam, "Trace-Driven Optimization of Networks-on-Chip Configurations", *Proc. ACM/IEEE Design Automation Conference*, 2010, pp. 432-437.

- Chapter 6 is based on the following draft:

– Marco A. Escalante, Andrew B. Kahng, Michael Kishinevsky, Umit Y. Ogras, **Kambiz Samadi** and Emily Shriver, "Multi-Product Floorplan Optimization Framework for Chip Multiprocessors", *draft in submission*, November 2010.

My coauthors (Prof. Luca Carloni, Prof. Chung-Kuan Cheng, Dr. Marco A. Escalante, Kwangok Jeong, Prof. Andrew B. Kahng, Dr. Michael Kishinevsky, Dr. Bin Li, Prof. Bill Lin, Dr. Umit Y. Ogras, Prof. Li-Shiuan Peh, Dr. Swamy Muddu, Dr. Alessandro Pinto, Dr. Puneet Sharma, Amirali Shayan, Dr. Emily Shriver and Rohit Sunkam Ramanujam) have all kindly approved the inclusion of the aforementioned publications in my thesis.

VITA

1980	Born, Tehran, Iran
2004	B.Sc., Computer Engineering, California State University, Fresno
2007	M.Sc., Electrical Engineering (Computer Engineering), University of California, San Diego
2008	C.Phil., Electrical Engineering (Computer Engineering), University of California, San Diego
2010	Ph.D., Electrical Engineering (Computer Engineering), University of California, San Diego

All papers coauthored with my advisor Prof. Andrew B. Kahng have authors listed in alphabetical order.

- Andrew B. Kahng, Bin Li, Li-Shiuan Peh and **Kambiz Samadi**, "ORION 2.0: A Power-Area Simulator for Interconnection Networks", to appear in *IEEE Transactions on Very Large Scale Integration Systems*.

- Andrew B. Kahng, Bill Lin, **Kambiz Samadi** and Rohit Sunkam Ramanujam, "Efficient Trace-Driven Metaheuristics for Optimization of Networks-on-Chip Configurations", *Proc. IEEE/ACM International Conference on Computer-Aided Design*, 2010, pp. 256-263.

- Kwangok Jeong, Andrew B. Kahng, Bill Lin and **Kambiz Samadi**, "Accurate Machine Learning-Based On-Chip Router Modeling", *IEEE Embedded Systems Letters* 2(3) (2010), pp. 62-66.

- Andrew B. Kahng, Bill Lin, **Kambiz Samadi** and Rohit Sunkam Ramanujam, "Trace-Driven Optimization of Networks-on-Chip Configurations", *Proc. ACM/IEEE Design Automation Conference*, 2010, pp. 432-437.

- Chung-Kuan Cheng, Andrew B. Kahng, **Kambiz Samadi** and Amirali Shayan, "Worst-case Performance Prediction Under Supply Voltage and Temperature Variation", *Proc. ACM/IEEE International Workshop on System-Level Interconnect Prediction*, 2010, pp. 91-96.

- Luca Carloni, Andrew B. Kahng, Swamy Muddu, Alessandro Pinto, **Kambiz Samadi** and Puneet Sharma, "Accurate Predictive Interconnect Modeling for System-Level Design", *IEEE Transactions on Very Large Scale Integration Systems* 18(4) (2010), pp. 679-684.

- Andrew B. Kahng, Bill Lin and **Kambiz Samadi**, "Improved On-Chip Router Analytical Power and Area Modeling", *Proc. Asia and South Pacific Design Automation Conference*, 2010, pp. 241-246.

- Kwangok Jeong, Andrew B. Kahng and **Kambiz Samadi**, "Architectural-Level Prediction of Interconnect Wirelength and Fanout", *Proc. IEEE International SOC Conference*, 2009, pp. 53-56.

- Andrew B. Kahng, Bin Li, Li-Shiuan Peh and **Kambiz Samadi**, "ORION 2.0: A Fast and Accurate NoC Power and Area Model for Early-Stage Design Space Exploration", *Proc. Design, Automation and Test in Europe*, 2009, pp. 423-28.

- Andrew B. Kahng and **Kambiz Samadi**, "Communication Modeling for System-Level Design", *Proc. IEEE International SOC Conference*, 2008, pp. 138–143.

- Luca Carloni, Andrew B. Kahng, Swamy Muddu, Alessandro Pinto, **Kambiz Samadi** and Puneet Sharma, "Interconnect Modeling for Improved System-Level Design", *Proc. Asia and South Pacific Design Automation Conference*, 2008, pp. 258-264.

ABSTRACT OF THE DISSERTATION

Accurate Estimators and Optimizers for Networks-on-Chip

by

Kambiz Samadi

Doctor of Philosophy in Electrical Engineering (Computer Engineering)

University of California, San Diego, 2010

Professor Andrew B. Kahng, Chair

Networks-on-chip (NoCs) are emerging as the de facto on-chip interconnection fabric of choice for both general-purpose chip multiprocessors (CMPs) [68, 108, 110] and application-specific multiprocessor systems-on-chip (MPSoCs) [43, 78]. When the number of on-chip cores increases, the need for scalable and high-bandwidth communication fabric becomes more evident [43, 78]. Another megatrend in advanced technologies is that power has become the most critical design constraint [57, 6].

In this thesis, we present integrated research on NoC power, performance and area modeling to enable efficient early-stage design space exploration that improves our understanding and characterization of the NoC power-area-latency design space. The intellectual merit of our proposed approaches stems from their balanced attack on necessary NoC-specific techniques for (1) architecture-level estimation (to provide

correct optimization objectives) and (2) architecture-level optimization (to expand the achievable design envelope). In the *architecture-level estimation* thrust, we develop new architecture-level estimation methods that are accurate and easily *portable* to different router microarchitectures. Also, our proposed models can accurately capture implementation effects. Specifically, we develop

- *automatic* generation of accurate architecture-level estimation models;
- *portable* models across different microarchitectures; and
- *accurate* modeling of application-specific integrated circuit (ASIC) implementation flow choices and their impacts.

In the *architecture-level optimization* thrust, we develop

- *trace-driven* optimizations of NoC configurations for actual traffic behavior and workloads; and
- *simultaneous* floorplan optimization of chip multiprocessors across multiple products.

The broader impact of this thesis lies in helping NoC intellectual property (IP) and MPSoC designers reduce design turnaround time in addition to product chip area, delay and power metrics. This will enable the design of more complex and functional products within a given cost and power envelope. With our models, we also develop an infrastructure to extract necessary model inputs from several reliable sources (e.g., Liberty [8], SPICE [15], etc.) to ease the updating of models as new technology files become available. Finally, a significant contribution of this thesis lies in providing a publicly available framework for accurate and efficient NoC modeling [12].

Chapter 1

Introduction

Networks-on-chip (NoCs) are emerging as the de facto interconnection fabric of choice for both general-purpose chip multiprocessors (CMPs) [68, 108, 110] and application-specific multiprocessor systems-on-chip (MPSoCs) [43, 78]. As the number of on-chip cores increases, a scalable and high-bandwidth communication fabric to connect them becomes important [28, 44, 79]. At the same time, power has become a critical design constraint in advanced technologies [57, 6]. With power being a first-order design constraint, NoCs must be designed carefully to minimize the overall power while meeting performance objectives. However, with increasing demand for network bandwidth, the power that an interconnection network consumes will also be substantial [57]. The International Technology Roadmap for Semiconductors (ITRS) [6] predicts that future generations of high-end VLSI designs will operate in 10-20 GHz range with the communication between cores in Gbit/s. This requires designers to work within a tight power budget.

With many opposing parameters, effective early-stage design space exploration is essential to enable the realization of achievable power-performance-area tradeoffs. Effective early-stage design space exploration depends on the availability of accurate architecture-level estimation models and high-level optimization techniques. The key characteristic of such optimization techniques is being able to accurately capture low-level implementation effects as well as the actual application performance. This thesis focuses on two important and complementary determinants of achievable power-performance-area tradeoffs in NoC design: (1) accurate estimation of architecture-level

power, performance and area and (2) new optimization methods for NoC configurations.

To overcome the limitations of existing NoC estimation and optimization methods, we first develop new architecture-level estimation methods that are accurate, can support projections into the future technology nodes, and are easily portable to different router microarchitectures. Our proposed models can also accurately capture impacts of implementation flows and options. We further develop new architecture-level optimization techniques that can target specific actual application traffic behaviors. This thesis seeks a "virtuous cycle" of synergies from its balanced attack on both architecture-level estimation and optimization thrusts, as illustrated in Figure 1.1. On the one hand, improved techniques for modeling and estimating achievable NoC design parameters (area, performance, power, etc.) provide more accurate and more easily evaluated objectives for optimization. However, improved optimizations will change the achievable envelope of NoC architecture design, leading to new estimation challenges.

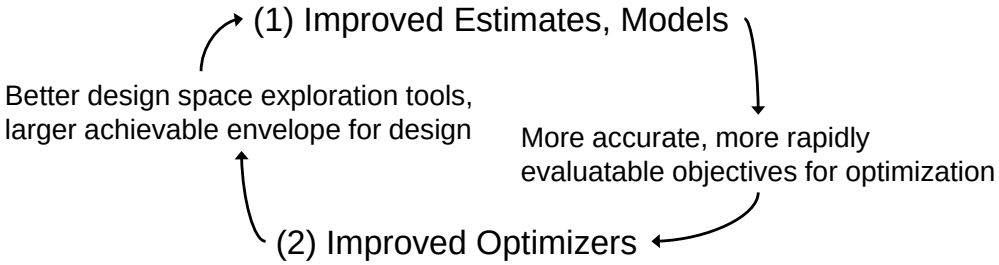

Figure 1.1: Virtuous cycle of estimation and optimization for future NoC architectures.

Current network-on-chip research directions can be broadly divided into two main categories: (1) estimation and (2) optimization, as shown in Figure 1.2.

In the context of estimation, the ongoing NoC research is at three abstraction levels: (1) circuit level, (2) architecture level and (3) system level. At the bottom level (i.e, circuit level), on-chip network modeling focuses on developing new physical estimation models for emerging circuit techniques. For example, new aggressive design of crossbar switches for high-performance and low-power applications use advanced circuit techniques such as bit-interleaved or double-pumped custom crossbars [57, 62]. Next, at the architecture level, relevant microarchitectural parameters are also included in the power, performance and area estimation models. This is an active area of research. As new microarchitectures and circuit implementations are developed, there is a corresponding

need for appropriate cost models. Existing architecture-level estimation methods can be classified as *template-based* approaches where the models are based on a set of circuit templates and assumed architectures [41, 63, 114]. Template-based models include parametric models in which parametric regression is used to derive analytical models based on the modeler's understanding of the underlying circuit implementation and architecture [36, 71, 77, 85]. We propose a new modeling framework based on *machine learning-based nonparametric* regression techniques where the goal is to decouple the understanding of the underlying circuit implementation and architecture from the modeling effort. We use both template-based and nonparametric regression techniques to develop accurate power, performance and area models for on-chip routers and interconnects. Our contributions in the context of NoC estimation are highlighted as the gray boxes in Figure 1.2. Finally, the highest level of estimation is at the system level, where many circuit-implementation and technology-dependent parameters are ignored. These techniques are mostly to simulate the functionality or to estimate high-level power consumption of the on-chip networks [49, 102, 98].

In the context of optimization, the ongoing NoC research directions span across all building blocks of on-chip networks: topology, routing, flow control, router microarchitecture, and link architecture.[1] Topology of a network affects its ability to efficiently disseminate information. Network topology not only plays an important role in network latency, throughput and power consumption, but also determines the routing strategy to map the cores to the network nodes [59, 80]. Routing choices are of great importance in determining the network performance and power consumption. Complicated routing strategies result in larger designs [50, 58]. Flow control mechanisms, on the other hand, affect the quality of service in on-chip networks and the complexity of the verification process [43, 55, 80]. Finally, router and interconnect architectures directly affect the overall network performance, power consumption, and area; they are the two physical components of any given NoC that must be carefully modeled and optimized. In this thesis, we focus on router microarchitecture optimization. Among the various components of an input-buffered router, the configuration of input buffers has been shown to have a major impact on both the overall performance of an on-chip network as well

[1] In this thesis, we use interconnect and link interchangably.

as its energy consumption [57, 67, 74, 63]. Therefore, determining the optimal buffer size and virtual channel (VC) allocation is of critical importance in maximizing performance and minimizing power consumption. We specifically focus on the problem of VC allocation. To do this, we propose a new *trace-driven* optimization paradigm in which we use the actual application during the optimization process rather than *average rate-driven* models (cf. Section 1.2).

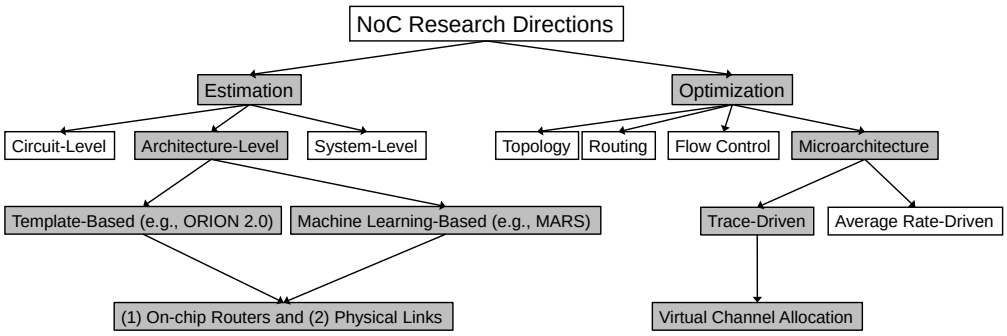

Figure 1.2: Taxonomy of NoC research areas. Gray items show the focus of this thesis.

1.1 The Estimation Problem

Architecture-level power estimation is extremely important to (1) verify that power budgets are approximately met by the different parts of the design, and (2) evaluate the effects of various high-level optimizations, which have been shown to have much more significant impact on power than low-level optimizations. Existing architecture-level estimation methods in the literature can be broadly classified as template-based approaches: in one way or another, these existing methods assume a specific architecture and underlying circuit implementation in their modeling efforts.

Other template-based approaches are based on parametric models [25, 27, 30, 36, 71, 85, 77]. These approaches assume a specific underlying on-chip router microarchitecture, requiring the development of new models for different microarchitectures.[2] Each architectural component is captured as a parametric function of configuration parameters. While some of these approaches aim to capture the detailed effects of different

[2]In this thesis, "on-chip router" and "router" are used interchangeably.

application-specific integrated circuit (ASIC) implementation flows by using parametric regression analysis on either pre-layout [30, 36, 71, 85] or post-layout [25, 27, 77] simulation results, the modeler still needs full comprehension of the underlying router microarchitecture in order to come up with a relevant parametric model. Moreover, it is difficult to capture configuration parameter interactions that may gain significance as the design complexity increases. Finally, most existing parametric modeling approaches fail to consider implementation and technology-dependent parameters in their models. The template-based approach has two potential limitations.

- First, for the power and area estimations to be accurate, the actual router microarchitecture used for implementation must closely match the microarchitecture assumed. Differences such as transistor netlist-based versus multiplexor tree-based crossbars, speculative versus non-speculative VC allocation, oblivious versus adaptive routing, etc. can lead to significantly different power and area costs. Although the circuit template-based methodology allows for development of new power and area cost models for different router microarchitectures, substantial development effort would be required to support new microarchitecture features (e.g., expressed virtual channels [70]) or inherently different classes of microarchitectures, e.g., the distributed shared buffer microarchitecture proposed in [103].

- Second, capturing the effects of different ASIC implementation flows and flow options is difficult. Modern integrated circuit (IC) implementation flows incorporate powerful logic synthesis and physical synthesis transformations (e.g., logic restructuring, gate sizing, buffer insertion, etc.) to satisfy stringent requirements that span not only performance, but also power, reliability and manufacturability. The detailed impact of such transformations are difficult to capture in static circuit templates, as they depend on implementation parameters such as process and library flavor, operating voltage, or target clock frequency. Similarly, effects of place-and-route choices (utilization, block aspect ratio, etc.) are also difficult to capture. Even more daunting, all of these implementation effects have sensitivities to the choice of specific CAD tool chains.

In addition to the above limitations, power, performance and area modeling for

architecture-level optimization suffers from:

- poor definition of inputs required to make design choices,
- *ad hoc* selection of model inputs, and
- lack of model extensibility across multiple/future technologies [35].

As we explain in Chapter 2, router microarchitecture and physical links are the two most significant contributors to NoC power, performance and area [44, 48]. Hence, this thesis focuses on architecture-level estimation and optimization of router microarchitecture and physical links.

Even though template-based approaches, including parametric regression, have the potential drawback of being tied to a given architecture and circuit implementation, they are still of great interest. This is because they enable efficient early-stage design space exploration with little or no implementation cost.[3] Hence, in the context of physical links, our objective is to provide architecture-level designers with fast and accurate models that can be used in the early phase of a system-on-chip (SoC) design process. To date, there are no accurate yet simple models available to architecture-level designers. Current models are either quite accurate but too complex to be employed at the architecture level, or else too coarse and inaccurate which leads to incorrect architectural design decisions. We show that accurate models can still be simple and that different optimization results and trends can be achieved from the use of improved models. Different from previous work in the literature, we build our models through accurate experimentation and calibration against industry technology files.

In addition, the most critical gap in existing architecture-level cost models and NoC optimizations is the lack of well-defined pathways to capture necessary technology and device parameters from the wide range of available sources. Since exploration of the architecture-level power-performance envelope is typically done for current and future technologies, the models driving architecture-level design must be derivable from standard technology files. Figure 1.3 shows an example infrastructure that we develop

[3] We note that for the parametric regression models to be relatively accurate, implementation data is required for calibration purposes.

to maintain our models with multiple sources of technology and circuit inputs (e.g., Liberty format [8], LEF [7]), as well as extrapolatable models of process (e.g., ITRS [6] and PTM [13]). Chapter 3 describes the details of our interconnect modeling approaches.

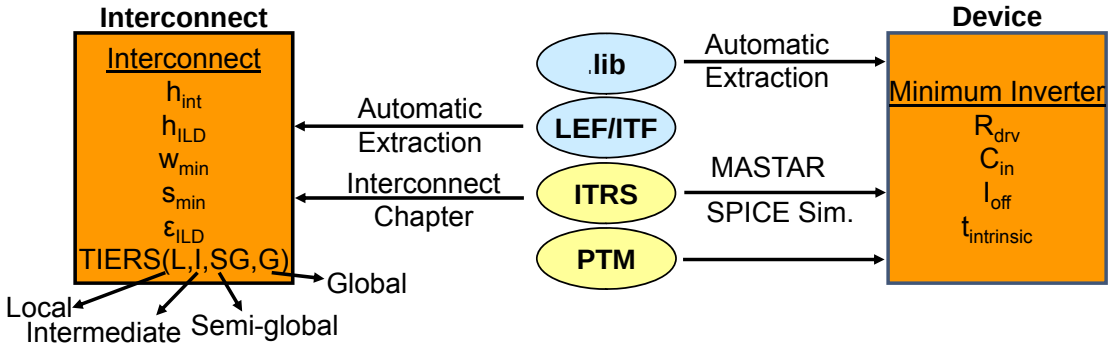

Figure 1.3: An example infrastructure to maintain the proposed interconnect model with inputs from reliable sources.

In the context of on-chip routers at the architecture level, Patel *et al.* [88] propose a power model for interconnection networks, deriving its power estimations based on transistor count. As the model does not instantiate the architectural parameters, it cannot be used to explore tradeoffs in router microarchitecture design. In addition, the existing template-based estimation approaches are exemplified by the widely-used early-stage NoC power estimation tool ORION 1.0 [114] which is based on parameterized power and area models derived from a mix of circuit templates. Other template-based approaches that are based on parametric regression include [29, 30]. However, when we validate the existing models (e.g., ORION 1.0) against existing NoC prototypes (i.e., the Intel 80-core Teraflops chip [57]) we notice up to 8× difference between ORION 1.0 estimations (per component) and silicon measurements. Also, the estimated total power is about 10× less than actual. Indeed, ORION 1.0 does not include clock and link power models, which are major components of NoC power.

Furthermore, existing on-chip router models (e.g., ORION 1.0, etc.) collect inputs from ad hoc sources to drive their internal power, performance, and area estimations. Similar to the interconnect modeling, we develop an infrastructure (i.e., using shell scripting) to extract technology and circuit inputs from reliable sources and to ease the updating of the models as new technology files become available. Given the short-

comings in the existing models, we develop ORION 2.0, a set of accurate architecture-level on-chip router power and area models. Chapter 4 gives further details of derivation and validation of ORION 2.0 models.

Even though our proposed template-based (including the parametric regression-based) models have significantly enhanced the state-of-the-art in architecture-level estimation of on-chip routers and interconnects, an easily *reproducible* modeling methodology which decouples the understanding of the underlying architecture and circuit implementation from the modeling effort is quite desirable. To enable such a methodology, we propose the use of machine learning-based regression techniques. In this methodology, we use existing layout data to perform data-driven predictive modeling via non-parametric regression analysis. We show the proposed machine learning-based models are quite accurate (i.e., up to within 3.5% of the layout data). Chapters 3 and 4 show the application of machine learning-based nonparametric regression in interconnect and on-chip router modeling, respectively. In addition, we show the extensibility of non-parametric methods to the modeling of metrics that go beyond power, performance and area; specifically, we exhibit new models of interconnect wirelength and fanout.

1.2 The Optimization Problem

The design of an on-chip network can be broken into its various building blocks: topology, routing, flow control, router microarchitecture and link architecture. Among these building blocks, router microarchitecture is of utmost importance due to its great impact on communication latency. As a result, significant research effort has been spent reducing router latency through modified router microarchitecture and design [48]. NoCs can be designed for general-purpose CMPs [68, 108, 110] or application-specific MPSoCs [43, 78]. The challenges are different in each case. Since general-purpose CMPs are designed to run a wide range of applications, the application traffic characteristics are inherently unknown *a priori*. Hence, the configurations of on-chip routers, such as the number of virtual channels, are typically *uniform* across all routers in the design. On the other hand, since application-specific MPSoCs are designed to implement specific functions efficiently, the configuration of each router in the network can

be *non-uniformly* optimized to the traffic characteristics of the particular application.

Though the problem of NoC configuration for application-specific MPSoCs is not new, prior approaches [37, 60, 61] have been *average-rate driven* in that the traffic characteristics have been modeled with *average data rates*. Unfortunately, average-rate models are poor representations of actual traffic characteristics of real applications. Figure 1.4 contrasts actual versus average traffic of two real applications from the PARSEC [31] benchmark suite. The actual traffic behavior tends to be very bursty, with substantial fluctuations over time. This motivates a hypothesis that average-rate driven approaches may be misled by average traffic characteristics, resulting in poor design choices that are not well-matched to actual traffic characteristics.

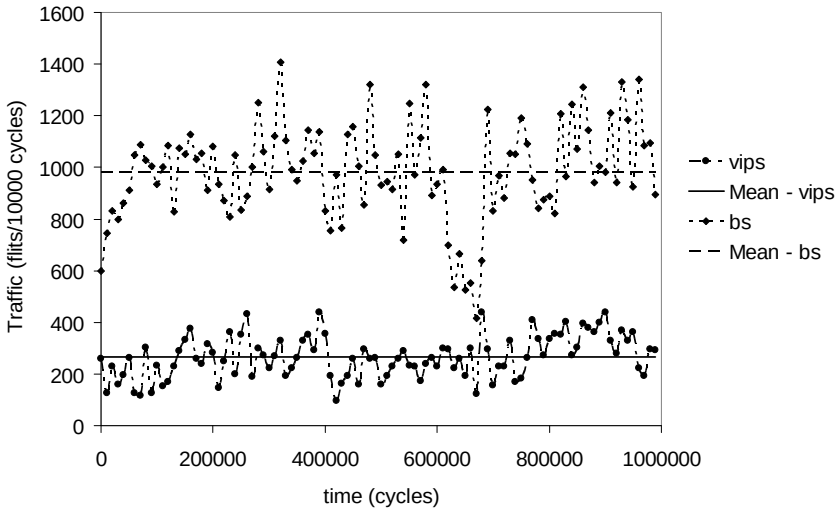

Figure 1.4: Actual versus average communication traffic between two arbitrary nodes in the network, for two different PARSEC benchmark applications.

There are two main reasons why an average-rate driven optimization approach leads to poor results. First, since the actual application traffic tends to fluctuate significantly, average-rate traffic models can greatly overestimate traffic loads during some intervals and underestimate during other intervals. Second, the need for network resources such as virtual channels is not necessarily dictated by average traffic loads. Specifically, virtual channels are useful in avoiding *head-of-line* (HOL) blocking situations that would cause a network channel to remain idle despite the presence of packets waiting to access it. However, this blocking phenomenon is closely related to the actual

traffic sequence rather than to the average traffic load. This thesis aims to improve the existing understanding of NoC power-performance-area envelope by introducing new modeling and optimization techniques.

In Chapter 5 we quantify the limitations of average-rate driven approaches for the specific problem of virtual channel allocation. We evaluate an existing average-rate driven VC allocation method [61] using applications in the PARSEC benchmark suite [31], which contains multi-threaded programs that are representative of emerging workloads. The evaluation is based on minimizing the total number of virtual channels allocated to achieve a given average packet latency performance. Our proposed trace-driven approaches match the average packet latencies achieved by average-rate driven optimization methods using up to 35% fewer VCs, with a corresponding reduction in buffer requirements (power and area).

In Chapter 6, we investigate the problem of multi-product floorplan optimization of chip multiprocessors. CMPs are one of the main consumers of NoCs. In a given CMP, there are different resources (i.e., tiles) that are placed on a gird. Tile-level floorplanning is done in such a way that tiles can easily communicate with each other through a mesh (or ring) network. State-of-the-art floorplan optimization techniques can be used to perform tile-level floorplanning [19, 97, 101, 104, 116]. In contrast to traditional chip floorplanning, we address the simultaneous optimization of the floorplans of multiple CMP products, and ensure that smaller floorplans can be obtained from larger ones through simple "chopping" operations.

1.3 This Thesis

In the *architecture-level estimation* thrust, this thesis achieves the following.

- We develop accurate and predictive interconnect models, along with a *reproducible* methodology to derive them.
- We build our predictive power, performance and area models using parametric regression analysis and industry technology files. Our framework is capable of modeling and optimizing buffered interconnects for various technology nodes. The framework is accessible through XML files or through a C++ API.

- We develop a machine learning-based interconnect performance modeling framework to drive efficient circuit optimizations. Our models account for supply voltage and temperature variations which are increasingly significant in advanced technologies.

- For on-chip routers, we develop ORION 2.0 [63], a power-area simulator for interconnection networks, which provides the most widely used on-chip router models in both academia and industry.

- We develop a *framework* to model on-chip router power, performance and area using machine learning-based nonparametric regression methods. Our approach aids the automatic generation of accurate architecture-level on-chip router models which also capture different IC implementation flow effects. We have released our models in the form of C++ functions and scripts to enable further NoC research and design.

In the *architecture-level optimization* thrust, this thesis achieves the following.

- We develop architecture-level NoC optimization techniques based on a *trace-driven* paradigm that directly incorporates actual application traffic behavior and workloads into the optimization process.

- We develop a new metric called "significant VC failure" which efficiently captures runtime VC contentions in the network and enables scaling of the proposed trace-driven approaches to larger networks.

- We develop a multi-product floorplan optimization framework for CMPs. Our approach is based on the fact that communication-aware floorplanning is done at the tile-level, *a priori*; hence, our focus is at the chip level where the goal is to efficiently determine the number of necessary resources as well as their placement on the chip across multiple products. Our approach can significantly reduce redesign overhead associated with each product and shorten the design turnaround time.

Finally, a significant contribution of this thesis is to provide publicly-available NoC power, performance and area models to enable further NoC research and design [12, 17].

The remainder of this thesis is organized as follows. Chapter 2 provides descriptions of on-chip router microarchitectural building blocks. Chapter 3 describes our proposed interconnect power, performance and area models and shows the extensibility of our machine learning-based modeling approach to metrics beyond power, performance and area. Chapter 4 gives details of ORION 2.0 power and area models. Chapter 5 proposes a new NoC configuration optimization paradigm, using the actual application trace. Chapter 6 provides a new approach to simultaneous floorplanning of multiple CMP products. Finally, Chapter 7 summarizes the main results of this thesis and points to a number of future directions.

Chapter 2

Network-on-Chip Architectural Building Blocks

2.1 Introduction

Since the introduction of research on multi-core chips more than a decade ago, on-chip networks have emerged as an important and growing field of research. As core counts increase, there is a corresponding increase in bandwidth demand to facilitate high core utilization. On the other hand, on-chip networks are prevalent in computing domains ranging from high-end servers to embedded system-on-chip (SoC) devices. This diversity of application platforms has led to research in on-chip networks spanning a variety of disciplines from computer architecture to computer aided design, embedded systems, VLSI, etc. [48]. This chapter provides a description of NoC architectural building blocks to familiarize the reader with the type of problems that this thesis aims to address. In addition, in Section 2.4, we describe the communication synthesis problem through an example of an existing system-level NoC optimization tool, COSI-OCC [92]. We show that our accurate models affect the solutions obtained by the system-level tool.

2.2 Network-on-Chip Building Blocks

The design of an NoC can be broken down into its various building blocks: topology, routing, flow control, router microarchitecture and design and link architecture. In the following, we briefly explain each building block [48].

- *Topology.* An on-chip network is composed of channels and router nodes. The network topology determines the physical layout and connections between nodes and channels in the network.

- *Routing.* For a given topology, the routing algorithm determines the path through the network that a message takes to reach its destination. A routing algorithm's ability to balance traffic (or load) has a direct impact on the throughput and performance of the network.

- *Flow control.* Flow control determines how resources are allocated to messages as they travel through the network. The flow control mechanism is responsible to allocate (and de-allocate) buffers and channel bandwidth to waiting packets. Resources can be allocated to packets in their entirety (i.e., in store-and-forward and virtual cut-through flow control); however, this requires very large buffer resources which are impractical for on-chip purposes. Most commonly, on-chip networks handle flow control at the flit level. Buffers and channel bandwidth are allocated on the smaller granularity of flits rather than whole packets; as a result, routers can be designed with smaller buffers.

- *Link architecture.* Links are a major component of the on-chip networks, but they are not treated with an in-depth analysis. In Chapter 3, we develop new interconnect power, performance and area models which efficiently capture the impacts of technology and circuit parameters as well as architectural parameters.

- *Router microarchitecture.* A generic router microarchitecture is comprised of the following components: input buffers, router state, routing logic, allocators and a crossbar switch. Router functionality is often pipelined to improve throughput. Delay through each router in the on-chip network is the primary contributor to

communication latency. Therefore, a significant portion of this thesis is dedicated to the modeling and optimization of on-chip router microarchitecture. Chapter 4 proposes different approaches to accurately model on-chip router power, performance and area. In Chapter 5, we propose new *trace-driven* virtual channel (VC) allocation optimization approaches which reduce router power by efficiently allocating VCs, using the application knowledge.

2.3 Router Microarchitecture

Routers must be designed to meet latency and throughput requirements given tight area and power constraints; this is a primary challenge that designers are facing as many-core systems scale. As router complexity increases with bandwidth demands, simple routers (unpipelined, wormhole, limited buffering) can be built when high throughput is not needed, so area and power overhead is low. Challenges arise when the latency and throughput demands of on-chip networks are increased. A router's architecture determines its critical path delay, which affects per-hop delay and overall network latency. Router microarchitecture also impacts network energy as it determines the circuit components in a router and their activity. The implementation of the routing, flow control and the actual router pipeline affects the efficiency with which buffers and links are used. Finally, the area footprint of the router is clearly determined by the chosen router microarchitecture and underlying circuits.

Figure 2.1 shows the microarchitecture of a VC router. The shown example has five input and output ports corresponding to the four neighboring directions (i.e., North, East, South and West) and the injection/ejection port that communicates with the local processing element. The major building blocks are the input buffers, route computation logic, virtual channel allocator, switch allocator and crossbar switch. Most on-chip network routers are input-buffered, that is, packets are stored in buffers only at the input ports [44, 48].

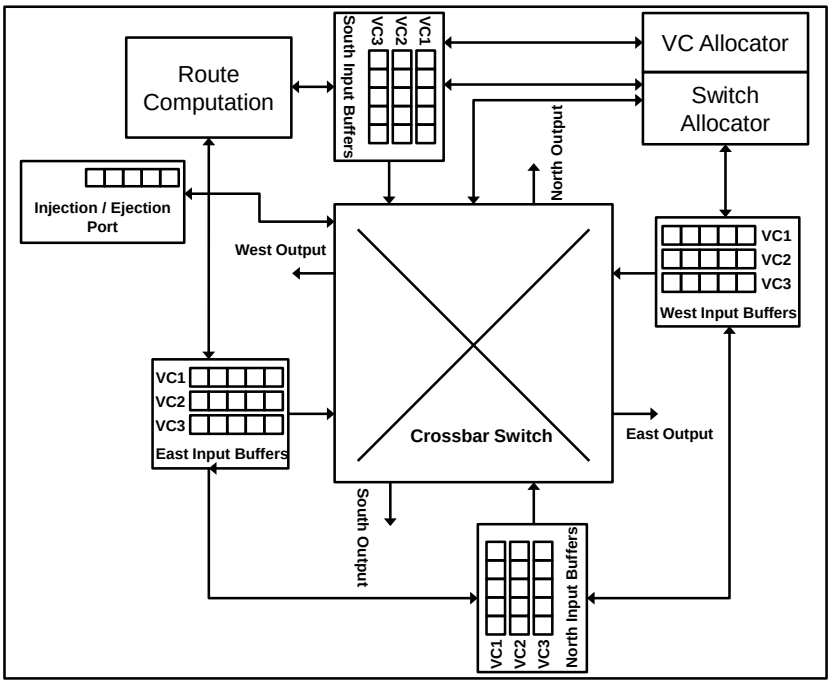

Figure 2.1: A virtual-channel router microarchitecture.

2.3.1 Router Pipeline

The router shown in Figure 2.1 has five logical stages. A head flit, upon arriving at an input port, is first decoded and buffered according to its input VC in the buffer write pipeline stage. Next, the routing logic performs route computation (RC) to determine the output port for the packet. The header then arbitrates for a VC corresponding to its output port in the VC allocation (VA) stage. Upon successful allocation of a VC, the header flit proceeds to the switch allocation stage and requests for the switch input and output ports. Upon winning the output port, the flit is read from the buffer and proceeds to the switch traversal stage. Finally, the flit is passed to the next node in the link traversal stage. Body and tail flits follow a similar pipeline except that they bypass RC and VA stages. The tail flit de-allocates the VC reserved by the head flit after leaving the router. Note that a wormhole router only requires four logical stages (i.e., no VA stage), and only has a single buffer queue in each input port [44].

A router that is running at a low clock frequency will be able to fit all five stages into a single clock cycle. For aggressive clock frequencies, the router architecture must

be pipelined. The actual physical pipeline depends on the implementation of each of these logical stages and their critical path delay. If the physical pipeline has five stages similar to the logical stages, then the stage with the longest critical path delay will set the clock frequency. For example, the VC allocation stage will set the clock frequency when the number of VCs is high, or the crossbar traversal stage will set the clock frequency if it has very wide and highly-ported crossbars. The clock frequency can also be determined by the overall system clock. Increasing the number of physical pipeline stages increases the per-hop router delay for each message, as well as the buffer turnaround time. The buffer turnaround time affects the minimum required buffering which in turn affects the throughput. Thus, pipeline optimizations have been proposed and employed to reduce the number of stages [28, 79].

In the following three subsections, we describe three major on-chip router building blocks: (1) buffers, (2) crossbar switch and (3) allocators and arbiters.

2.3.2 Buffer Organization

Buffer organization has a large impact on network latency, as it influences how efficiently packets share link bandwidth. The buffers store the incoming flits, and house them until they depart the router. This is in contrast to a processor pipeline that latches instructions in buffers between each pipeline stage. If source routing is not used, the route computation block will compute (or look up) the correct output port for the incoming packet. The virtual channel and switch allocators determine which flits are selected to proceed to the next stage where they traverse the crossbar. Finally, the crossbar switch is responsible to physically move flits from the input port to the output port [48].

Router buffers can be built using flip-flops or memory cells (e.g., static random access memories (SRAMs) and register files), depending on given area and performance constraints. For very small buffers, flip-flops suffice and can be readily synthesized without requiring memory generators. Flip-flops, however, have much poorer area, power and delay characteristics compared to SRAMs and register files.

A better approach to implement flip-flop-based FIFOs is to use a matrix of flip-flops with *write* and *read pointers* to avoid write and read energy consumption at every cycle due to shifts (e.g., as in shift registers). To implement this, we add control circuitry

to an existing matrix of flip-flops to handle the operation of write/read pointers. The write pointer points to the head of the queue and the read pointer points to the tail of the queue. The pointer advances one position for each write or read operation.

However, with larger buffer sizes, SRAMs prevail [113]. Most of the proposed on-chip routers are input-buffered where the buffering is done at the input ports. This is because input buffer organization allows use of power- and area-efficient single-port SRAM. For smaller buffer sizes, register file cells tend to occupy a smaller footprint as they do not require differential sense amplifiers, and can support faster access times.[1] On the other hand, for larger buffer sizes, SRAMs provide better integration density. Figure 2.2 shows the structure of a FIFO buffer implemented using SRAM. In Figure 2.2, T_p is the pass transistor connecting bitlines and memory cells; T_{wd} is the wordline driver; T_{bd} is the write bitline driver; T_c is the read bitline precharge transistor; and T_m shows the memory cell cross-coupled inverters. In addition, h_{cell}, w_{cell}, d_w, F, and B denote memory cell height, memory cell width, wire spacing, flitwidth, and buffer size in flits, respectively.

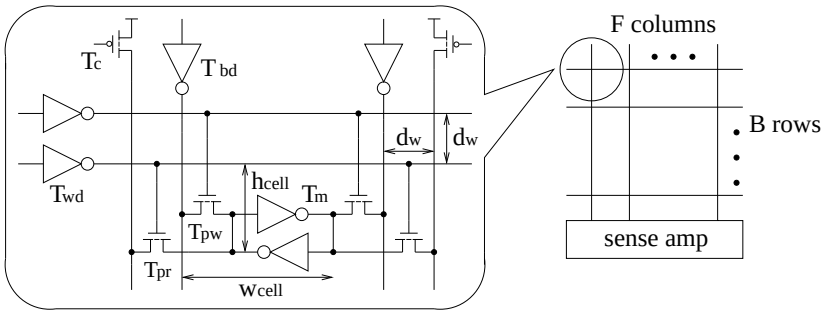

Figure 2.2: SRAM FIFO with one write and one read port [63].

In an input-buffered router, potential buffering organization schemes are as follows.

- *Single fixed-length queues.* In this scheme, incoming flits are written into the tail of the queue, while the flit at the head of the queue is read and sent through the crossbar switch and onto the output links (when it wins arbitration). The single

[1] This thesis focuses on the input-buffer router.

queue has a fixed length, so the upstream router can keep track of buffer availability and ensure that a flit is only forwarded if there is a free buffer downstream. Such a buffer organization works with both store-and-forward and wormhole flow control. However, a single queue can lead to situations where a packet at the head of the queue is blocked (as its output port is held by another packet), while a packet further behind in the queue whose output port is available cannot make forward progress because it has to wait for the head of the queue to clear. Such unnecessary blocking is known as *head-of-line blocking* [45, 61].

- *Multiple fixed-length queues.* Having multiple queues at each input port alleviates head-of-line blocking problem. Each of the queues is called a VC; multiple VCs multiplex and share a given physical link.

- *Multiple variable-length queues.* A potential drawback of the previous scheme occurs when there is imbalance in the traffic. One VC can be full and unable to accept further flits while another VC is empty, which results in poor buffer utilization and hence low network throughput. To overcome this problem, each VC queue can be variable-length, sharing a large buffer resource [107]. This allows better buffer utilization, but at the expense of more complex circuitry to keep track of the head and tail of the queues. Also, to avoid deadlocks and ensure forward progress, one flit buffer needs to be reserved for each VC so that other VCs do not fill up the entire shared buffer.

Among all router resources, buffers consume a significant (e.g., 30% [57, 67, 74, 63]) portion of the total router power. Hence, minimizing the number of buffers is important to reduce router power consumption. Therefore, determining the optimal buffer size and VC allocation is of critical importance to maximize performance and minimize power consumption. In Chapter 5, we propose new *trace-driven* approaches to more efficiently allocate virtual channels to further reduce power without any performance penalty, using the application knowledge.

2.3.3 Switch Design

The crossbar switch is the main part of the router datapath where it connects the input and output ports. It is well known that the crossbar switch area increases as a square function of the number of router ports $O(n_{port}^2)$, where n_{port} denotes the number of router ports [44, 48, 79]. The crossbar dominates a large portion of the total router area (e.g., up to 53% [115]). Therefore, design of crossbar switch for high-performance and low-power applications is a challenge, such as the bit-interleaved or double-pumped custom crossbars used in the Intel Teraflops chip [57].

In Chapter 4, we model the following two crossbar designs: (1) multiplexer tree and (2) matrix [63].

- *Multiplexer tree.* A multiplexer tree crossbar consists of several multiplexers to set up the connection of the switch. It determines which input port(s) should be connected to which output port(s). Most low-frequency router designs use this type of crossbar as it can be easily synthesized using existing standard cell library multiplexers.

- *Matrix.* A matrix crossbar has a crosspoint-based organization with select signals feeding each crosspoint. The setup for the connection of the switch is shown in Figure 2.3. In this figure, the small square box represents a connector, which can be either a tristate buffer or a transmission gate. Also, T_{id} and T_{od} denote input and output drivers, respectively.

A router microarchitect needs to decide on the crossbar switch speedup, namely, the number of input and output ports in the crossbar relative to the number of router input and output ports. Crossbars with higher speedups provide more internal bandwidth between router input and output ports. This eases the allocation problem and improve flow control. If each VC has its own input port to the crossbar, a flit can be read out of every VC every cycle, so multiple VCs need not contend for the same crossbar input port. For example, a 10×5 crossbar achieves close to 100% throughput even with a simple allocator (allocators are discussed in the next subsection) [48]. By providing more inputs to select from, there is a higher probability that each output port will be matched each cycle. The use of output speedup allows multiple flits to be sent to

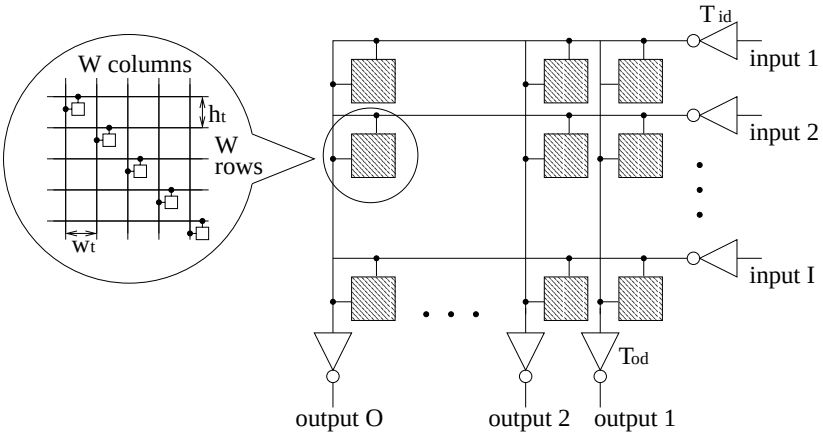

Figure 2.3: Matrix crossbar with I input ports and O output ports. [63].

the same output port each cycle, thus reducing the contention. A crossbar with output speedup requires output buffers to multiplex flits onto a single output port [44, 48].

With the crossbar taking up a significant portion of a router's footprint and power budget, microarchitectural techniques that target optimizing crossbar power and performance are of great importance. One approach is to *dimension-slice* a crossbar. For example, in a 2-dimensional mesh, two 3×3 crossbars can be used instead of one 5×5 crossbar, with the first crossbar dedicated to the traffic that remains in the X-dimension, and the second crossbar dedicated to the traffic remaining in the Y-dimension [44, 83]. A port on the first crossbar connects with a port on the second crossbar while those remaining within a dimension only traverse one crossbar as shown in Figure 2.4.[2] In Figure 2.4, the arrows represent the corresponding input and output ports. This is particularly suitable for the dimension-ordered routing protocol where traffic mostly stays within a dimension. On the other hand, bit interleaving the crossbar aims to reduce flitwidth fw, instead. It sends alternate bits of a link on the two phases of a clock on the same line, thus halving fw. The Intel Teraflops chip [57] architecture uses bit interleaving.

2.3.4 Arbiters and Allocators

An allocator matches multiple requests to multiple resources. An arbiter matches multiple requests to only one resource. In a wormhole router, the switch arbiter at each

[2]Figure adapted from Figure 7.5 in [44].

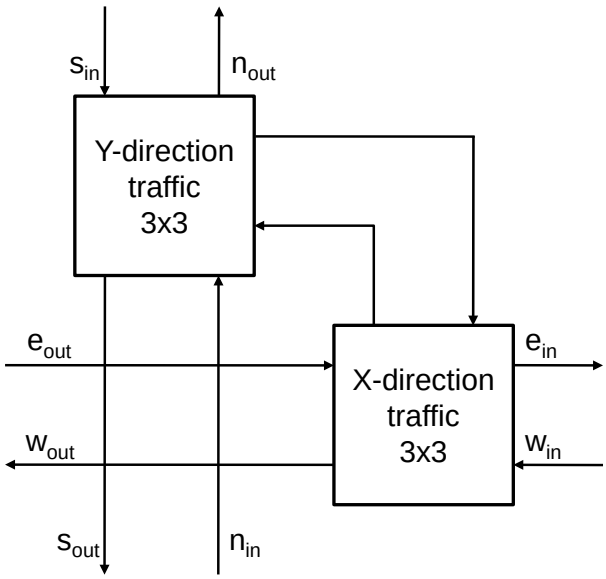

Figure 2.4: Dimension-slicing: two one-dimensional crossbars, one carrying Y-direction traffic and one carrying X-direction traffic.

output port matches and grants that output port to requesting input ports. Hence, there are n_{port} arbiters, one per output port, where each arbiter matches input port requests to the single output port under contention.[3]

In a VC router, we need (1) a virtual-channel allocator (VA) and (2) a switch allocator (SA). The VA resolves the contention for output VCs and grants them to input VCs. The SA grants crossbar switch ports to input VCs. Only the head flit of a packet needs to access the virtual-channel allocator, while the switch allocator is accessed by all flits. The SA grants access to the switch on a cycle-by-cycle basis. An allocator/arbiter that delivers high matching probability translates to more packets successfully obtaining virtual channels and passage through the crossbar switch, and thereby leads to higher network throughput. Allocators and arbiters must be fast and able to be pipelined so that they can work at high clock frequencies [48].

In Chapter 4, we model two arbiter implementations: (1) round-robin and (2) matrix.

- *Round-robin arbiter.* With a round-robin arbiter, the last request to be serviced

[3] In this thesis, we assume that the number of input and output ports are equal. However, our developed approaches are not restricted to routers with equal number of input and output ports.

will have the lowest priority in the next round of arbitration. For example, Figure 2.5 shows a set of requests from four different requestors.[4] Suppose that the last request serviced prior to the new set of requests was from Requestor A. Hence, Requestor B has the highest priority at the start of our example. With the round-robin arbiter, requests are satisfied in the following order: B_1, C_1, D_1, A_1, D_2, A_2.

- *Matrix arbiter.* A matrix arbiter operates so that the least recently served requestor has the highest priority. A triangular array of state bits is used to implement priorities. Once bit (i,j) is set, i has a higher priority than j. Figure 2.6 shows a circuit representation of a matrix arbiter with R requests. Further functional details of the matrix arbiter are explained in Chapter 4.

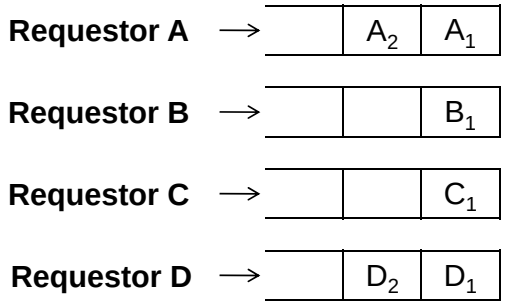

Figure 2.5: An example of request queues and corresponding requestors in a round-robin arbiter.

Figure 2.7 shows how each stage of a 3 : 4 separable allocator (an allocator matching 3 requests to 4 resources) is composed of arbiters. The 3 : 1 arbiters in the first stage decide which of the three requestors win a specific resource, while the 4 : 1 arbiters in the second stage ensure that a requestor is granted just one of the four resources. Different arbiters are used in practice, with round-robin arbiters being the most popular due to their simplicity [44].

We also modify the separable VC allocator microarchitecture used in previous work (e.g., ORION 1.0) to optimize its power consumption. Instead of two stages of arbiters, we have a single stage of $n_{port} \times n_{vc}$ arbiters, each governing one specific

[4]Figure adapted from Figure 18.6 in [44].

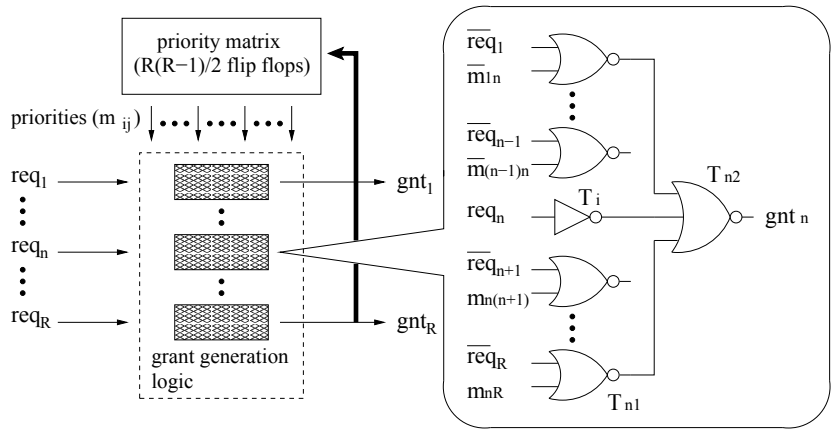

Figure 2.6: An example matrix arbiter with R requests [63].

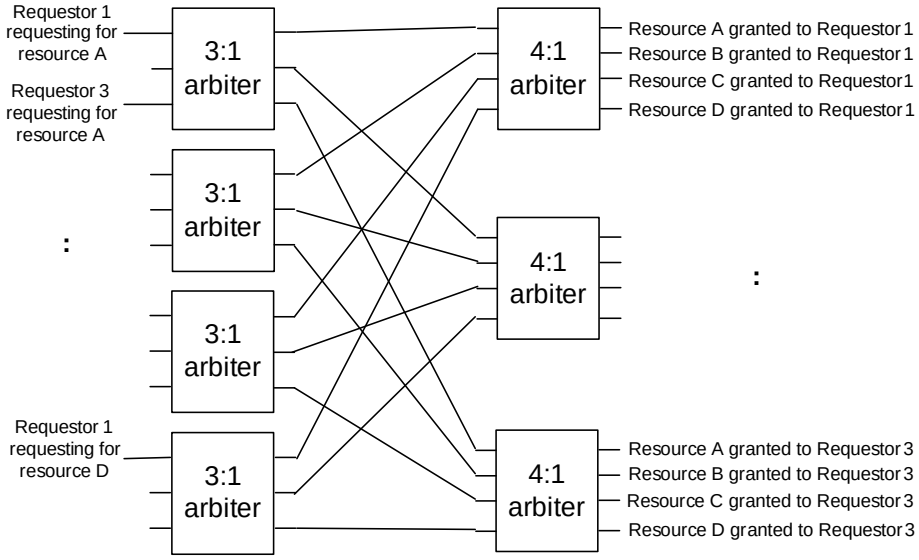

Figure 2.7: A separable 3 : 4 allocator (3 requestors, 4 resources) which consists of four 3 : 1 arbiters in the first stage and three 4 : 1 arbiters in the second stage [44].

output VC, where n_{port} and n_{vc} are the numbers of router ports and VCs, respectively. Instead of sending requests to all output VCs of the desired output port, an input VC first checks the availability of output VCs, then sends a request for any one available output VC. The arbiters will resolve conflicts where multiple input VCs request the same output VC. This design has lower matching probability but uses only one stage of arbiters; hence, it significantly saves power. We also add a new VC allocator model

in ORION 2.0 which models VC allocation as VC "selection" as was first proposed in [69]. Here, a VC is selected from a queue of free VCs, *after* switch allocation. Thus, the complexity (delay, power and area) of VC allocation does not grow with the number of VCs.

2.4 Communication Synthesis

NoCs have been proposed as the solution to the problem of connecting an increasing number of processing cores on the same die [28, 43, 55]. Key steps in the optimization of the NoC design include topology selection and assignment of routes for packets as they travel from a source core to a destination core. Some network design ideas can be borrowed from the computer science community that address the same problems for local area networks and supercomputer networks. However, the challenge is to leverage the intrinsic characteristics of on-chip communication to achieve both energy efficiency and high performance [79].

Each target technology offers a variety of possibilities to the NoC designer who can decide on the number and position of network access points and routers as well as which metal layer to use to implement each given channel. Since the design space of the possible topologies is large, choosing the best one is a difficult problem that cannot be solved only by experience. In fact, the problem is even harder given the heterogeneity of the cores and the traffic patterns among them. Therefore, the development of automatic tools to design NoCs is a key enabler for the success of the NoC design paradigm.

In this thesis, Chapters 3 and 4 develop (power, performance, and area) cost models for on-chip routers and interconnects to enable efficient and accurate design space exploration of NoCs. To assess the impact of the proposed models on system-level optimization outcomes, we integrate our models into COSI-OCC, a system-level NoC optimization tool. COSI-OCC is an open-source software infrastructure for the automatic synthesis of On-Chip Communication (OCC) [92]. Figure 2.8 shows the design flow implemented in COSI-OCC. The input is a *project* file that contains pointers to the *communication constraint* file and to the *library* file. The constraint file contains the description of the IP cores and the communication constraints among them. An IP

core can be manually placed on the chip, thereby having fixed position and dimensions, or it can be characterized by its area only. If there are unplaced IP cores, PARQUET [19] is used to floorplan the chip. An end-to-end communication constraint is defined by a source core, a destination core, a minimum bandwidth and a maximum number of hops.

The library file contains the description of the library elements. Each element is characterized by a set of architectural parameters (e.g., flitwidth, maximum number of router ports, etc.), and a model that defines its performance and cost (in terms of power and area). The models used in the library file are developed using the approaches explained in Chapters 3 and 4. The user can select the appropriate synthesis algorithm to derive an implementation depending on the optimization goal (minimum power, minimum area or minimum delay). The development of new synthesis algorithms is facilitated by the simple, standard interface with the library. This defines an application programming interface (API) to retrieve the performance and cost of a component (e.g., a point-to-point link) given its configuration (e.g., clock speed and total bandwidth). Such an API is of extreme importance to system-level designers who do not wish to be concerned with low-level technology details.

Chapter 3 proposes accurate system-level interconnect power, performance and area models that can be easily used in system-level optimization of NoC designs. Models developed in Chapter 3 also consider effects such as supply voltage and temperature variations which are artifacts of the advanced silicon and emerging technologies (e.g., 3D Integrated Circuits (3DIC)). In addition, Chapter 4 proposes accurate and portable power, performance and area models for on-chip routers which enable efficient system-level optimization of NoCs across different microarchitectures and circuit implementations. The results of our proposed cost models can be easily integrated into existing system-level optimization tools such as COSI-OCC (Figure 2.8). We have released our on-chip router models, ORION 2.0, which now provide the most widely used on-chip router power and area models in both academia and industry [12]. In addition to our modeling efforts, we propose new trace-driven NoC optimization approaches to further reduce router power with no penalty on performance. To do this, we take an existing NoC solution (e.g., provided by a system-level tool such as COSI-OCC) and, using the

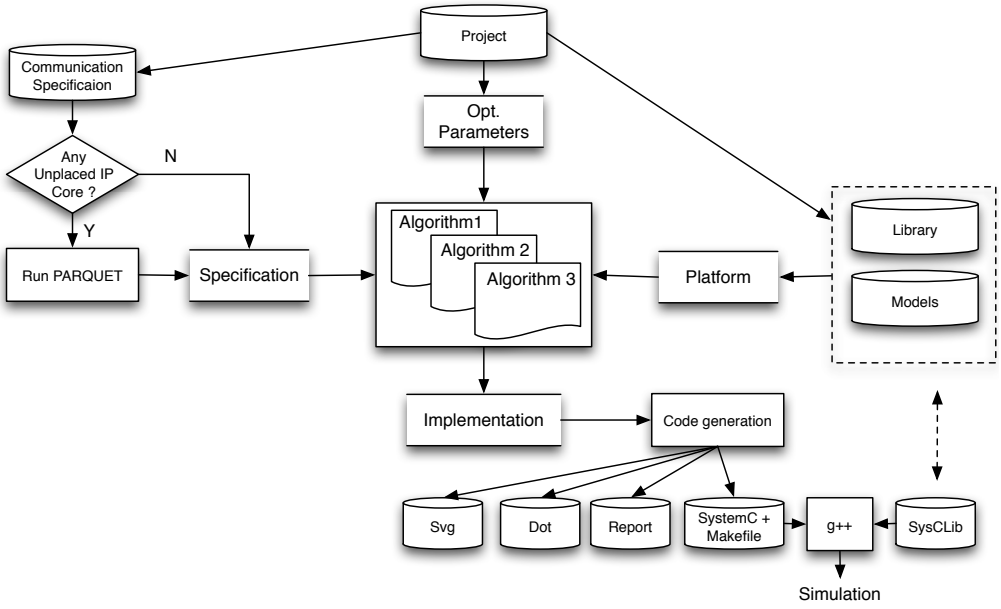

Figure 2.8: COSI-OCC design flow [35].

application knowledge, efficiently allocate virtual channels to it. Our combined modeling and optimization work aims to improve the understanding of the achievable NoC power-performance-area design space.

Chapter 3

On-Chip Wire Power, Performance and Area Modeling

3.1 Introduction

Due to increasing complexity of Systems-on-Chip (SoCs) and poor scaling of interconnects with successive technology nodes, on-chip communication has become a performance bottleneck and a significant consumer of power and area budgets [56, 105]. Decisions made in the early stages of the design process have the largest potential to optimize the system with respect to power and other objectives [93]. Therefore, to achieve meaningful optimizations and to reduce guardbanding, it is crucial to account for interconnects during system-level design by modeling their performance, power and area.

During system design, organizational and technological choices are made. At this stage, the design team is concerned with implementing the hardware architecture determined during the conceptualization and modeling phases of design. The design process is supported by hardware synthesis tools and software compilers. Energy efficiency can be achieved by leveraging the degrees of freedom of the underlying hardware technology. Even within the framework of a standard implementation technology, there are ample possibilities to reduce power consumption. System-level decisions affect primarily the global interconnects by setting their lengths, bitwidths and speed require-

ments. Local interconnects are typically less affected as they are either already routed in IP blocks or routed by automatic back-end routing tools.

This chapter focuses on interconnect delay, power and area models that are usable by the system-level designer at an early phase of the design process. We first study what system-level designers require in a model of global interconnects, then discuss the shortcomings of the models that are presently used. We then propose new predictive models and present a reproducible methodology to obtain them. Since the accuracy of such models relies on the accuracy of the underlying technology parameters, we also highlight reliable sources that are easily available to the system-level designer for present and future technologies. We compare predictions from our proposed interconnect models with existing models and show the impact of improved accuracy on system-level design choices by contrasting the NoC topologies generated by the COSI-OCC [92] using both existing models and our models.

We also propose a new framework for gate delay modeling under supply voltage and temperature variations, using *machine learning-based nonparametric regression* methods. We develop early-stage delay models to enable worst-case performance prediction that can drive efficient circuit optimization (e.g., of on-chip power delivery networks). Finally, we validate the proposed models using layout and SPICE simulation data. The contributions of this chapter are as follows.

- We define the requirements that a system-level model for global buffered interconnects should satisfy, and we then discuss the shortcomings of the models available in the literature.

- We present our predictive models together with a *reproducible* methodology to derive them.

- We build our models through accurate experimentations and calibrations against industry technology files, and provide necessary explanations of the models and associated parameters. We apply linear and quadratic regressions to obtain the fitting coefficients of our predictive models.

- We propose a new modeling methodology that uses machine learning-based nonparametric regression methods to develop an accurate closed-form interconnect

performance model under supply voltage and temperature variations.

- Finally, we demonstrate that nonparametric regression methods can enable automatic generation of interconnect models based on circuit and architecture-level parameters.

The remainder of this chapter is organized as follows. Section 3.2 describes requirements for models to be usable at the system level. Section 3.3 describes the detailed methodology to develop predictive interconnect power, performance and area models. In Section 3.4, we propose the use of machine learning-based nonparametric regression to model interconnect performance under supply voltage and temperature variations. We also show the extensibility of nonparametric methods to the modeling of metrics that go beyond power, performance and area; specifically, we exhibit new models of interconnect wirelength and fanout. Finally, Section 3.5 concludes the chapter.

3.2 Model Requirements

System-level designers require *accurate yet simple* models of implementation fabrics (e.g., communicating entities and interconnections between them) to bridge planning and implementation, and enable meaningful design optimization choices. Today, performance and power modeling for system-level optimization suffers from

- poor definition of inputs required to make design choices;
- *ad hoc* selection of models as well as sources of model inputs;
- lack of model extensibility across multiple (and future) technologies; and
- inability to explore different implementation choices and design styles.

In this section, we discuss the accuracy and extensibility of previous models, as well as key modeling deficiencies that this thesis addresses.

3.2.1 Accuracy

Communication mechanisms between subsystems (such as IP blocks and routers) are realized using high-speed bus structures or point-to-point interconnects. The delay and bandwidth envelope of such interconnects is defined by optimally buffered structures, and must be accurately modeled to enable synthesis of optimal (e.g., minimum-latency or minimum-power) communication topologies. Just as with technology mapping in logic synthesis, on-chip communication synthesis is driven by models of latency and power consumption. The accuracy of such models should be comparable to that available during physical synthesis due to the high sensitivity of design outcomes. For example, poor models of interconnect latency can increase hop count and introduce unneeded routers between communicating blocks; this in turn can increase chip area and power consumption.

Existing methods for on-chip communication synthesis [91] and analysis [55] primarily use "classic" delay and power models of Bakoglu [24], or more recently of Pamunuwa *et al*. [87]. The popularity of these models with the NoC research community is likely due to the following reasons.

- *Simplicity and ease of use.* Bakoglu's delay model for buffered interconnect lines [24] is based on lumped approximation of the distributed parasitics of the interconnect. Drivers and buffers are modeled as simple voltage sources with series resistance connected to the interconnect load. These approximations make the buffered interconnect delay model amenable to analytical, closed-form representation, and hence adoptable in NoC synthesis flows.

- *First-order accuracy.* Bakoglu and Pamunuwa *et al.* use a simple step voltage source with series resistance to represent a driver and a buffer. Interconnect load is lumped at the output of the cell to compute cell delay. Interconnect delay is computed as *Elmore delay*, i.e., the first moment of the impulse response of the distributed RC line.

- *Inertia.* There have not been any compelling reasons to use alternative, more accurate models. To this point, we show that accurate models can still be simple,

and that different optimization results and trends follow from use of improved models.

The remainder of this subsection lists key factors that are not addressed by existing delay models. In 180 nm and below process nodes, these factors lead to inaccuracy in delay (latency) computation.

- *Transition time (slew) dependence.* A simple cell delay model of step voltage source with constant series resistance fails to capture the impact of input slew on delay. A finite input slew rate changes the drive resistance and cell delay, as well as the output voltage waveform that drives other cells. To the best of our knowledge, none of the delay models used in the NoC literature considers the impact of slew on delay.

- *Interconnect resistivity.* Resistance directly affects interconnect delay and shields the load capacitance seen by driving buffers. As interconnect dimensions continue to scale, electron scattering has started to affect the resistivity [6, 100]. Also, copper interconnect manufacturing requires use of a barrier layer that reduces the effective width and thickness of the metal. Existing delay models ignore these two effects and thus incur a considerable loss of accuracy.

- *Coupling capacitance.* Crosstalk from capacitive coupling affects signal transition time and delay along interconnects. Classic models such as Bakoglu's do not consider coupling between neighboring interconnects, and hence are oblivious to the resulting delay and transition time changes. Pamunuwa *et al.* consider the impact of switching activity on the 'Miller' coupling between neighboring lines, and hence on delay, but fail to model the impact on transition time. This leads to inaccurate delay computations for cells driven by the affected signal.

The aforementioned deficiencies in gate and wire delay models are addressed to some extent in the large body of works on gate delay [23, 45] and interconnect delay [90, 94] modeling. However, such models (e.g., AWE-based approaches [94]) need detailed interconnect parasitic information which is unavailable at the system-level design phase. For gate delay, works such as that of Arunachalam *et al.* [23] model input voltage as a

piecewise-linear function and choose the value of series resistance more elaborately. The main drawback of such approaches is that they model drive resistance independently of input transition time (slew). In reality, drive resistance (r_{drv}) varies with input slew and also affects output slew. Shao *et al.* [99] recently proposed a gate delay model that relies on a second-order RC model of the gate. They propose analytical formulas to compute the output voltage waveform for a given ramp input waveform. However, they do not address gate loading during model construction. For a gate delay model to be accurate, drive resistance dependence on input slew, and output slew dependence on load capacitance and input slew, must both be considered.

3.2.2 Design Styles and Buffering Schemes

System-level designers usually ignore design-level degrees of freedom such as wire shielding, wire width sizing and spacing, etc. when modeling interconnect latency and power. Yet, optimizations of design styles or buffering schemes can have a huge impact on the envelope of achievable system performance. For example, shielding an interconnect with lines tied to power and/or ground rails on both sides reduces worst-case capacitive coupling and improves delay. Wire width sizing and spacing also improve delay. This is because when we make a wire wider, interconnect resistance decreases linearly while capacitance increases sublinearly. Wire capacitance has two components: (1) ground capacitance and (2) coupling capacitance. When wires become wider, only the ground capacitance increases, while the coupling capacitance remains the same. This decreases the overall RC constant , and hence reduces interconnect delay.

In addition to design style choices, the buffering objective can also be significant. Interconnect delay models of Bakoglu and Pamunuwa *et al.* incorporate buffering schemes that minimize end-to-end delay (*min-delay* buffering), and are used extensively in the NoC literature. However, min-delay buffering can result in unrealistically large buffer sizes, and hence high dynamic and leakage power. Another common buffering objective is to minimize the *energy-delay* product. As shown in Figure 3.1, the optimal buffer size using min-delay objective can be 480 times that of the minimum-sized inverter. When optimizing the energy-delay product, the inverter size drops to 50–60 times minimum. Hence, it is necessary for system-level design optimization to compre-

hend power-aware buffering schemes and, more generally, the key circuit-level choices that maximize achievable performance.

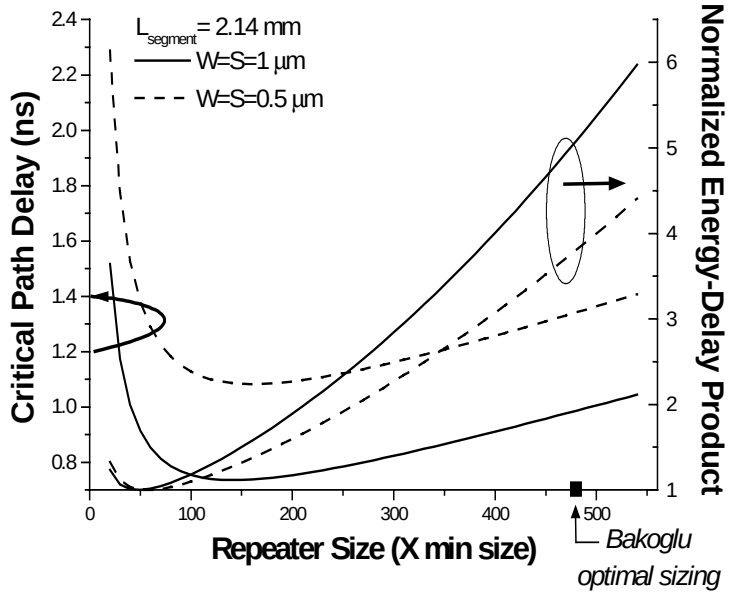

Figure 3.1: Comparison of min-delay and energy-delay product objectives for buffer insertion [34].

3.2.3 Model Inputs and Technology Capture

Perhaps the most critical gap in existing system-level and NoC optimizations is the lack of well-defined pathways to capture necessary technology and device parameters from the wide range of available sources. Since exploration of the system-level performance and power envelope is typically done for current and future technologies, the models driving system-level design must be derivable from standard technology files (e.g., Liberty format [8], LEF [7]), as well as extrapolatable models (e.g., PTM [13], ITRS [6]). Earlier works on NoC design space exploration and synthesis [91, 102] collect inputs from *ad hoc* sources to drive internal models of performance, power and area. However, since exploration is being performed at a very high level, the use of non-standard interfaces and data sources can often lead to misleading conclusions that can have significant impact on the final outcome. Instead, inputs that accompany system-

level models must come from standard sources and be conveyed through standardized interfaces and formats.

3.3 Buffered Interconnect Model

In this section, we describe our models and present a methodology to construct them from reliable sources for existing and future technologies. We account for previously-ignored effects such as slew-dependent delay and scattering-dependent wire resistivity changes. Our models are by construction calibrated against SPICE and contain relevant circuit- and architecture-level parameters.

3.3.1 Repeater Delay Model

We now present our *repeater* delay model and describe its derivation.[1] For brevity, the following study is presented only for rise transitions in inverters implemented in 65 nm technology. The derived functional forms are identical for fall transitions. For buffers and other technology nodes, only the function coefficients change.

Repeater delay can be decomposed into *load-independent* and *load-dependent* components as

$$t_{rpt} = t_i + r_{drv} \times c_{load} \tag{3.1}$$

where t_{rpt} is the repeater delay and t_i is the load-independent or intrinsic delay of the gate. $r_{drv} \times c_{load}$ is the load-dependent delay term, where r_{drv} is the drive resistance and c_{load} is the load capacitance.

The intrinsic delay, t_i, can potentially depend on the input slew of the gate and the gate size. However, as shown in Figure 3.2, t_i is practically independent of the gate size, and depends nonlinearly on the input slew. The independence of intrinsic delay from gate size can be understood as follows. For inverters, larger sizes are attained by connecting in parallel multiple identical devices (fingers), which switch simultaneously and have negligible impact on each other. As the inverter size increases, the number of parallel-connected devices increases, but the intrinsic delay remains unaffected due to

[1] We use the term 'repeater' to denote both an inverter and a buffer.

the independent switching of the devices. For buffers, the intrinsic delay additionally comprise the delay of the inverter in the first stage which drives the inverter in the second stage. As the buffer size increases, the size of the second-stage inverter increases and the size of the first-stage inverter also increases to maintain small intrinsic delay. Consequently, the total intrinsic delay of buffers is nearly independent of the buffer size.

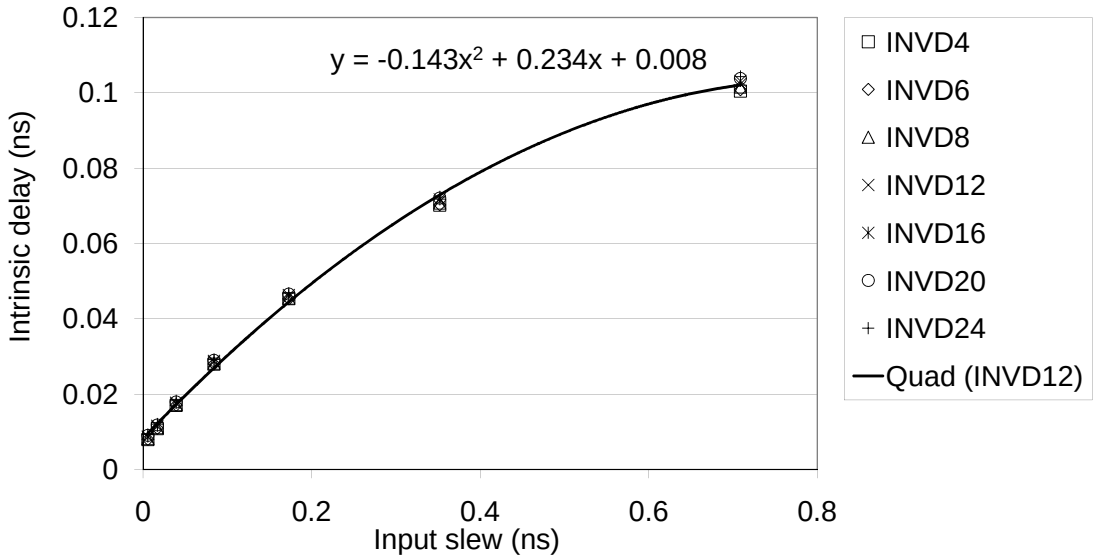

Figure 3.2: Dependence of repeater intrinsic delay on input slew and inverter size. Intrinsic delay is essentially independent of repeater size and depends nonlinearly on input slew.

The intrinsic delay is nonlinear with respect to input slew. We have found a quadratic relationship to be a good tradeoff between simplicity and accuracy. The quadratic dependence of intrinsic delay on input slew is captured by the equation

$$t_i(t_{slew,in}) = \alpha_0 + \alpha_1 \times t_{slew,in} + \alpha_2 \times t_{slew,in}^2 \tag{3.2}$$

where $t_{slew,in}$ denotes the input slew, and α_0, α_1 and α_2 are the coefficients determined by quadratic regression. The dependence of drive resistance on input slew has often been ignored [24, 87, 40], but this can contribute to substantial error in delay prediction. Figure 3.3 shows the dependence of r_{drv} on input slew and repeater size. We observe that r_{drv} is nearly linear with input slew, especially for larger input slew values. We also

note that both the intercept and slope vary with repeater size; hence, r_{drv} can be written as

$$r_{drv} = r_{drv0} + r_{drv1} \times t_{slew,in} \tag{3.3}$$

where r_{drv0} and r_{drv1} are coefficients that depend on the repeater size.

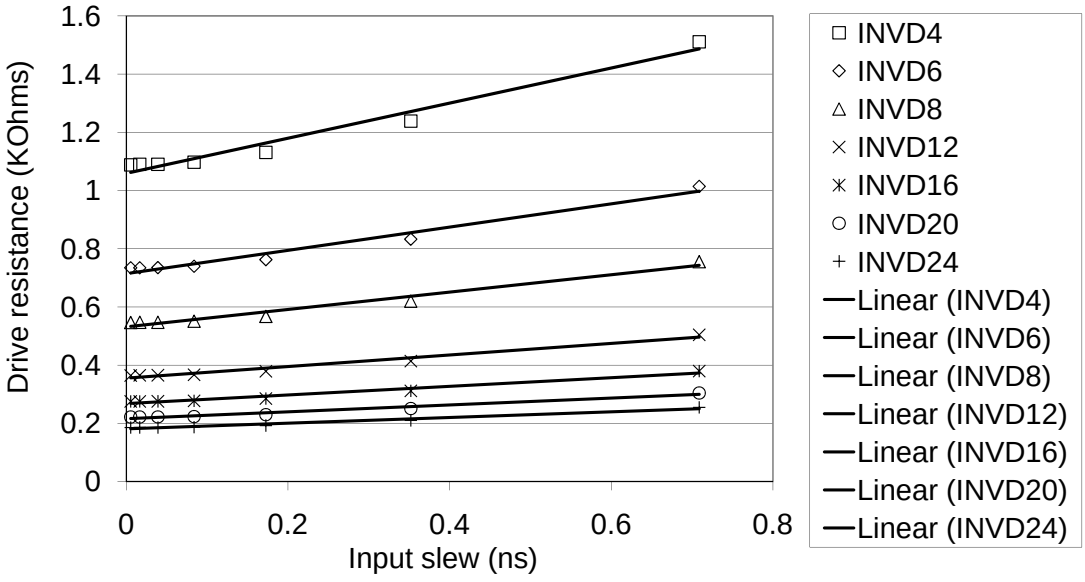

Figure 3.3: Dependence of drive resistance on input slew and repeater size. Drive resistance depends linearly on the input slew. Both the intercept and the slew are affected by the repeater size.

Both r_{drv0} and r_{drv1} can be readily calculated using linear regression for a few repeater sizes. Previous works (e.g., [24]) have assumed r_{drv} to be inversely proportional to the repeater size. We have confirmed this relationship to be sufficiently accurate for sub-90 nm technology modeling. To be precise, we use the PMOS (NMOS) device width as the repeater size for rise (fall) transition. As shown in Figure 3.4, both r_{drv0} and r_{drv1} are linearly proportional to the inverse of the repeater size, and the exact coefficients can be calculated using linear regression with zero intercept.[2] Thus,

[2]All graphs are generated using simple SPICE simulations for a set of input slew values, output capacitance values and repeater sizes.

$$r_{drv0}(size_{rpt}) = \beta_0/size_{rpt} \tag{3.4}$$

$$r_{drv1}(size_{rpt}) = \beta_1/size_{rpt} \tag{3.5}$$

where $size_{rpt}$ is the repeater size which is equal to the PMOS (NMOS) width for rise (fall) transitions, and β_0 and β_1 are fitting coefficients.

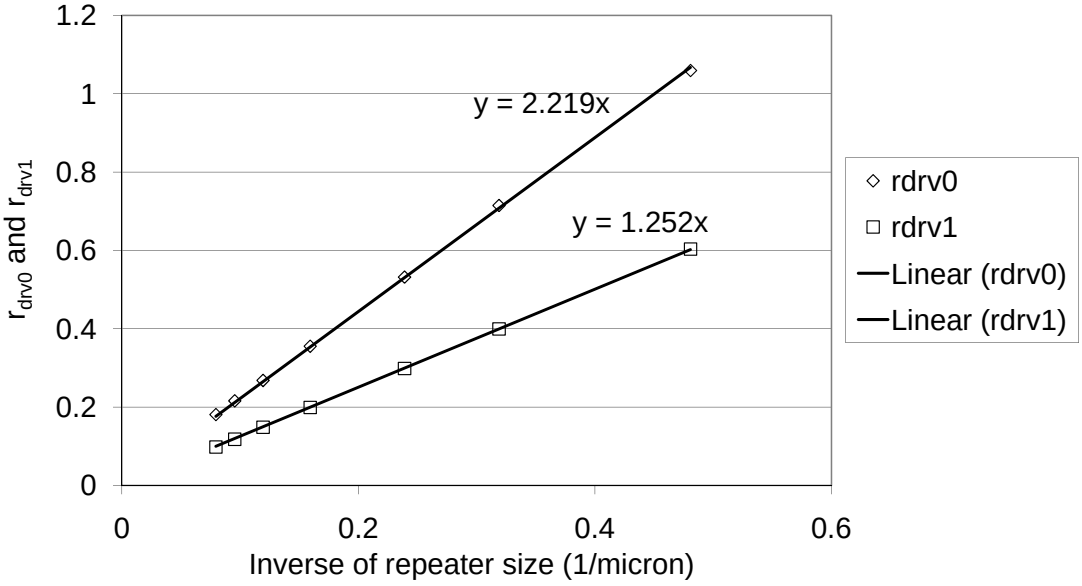

Figure 3.4: Coefficients r_{drv0} and r_{drv1} vary linearly with the inverse of the repeater size with zero intercept.

Repeater Output Slew Model

Since our gate delay model depends on input slew, we must also model output slew of the previous stage of the buffered interconnect. Slew is not a crucial metric at the system level, and its only use arises in delay calculation. Furthermore, while repeater delay depends on slew, inaccuracies arising in slew estimation tend to be masked in delay calculation. As a result, accuracy requirements for the slew model are less stringent than those for the delay model.

As with gate delay, slew depends on repeater size, input slew and load capacitance. Figure 3.5 shows the dependence of output slew on load capacitance and input

slew. Output slew depends strongly on the load capacitance, and we have found a linear relationship to be a good tradeoff between simplicity and accuracy. Note that the slope is nearly independent of the input slew, while the intercept is dependent on it. Assuming that the intercept depends linearly on the input slew, the output slew for a given repeater can be written as

$$t_{slew,out}(c_{load}, t_{slew,in}) = s_{o0} + s_{o1} \times t_{slew,in} + s_{o2} \times c_{load} \tag{3.6}$$

where $t_{slew,out}$ is the output slew, and s_{o0}, s_{o1} and s_{o2} are fitting coefficients readily derived from multiple linear regressions.

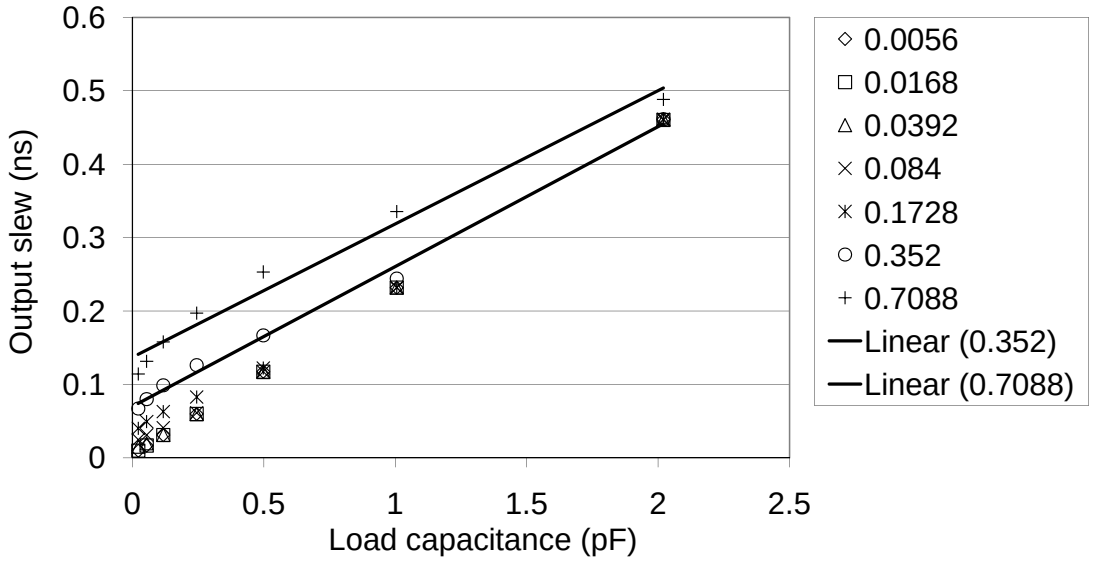

Figure 3.5: Dependence of output slew on load capacitance and input slew. Output slew depends linearly on load capacitance. The slope of the linear fit is nearly independent of the input slew, but the intercept depends on it.

The impact of repeater size on the coefficients s_{o0}, s_{o1} and s_{o2} is shown in Figure 3.6. We consistently observe that s_{o0} and s_{o2} are independent of the repeater size, but s_{o1} varies inversely with repeater size. Hence, output slew can be calculated as

$$t_{slew,out}(c_{load}, t_{slew,in}, size_{rpt}) = \gamma_0 + \frac{\gamma_1 \times c_{load}}{size_{rpt}} + \gamma_2 \times t_{slew,in} \tag{3.7}$$

where γ_0, γ_1 and γ_2 are constants.

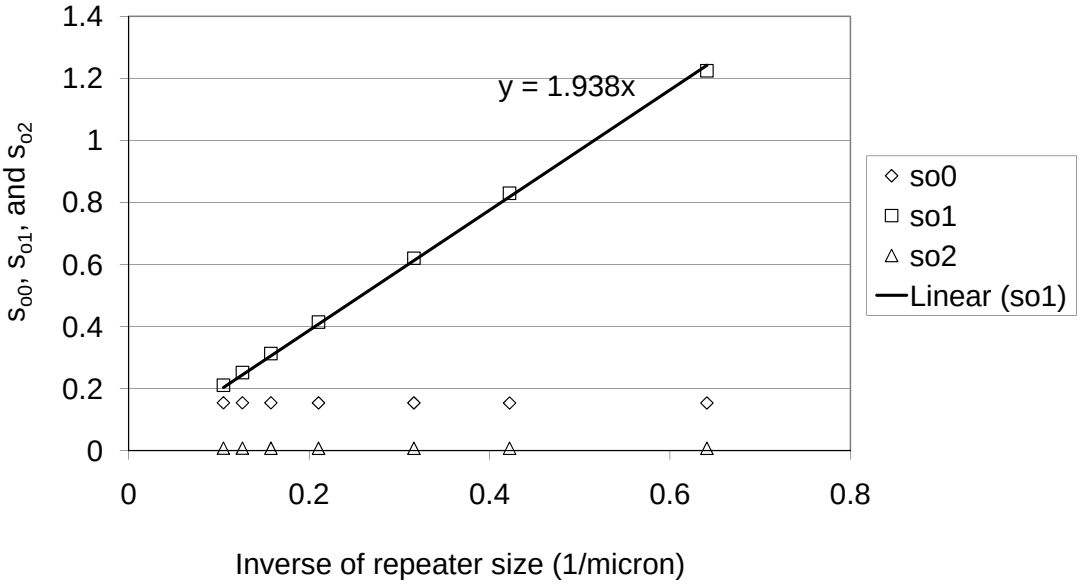

Figure 3.6: Dependence of coefficients s_{o0}, s_{o1} and s_{o2} on inverse of repeater size. s_{o0} and s_{o2} are independent of repeater size, while s_{o1} varies inversely with repeater size.

Repeater Input Capacitance Model

The input capacitance of a repeater is required to calculate the load capacitance of the previous stage. As expected, the input capacitance is proportional to the repeater size. However, even if the PMOS/NMOS ratio changes with repeater size, input capacitance can be modeled as

$$c_{in} = \eta \times (w_{PMOS} + w_{NMOS}) \tag{3.8}$$

where c_{in} is the input capacitance, w_{PMOS} and w_{NMOS} are respectively PMOS and NMOS widths, and η is a coefficient derived using linear regression with zero intercept.

3.3.2 Wire Delay Model

For wire delay we use the model proposed by Pamunuwa *et al.* [87] which accounts for crosstalk-induced delay:

$$t_{int} = r_{int} \times (0.4c_g + \frac{\mu_i}{2}c_c + 0.7c_{in}) \tag{3.9}$$

where t_{int}, r_{int}, c_g, c_c and c_{in} respectively denote interconnect delay, wire resistance, ground capacitance, coupling capacitance and input capacitance of the next-stage repeater. The coefficient μ_i reflects switching patterns of neighboring wires and is equal to 1.51 for worst-case switching. We enhance the quality of the wire delay model by considering two important factors that affect wire resistance.

- *Scattering-aware resistivity.* The rapid rise of wire resistivity due to electron scattering effects (grain boundaries and interfaces) at small cross-sections poses a critical challenge for on-chip interconnect delay. For 65 nm and beyond, scattering can degrade delay by up to 70% [100] and must be accounted for in delay modeling. We adopt the following closed-form width-dependent resistivity equation [100]:

$$\rho(w_{int}) = \rho_B + \frac{K_\rho}{w_{int}} \tag{3.10}$$

  where w_{int} is the interconnect width, ρ_B = 2.202 $\mu\Omega$.cm, and K_ρ = 1.030$\times 10^{-15}$ Ω.m^2. The above model has been verified against measurement data from [95] and has been used in ITRS since 2004 [6].

- *Interconnect barrier.* To prevent copper from diffusing into surrounding oxide, a thin barrier layer is added to three sides of a wire. This barrier affects the wire resistance calculation as [56]:

$$r_{int} = \frac{\rho \times l_{int}}{(h_{int} - h_{barrier})(w_{int} - 2h_{barrier})} \tag{3.11}$$

  where h_{int} and $h_{barrier}$ respectively denote the interconnect and barrier heights, l_{int} is the interconnect length, and ρ is computed using Equation (3.10).

3.3.3 Power Models

Power is a first-order design objective and must be modeled early in the design flow [93]. In current technologies, leakage and dynamic power are the primary forms of power dissipation. In repeaters, leakage current flows in both output states; NMOS

devices leak when the output is high, while PMOS devices leak when the output is low. This is also applicable for buffers because the second-stage devices are the primary contributors due to their large sizes. Leakage power has two main components: (1) subthreshold leakage and (2) gate-tunneling leakage. Both components depend linearly on device size. Thus, repeater leakage can be calculated using:

$$p_{leak}^{rpt} = \frac{p_{leak}^{NMOS} + p_{leak}^{PMOS}}{2} \tag{3.12}$$

$$p_{leak}^{NMOS} = \kappa_0^n + \kappa_1^n \times w_{NMOS} \tag{3.13}$$

$$p_{leak}^{PMOS} = \kappa_0^p + \kappa_1^p \times w_{PMOS} \tag{3.14}$$

where p_{leak}^{NMOS} and p_{leak}^{PMOS} are the leakage power for NMOS and PMOS devices, respectively, and κ_0^n, κ_1^n, κ_0^p and κ_1^p are coefficients determined using linear regression. State-dependent leakage modeling can also be performed using Equations (3.13) and (3.14) separately.

In present and future technologies, the dynamic power of devices is primarily due to charging and discharging of capacitive loads (wire and input capacitance of the next-stage repeater). Internal power dissipation, arising from charging and discharging of internal capacitances and short-circuit power, is noticeable for repeaters only when the input slews are extremely large. Dynamic power is given by the well-known equations:

$$p_{dyn} = a \times c_{load} \times v_{dd}^2 \times f_{clk} \tag{3.15}$$

$$c_{load} = c_{in} + c_g + c_c \tag{3.16}$$

where p_{dyn}, a, c_{load}, v_{dd} and f_{clk} respectively denote the dynamic power, activity factor, load capacitance, supply voltage and clock frequency. The load capacitance is the sum of the input capacitance of the next repeater (c_{in}), and the ground (c_g) and coupling (c_c) capacitances of the wire driven.

3.3.4 Area Models

Since repeaters are composed of several fingered devices connected in parallel, repeater area grows linearly with the repeater size. For existing technologies, the area

can be calculated as

$$A_{rpt} = \tau_0 + \tau_1 \times w_{NMOS} \tag{3.17}$$

where A_{rpt} denotes repeater area, and τ_0 and τ_1 are coefficients found using linear regression. For future technologies, area values may not be available to perform linear regression. Hence, we propose the use of feature size, contacted pitch and row height – all of which become available early in the process and library development and are also predictable – to estimate area as:

$$n_{finger} = (w_{PMOS} + w_{NMOS} + 2 \times \lambda)/h_{row} \tag{3.18}$$

$$w_{row} = n_{finger} \times (\lambda + pitch_{cont}) + pitch_{cont} \tag{3.19}$$

$$A_{rpt} = h_{row} \times w_{row} \tag{3.20}$$

where n_{finger} is the calculated number of fingers, λ is the feature size, h_{row} is the row height, w_{row} is the calculated row width, and $pitch_{cont}$ is the contacted pitch. The area of global wiring can be calculated as

$$A_{int} = bw \times (w_{int} + s_{int}) + s_{int} \tag{3.21}$$

where A_{int} denotes the wire area, bw is the bitwidth of the bus, and w_{int} and s_{int} are the interconnect width and spacing computed from the width and spacing of the layer (global or intermediate) on which the wire is routed, and from the design style.

3.3.5 Overall Modeling Methodology

Our delay, power and area models can be derived from the following inputs.

- For *repeater delay calculation*, delay and slew values for a set of input slew and load capacitance values, along with input capacitance values, are required for a few repeaters. Since the coefficients are derived using regression, a larger data set improves accuracy. The required data set is available from Liberty and timing library format (TLF) files [8] or can be generated using SPICE simulations for existing technologies. Since libraries are not available for future technologies, SPICE simulations must be used along with SPICE netlists for repeaters and predictive device models such as PTM [13]. To construct the repeater netlists, a

PMOS/NMOS ratio is assumed (from previous technology experience or from expected PMOS/NMOS drive strengths) and kept constant for all repeaters; a variety of repeaters are constructed for different device sizes.

- For *wire delay calculation*, we require the wire dimensions and inter-wire spacings for global and intermediate layers. These values are available in LEF (lateral dimensions) and ITF (vertical dimensions) files for existing technologies, and in the ITRS for future and existing technologies.

- For *power calculations*, input capacitance (computed in repeater delay calculation) and wire parasitics (computed in wire delay calculation) are used. Additionally, device leakage is required and can be computed from Liberty and TLF library files or SPICE simulations.

- For *area calculations*, wire dimensions used in wire delay calculation are used for wire area. Repeater area is readily available for existing technologies in Liberty or LEF files or from layouts. For future technologies, ITRS A-factors can be used or Equations (3.18)-(3.20) can be used along with the feature size, row height and contacted pitch, all of which values are available early in the process and library development.

Finally, the total delay of a buffered interconnect is the sum of the delays of all repeaters and wire segments in it. We assume that there is negligible slew degradation and resistive shielding (of capacitive load) due to the wires. Table 3.1 lists the coefficients derived for TSMC 90 nm and 65 nm high-speed technologies.

3.3.6 Interconnect Optimization

Delay-optimal buffering optimizes the size and number of repeaters and has been addressed under simple delay models in [24, 87, 40]. However, delay-optimal buffering results in extremely large repeaters, having sizes that are never used in practice due to area and power consumption considerations. Cao *et al.* [34] showed that use of smaller buffers improves the energy-delay product significantly while only marginally worsening delay.

Table 3.1: Coefficients for our model derived from TSMC 90 nm and 65 nm technologies. α, β and γ are for the rise transition.

	α_0	α_1	α_2	β_0	β_1
90 nm	0.013	0.217	-0.088	3.008	1.494
65 nm	0.008	0.234	-0.144	2.219	1.252
	γ_0	γ_1	γ_2	η	κ_0^n
90 nm	0.015	5.553	0.128	0.0015	-6.128
65 nm	0.012	4.162	0.142	0.0011	-6.034
	κ_1^n	κ_0^p	κ_1^p	τ_0	τ_1
90 nm	29.313	1.261	13.274	1.312	1.099
65 nm	26.561	1.238	27.082	0.657	0.866

While previously-proposed closed-form optimal buffering solutions are efficient to compute, they are difficult to adapt to more complex and accurate delay models. Furthermore, hybrid objective functions that optimize delay, power and area are even more difficult to handle. With this in mind, we have developed an iterative optimization technique that evaluates a given objective function for a given number and size of repeaters, while searching for the optimal (number, size) values. We have found that realistic objective functions are convex, making binary search for the optimal repeater size feasible.

Our iterative optimization is easily extensible to other interconnect optimizations such as wire sizing and wire spacing, but the runtime grows exponentially with the number of optimization knobs. In general, wire sizing and wire spacing are weaker optimization knobs and their effect at the system-level can be ignored. We optimize only the number and size of repeaters during interconnect optimization. However, we support the use of double-width and double-spacing design styles which the system designer can invoke to optimize interconnect area, delay, noise and power.

Figure 3.7 shows the Pareto-optimal delay-power tradeoff for a 5 mm global buffered interconnect in 90 nm and 65 nm technologies. We note that for both technologies, power can be reduced by 20% at the cost of less than 2% degradation in delay.

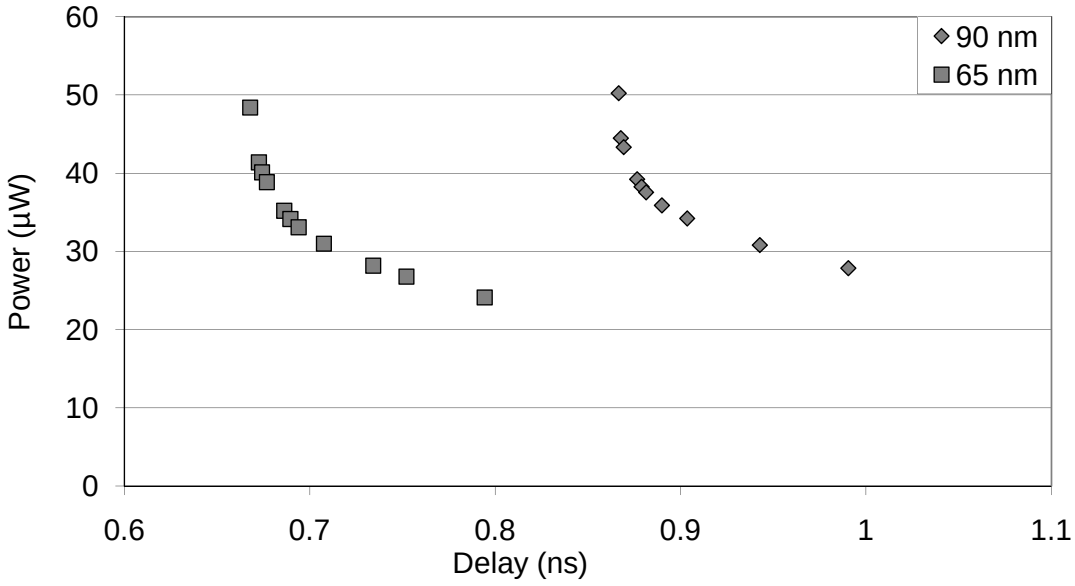

Figure 3.7: Pareto-optimal frontier of the delay-power tradeoff in 90 nm and 65 nm technologies.

3.3.7 Publicly-Available Framework

Finally, we have developed a framework capable of modeling and optimizing buffered interconnects for various technologies and under different design styles [12]. The framework is accessible through XML files or through a C++ application programming interface (API). We have packaged the framework with XML files for both future and existing technologies corresponding to commercial foundry processes (90 nm, 65 nm and 45 nm), ITRS and PTM.

3.3.8 Model Evaluation and Discussion

To assess the accuracy of our model with respect to previously-proposed models ([24] and [87]) we consider buffered interconnects of lengths 1 mm, 3 mm, 5 mm and 10 mm for three technology choices (90 nm, 65 nm and 45 nm), two design styles (single-width-single-spacing (SW-SS) and shielding), and global wiring regime against physical

implementation.[3]

To create the layout of a buffered interconnect, we first define the placement area in *Cadence SOC Encounter v6.1*. Repeaters are then placed at equal distances along the wire length to buffer the interconnect uniformly. Connections between inputs, outputs and the buffers are created by *Cadence NanoRoute*. The values of minimum wire spacing and wire width are chosen from the input LEF file. Parasitic extraction on the buffered interconnects is performed using *SOC Encounter*'s built-in extractor. To perform timing analysis, we read in the parasitics output from *SOC Encounter* in standard parasitic exchange format (SPEF) and the timing library (Liberty format) into *PrimeTime v2006.12* for signoff delay calculation. The results of our accuracy studies are presented in Table 3.2 as a function of the interconnect length l_{int} and the design style DS. The columns denoted as $Bakoglu$, $Pamunuwa$ and $Prop.$ report the errors in delay prediction using Bakoglu's model [24], Pamunuwa's model [87] and our proposed model with respect to the delay of the buffered interconnect evaluated using *PrimeTime* (input transition time = 300 ps), which is reported in column PT. We observe that predictions using our proposed method match values from *PrimeTime* to within 12%. In comparison, previous models have error in the range of -7% to +97%.

Finally, the column denoted as RT reports the ratio of the CPU runtime of our proposed model versus *PrimeTime* (runtimes of the Bakoglu and Pamunuwa models are similar to ours since they are also simple analytical models). To perform runtime comparison, we use the following approach. For *PrimeTime*, we measure the time from when the tool starts to calculate the interconnect delay (i.e., when "report_timing" is called) until it returns the delay value.[4] For our model, we measure the computation time, i.e., from when inputs are available until the delay estimate is returned. Our models are implemented in C++. We report the average runtime values over 50 trials. Our proposed model is computationally at least 2.1× faster than *PrimeTime* when both are run on a 2.4 GHz Intel Xeon workstation. More importantly, our models avoid the significant setup time, license management, etc. required for *PrimeTime*. In summary,

[3]Since delay changes linearly with respect to length for buffered interconnects (Table 3.2), 1 mm, 3 mm, 5 mm and 10 mm are representative of other lengths that require buffering.

[4]To run *PrimeTime* we require several components including netlist, SPEF, and Liberty files, all of which require significant amount of time to generate. We consider these as one-time overheads and do not include them in our runtime analysis.

our new models achieve significant accuracy improvement and runtime improvement when compared to previous models and *PrimeTime*, respectively.

We also verify the accuracy of our leakage power and repeater area models. With respect to the cell leakage power values reported in the Liberty files for 90 nm, 65 nm and 45 nm technologies, the maximum error of our predictive model is less than 11%.[5] With respect to the cell area values of the corresponding cells in the Liberty files, the maximum error of our predictive model is less than 8%.

To assess the impact of improved accuracy on system-level design space exploration, we integrate our models in COSI-OCC [92], a system-level tool for synthesis of NoCs. We use two representative SoC designs as testcases. The first design (VPROC) is a video processor with 42 cores and 128-bit datawidths. The second design is based on a dual video object plane decoder (dVOPD), where two video streams are decoded in parallel using 26 cores and 128-bit datawidths. Table 3.3 compares the interconnect power, delay and area when COSI-OCC's original model and the proposed model are used. The original model uses the Bakoglu delay model and does not consider any of the improvements that we have discussed. It also obtains its technology inputs from PTM models which are not calibrated to industry library files. The clock frequencies used are 1.5 GHz, 2.25 GHz, and 3.0 GHz for 90 nm, 65 nm, and 45 nm technology nodes, respectively. Hop count, which captures the communication latency, is also reported.

The main differences between the NoC architectures obtained using the original and the proposed models are in the power and hop-count figures across all technology processes. The dynamic power consumption estimated by the proposed model is up to three times larger than the dynamic power consumption estimated by the original model for the 90 nm and 65 nm technology nodes. The difference is due to the coupling capacitance that is neglected by the original model, and the different size and number of repeaters used by the two models. For the proposed model, we observe an increase in dynamic power going from 65 nm to 45 nm. This is due to the supply voltage increase in the library files from 1 V to 1.1 V, respectively. This difference also widens the gap in dynamic power between the original and the proposed model. The leakage power is also different, mainly as a consequence of the number and size of the repeaters that

[5]The repeater sizes used in our experiments include INVD4, INVD6, INVD8, INVD12, INVD16, and INVD20.

Table 3.2: Evaluation of model accuracy.

Tech.	l_{int} (mm)	DS	PT (ns)	$Bakoglu$ (%)	$Pamunuwa$ (%)	$Prop.$ (%)	RT (X)
90 nm	1	SW-SS	0.144	89.9	26.3	-11.2	2.2
		shielding	0.108	84.2	22.3	-8.6	2.3
	3	SW-SS	0.411	91.1	36.2	-2.3	2.1
		shielding	0.398	89.6	31.2	-1.8	2.2
	5	SW-SS	0.670	97.0	66.4	-8.2	2.3
		shielding	0.659	92.4	65.2	-6.7	2.3
	10	SW-SS	1.394	85.6	52.3	-10.4	2.3
		shielding	1.344	79.5	47.6	-7.1	2.3
65 nm	1	SW-SS	0.116	6.1	53.2	-4.3	2.2
		shielding	0.107	5.1	50.9	-3.1	2.2
	3	SW-SS	0.318	-2.3	45.3	-3.5	2.2
		shielding	0.302	-3.4	41.3	-2.9	2.1
	5	SW-SS	0.505	-6.9	33.7	-5.0	2.2
		shielding	0.489	-4.5	31.9	-3.9	2.3
	10	SW-SS	1.061	-3.1	39.6	-4.9	2.1
		shielding	1.012	-4.5	29.8	-2.9	2.3
45 nm	1	SW-SS	0.107	16.3	33.8	6.3	2.1
		shielding	0.098	11.2	31.2	6.2	2.1
	3	SW-SS	0.301	17.4	26.6	8.5	2.2
		shielding	0.291	14.2	26.1	7.9	2.1
	5	SW-SS	0.485	23.4	29.3	9.7	2.2
		shielding	0.474	24.2	26.7	7.8	2.2
	10	SW-SS	0.990	21.2	32.6	9.9	2.2
		shielding	0.962	24.4	23.8	9.1	2.3

are optimistically estimated by the original model. Also, the original model turns out to be very optimistic in allowing the use of excessively long wires: this is an example of a non-conservative abstraction that leads to design solutions that are actually not implementable. Finally, we note that the difference in area estimates between the original and proposed models is very large because of simplistic assumptions regarding area occupancy in the original model.

Table 3.3: Model impact on NoC synthesis.

SoC		p_{dyn} (mW)		p_{leak} (mW)		A_{rpt} (mm^2)	
		Orig.	Prop.	Orig.	Prop.	Orig.	Prop.
VPROC	90 nm	117.3	364.8	38.1	99.6	0.070	0.009
	65 nm	51.1	179.9	69.9	86.7	0.036	0.007
	45 nm	18	231	49	291	0.02	0.003
dVOPD	90 nm	63.4	88.0	14.2	32.5	0.026	0.003
	65 nm	27.3	73.2	25.7	33.2	0.013	0.003
	45 nm	9.6	98	18.1	142	0.007	0.002
SoC		A_{total} (mm^2)		Ave. #hops		Max. #hops	
		Orig.	Prop.	Orig.	Prop.	Orig.	Prop.
VPROC	90 nm	0.370	0.346	3.09	3.01	4	5
	65 nm	0.217	0.223	3.10	3.42	4	6
	45 nm	0.138	0.137	3.1	3.2	4	6
dVOPD	90 nm	0.141	0.162	1.76	1.76	3	3
	65 nm	0.082	0.085	1.76	1.91	3	4
	45 nm	0.053	0.029	1.76	2.12	3	5

3.4 Worst-case Interconnect Performance Prediction

The power distribution network (PDN) is a major consumer of interconnect resources in deep-submicron designs (e.g., more than 30% of the entire routing area [117]). Hence, efficient early-stage PDN optimization enables the designer to ensure

a desired power-performance envelope. On the other hand, as technology scales, gate delays become more sensitive to power supply variation. In addition, emerging 3D designs are more prone to supply voltage and temperature variations due to increased power density. In this section, we develop accurate inverter cell delay and output slew models under supply voltage and temperature variations.

In sub-65 nm designs, power and ground voltage-level fluctuations have become a primary concern for power integrity because circuit timing has become more susceptible to supply voltage noise. Thus, designers must take into consideration the impact of supply voltage noise to ensure successful chip design [84]. Existing works [46, 119, 120] on supply voltage noise and its implications on PDN optimization are oblivious to the timing impacts of the supply voltage noise. In this section, we develop closed-form performance models under supply voltage and temperature variations that aid designers to assess the impact of PDN design choices on the performance of the design.

Temperature variations affect transistor characteristics including threshold voltage, drive current, and off-current. Hence, it is important to accurately model the impact of temperature on circuit performance. Existing literature [26, 51] propose closed-form expressions that consider the impact of temperature on cell delay. However, in this section we consider the combined effects of both supply voltage and temperature variations on circuit performance.

In addition, emerging 3D designs are more prone to supply voltage noise due to increase in power and current demands and variations among tiers. Compensation of the supply voltage variation requires a fair amount of the silicon real estate (e.g., decoupling capacitance allocation), routing resources and increased packaging costs. Increased power density in 3D designs also requires close attention to the impact of temperature on circuit performance. Hence, to guarantee a given performance envelope, designers need to characterize the impact of supply voltage and temperature variations on circuit timing. Furthermore, there are a number of problems caused by dynamic effects of supply voltage noise. These effects include (1) change in maximum frequency of a critical path, and (2) degradation of the clock network performance. Thus, designers must consider the dynamic effect of supply voltage noise early in the design cycle.

Finally, the PDN is a major consumer of resources (e.g., more than 30% of the entire routing area) in wire-limited deep-submicron designs [117]. Conventionally, the PDN is designed to satisfy power integrity constraints, but without understanding the true implications of supply voltage noise on delay, correct optimization of the PDN is impossible. To close this gap, our present work gives a methodology to model the delay impact of supply voltage noise (characterized by noise slew, offset and magnitude). We believe our inverter cell delay and output slew models can efficiently drive accurate worst-case performance-driven PDN optimization, as shown in Figure 3.8.

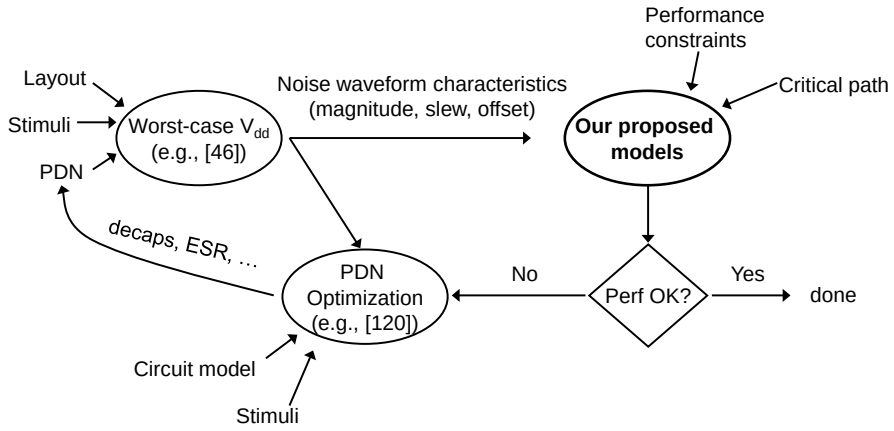

Figure 3.8: Accurate worst-case performance-driven power distribution network optimization flow.

In this section, we propose a new modeling paradigm in which we use machine learning-based nonparametric regression techniques to develop accurate early-stage performance models under dynamic supply voltage and temperature variations. Furthermore, we introduce a reproducible flow to aid automatic generation of accurate performance estimation models (e.g., using generic critical paths).

3.4.1 Implementation Flow

Figure 3.9 shows our implementation flow, which begins with SPICE simulations using foundry SPICE models and extracted or circuit description language (CDL) SPICE netlists for each gate type. We measure the 50% delay and output slew of each gate with respect to a number of different parameters. Our experiments have three main

axes: (1) cell delay parameters, (2) supply voltage noise parameters and (3) temperature. These parameters and the values that they take on in our experiments are explained below. Cell delay parameters include input slew ($t_{slew,in}$), output load (c_{load}) and cell size ($size_{cell}$). For supply voltage, we use 0.9 V as the nominal value with noise waveform superimposed on it.

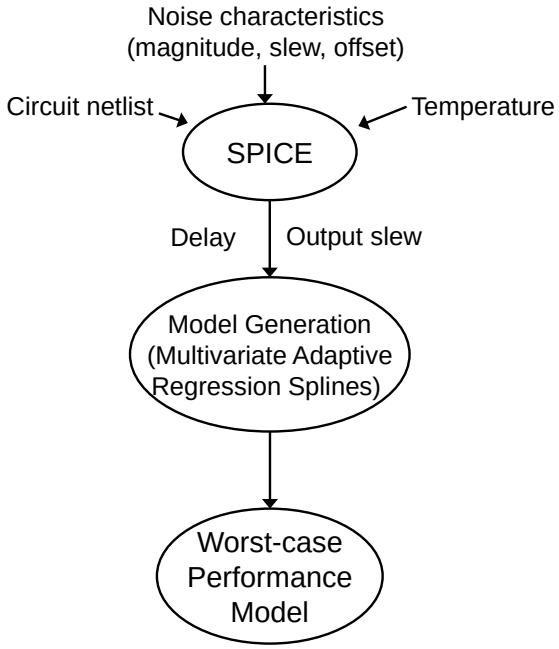

Figure 3.9: Implementation flow.

Supply voltage noise parameters include noise amplitude (v_{noise}), noise slew ($t_{slew,noise}$) and noise offset (off_{noise}). Noise offset denotes the noise transition time with respect to that of the input signal transition. Finally, temperature denotes the operating temperature of the transistors. In our studies, we use two different cells, an inverter (INV) and a 2-input NAND (ND2), to show the applicability of our modeling approach. For worst-case performance modeling, we implement our basic cell delay and output slew models in C++. Using our basic delay and output slew models, we construct path delay models with arbitrary numbers of stages and a mix of different cells. We run a total of 30,720 SPICE simulations and gather delay and output slew values corresponding to different parameter combinations (see Table 3.4).

Our SPICE simulations use *Synopsys HSPICE v.Y-2006.03* [15] and 65 nm foun-

dry SPICE models and netlists. We perform our experiment using typical corner and normal-V_{th} (NVT) transistors. We also use *MARS3.0* [9] to implement nonparametric regression techniques.

Table 3.4: List of parameters used in our studies.

Parameter	Values
$t_{slew,in}$	{0.00056, 0.00112, 0.0392, 0.1728,0.56, 0.7088}ns
c_{load}	{0.0009, 0.0049, 0.0208, 0.0842}pF
$size_{cell}$	INV: {1, 4, 8, 20} ND2: {1, 2, 4, 8}
v_{noise}	{0, 0.054, 0.144, 0.27}V
$t_{slew,noise}$	{0.01, 0.04, 0.07, 0.09}ns
off_{noise}	{-0.15, -0.05, 0, 0.05, 0.15}ns
$temp$	{-40, 25, 80, 125}°C

3.4.2 Modeling Methodology

Modeling Flow

Previous delay estimation techniques do not consider the dynamic impact of supply voltage noise on cell delay [53, 76, 96]. By contrast, we propose to pursue a different modeling paradigm in which we use machine learning-based nonparametric regression techniques to capture the dynamic impact of supply voltage noise on cell delay. To illustrate the basic idea, consider the following baseline model-generation flow.

- We begin with a parameterized SPICE netlist for a given inverter cell. We refer to this as a *configurable* inverter SPICE specification, which will be used to generate the representative inverter cell delay under different cell and supply voltage noise parameters. For example, a given SPICE simulation setup can be configured with respect to (1) input slew, (2) output load, (3) inverter size, (4) supply voltage noise

magnitude, (5) noise slew, (6) voltage noise offset (i.e., with respect to the input transition), and (7) temperature.

- Using a small subset of selected configurations for *training*, we run each configuration in the training set through SPICE simulations to obtain accurate cell delay and output slew values.

- Finally, we apply machine learning-based nonparametric regression to derive the cell delay estimation models from the training set of delay values.

In general, the modeling problem aims to approximate a function of from several to many variables using only the dependent variable space. This generic formulation has applications in many disciplines. The goal is to model the dependence of a target variable y on several predictor variables $x_1, \cdots, x_n$ given N realizations $\{y_i, x_{1i}, \cdots, x_{ni}\}_1^N$. The system that generates the data is presumed to be described by

$$y = f(x_1, \cdots, x_n) + \epsilon \tag{3.22}$$

over some domain $(x_1, \cdots, x_n) \in \mathcal{D} \subset \mathcal{R}^n$ containing the data [52]. Function f captures the joint predictive relationship of y on $x_1, \cdots, x_n$, and the additive stochastic noise component ϵ usually reflects the dependence of y on quantities other than $x_1, \cdots, x_n$ that are neither controlled nor observed. Hence, the aim of the regression analysis is to construct a function $\hat{f}(x_1, \cdots, x_n)$ that accurately approximates $f(x_1, \cdots, x_n)$ over the domain $\mathcal{D}$ of interest.

There are two main regression analysis methods: (1) parametric and (2) nonparametric. The former approach has limited flexibility and produces accurate approximations only if the assumed underlying function $\hat{f}$ is close to f. In the latter approach, $\hat{f}$ does not take a predetermined form, but is constructed according to information derived from the data. *Multivariate adaptive regression splines* (MARS) [9] is a nonparametric regression technique that is an extension of piecewise-linear models, and that automatically captures nonlinearities and parameter interactions. In this section, we use the MARS-based approach to model the dynamic impact of supply voltage noise on cell delay.

Multivariate Adaptive Regression Splines

Given different cell and supply voltage noise parameters $\mathcal{X}$, we apply MARS to construct the cell delay model, $t_{cell} = \hat{f}(x_1, \cdots, x_n)$. Variables $x_1, \cdots, x_n$ denote cell and supply voltage noise parameters as well as temperature. The general MARS model can be represented as [121]

$$\hat{y} = e_0 + \sum_{i=1}^{I} e_i \prod_{j=1}^{J} b_{ij}(x_{ij}) \tag{3.23}$$

where $\hat{y}$ is the target variable (i.e., inverter delay and output slew in our problem), e_0 is a constant, e_i is the fitting coefficient (where $1 \leq i \leq I$), and $b_{ij}(x_{ij})$ is the truncated power basis function[6] with x_{ij} being the cell and supply voltage noise and temperature parameters used in the i^{th} term of the j^{th} product. I is the number of basis functions and J limits the order of interactions. In our experiments, we set the number of basis functions to 100 and the order of interactions to six, i.e., every parameter can interact with all the other parameters. The basis function $b_{ij}(x_{ij})$ is defined as

$$b_{ij}^{-}(x - v_{ij}) = [-(x - v_{ij})]_{+}^{q} \tag{3.24}$$

$$= \begin{cases} (v_{ij} - x)^q & x < v_{ij} \\ 0 & \text{otherwise} \end{cases}$$

$$b_{ij}^{-}(x - v_{ij}) = [+(x - v_{ij})]_{+}^{q} \tag{3.25}$$

$$= \begin{cases} (x - v_{ij})^q & x > v_{ij} \\ 0 & \text{otherwise} \end{cases}$$

where q (≥ 0) is the power to which the splines are raised to adjust the degree of $\hat{y}$ smoothness, and v_{ij} is called a *knot*. When q = 0 simple linear splines are applied.

The optimal MARS model is built in two passes. (1) Forward pass: MARS starts with just an intercept, and then repeatedly adds basis function in pairs to the model. The total number of basis functions is a user-specified input to the modeling. (2) Backward

[6] Each basis function can be a constant, a hinge function that is of form $max(0, e-x)$ or $max(0, x-e)$, or a product of two or more hinge functions.

pass: during the forward pass, MARS usually builds an overfit model; hence, to build a model with better generalization ability, the backward pass prunes the basis functions using a *generalized cross-validation* (GCV) scheme.

$$GCV(K) = \frac{1}{N}\frac{\sum_{k=1}^{N}(y_k - \hat{y})^2}{[1 - \frac{C(M)}{N}]^2} \tag{3.26}$$

where N is the number of observations in the data set, K is the number of non-constant terms, and $C(M)$ is a complexity penalty function to avoid overfitting.

3.4.3 Accurate Cell Delay Modeling

In this subsection, we discuss the impact of supply voltage noise and temperature variations on cell delay. Note that delay modeling under supply voltage and temperature variations is a nontrivial task. We show an example of our proposed delay and output slew models derived using machine learning-based nonparametric regression techniques. We also propose a methodology to find the worst-case input configuration that maximizes the delay of a given path.

Cell Delay and Output Slew Models

In the existing literature [53, 76], supply voltage variation is assumed to be constant (time-invariant). When the supply voltage varies slowly with respect to the clock period, this is reasonable. This assumption enables us to predict the timing impact of the supply voltage noise: the worst-case delay corresponds to the worst-case noise that can occur when the target cell is switching. When the supply voltage varies slowly, the delay degradation is proportional to the peak of the noise [96]. To better capture the impact of time-varying supply voltage noise we must consider the noise waveform characteristics including (1) noise magnitude, (2) noise slew and (3) noise offset. Figure 3.10 shows the impact of noise slew on inverter delay. We observe that noise slew affects cell delay only when it is comparable to input slew. Hence, we must take into consideration the specific noise waveform characteristics to ensure more accurate delay modeling.

Existing PDN optimization frameworks [119, 120] use *fluctuation area*, i.e., the area under the noise waveform, as the metric to represent the supply voltage noise. It

is easy to see that such an approach can incur significant error in the delay estimation. Consider two scenarios: (1) $t_{slew,noise}$ = 0.2 ns, v_{noise} = 0.2 V and (2) $t_{slew,noise}$ = 0.4 ns, v_{noise} = 0.1 V. Using a triangular waveform to represent the supply noise, the two scenarios have different noise waveforms, yet have similar areas under the noise curve. When we evaluate gate delay under each of these scenarios, we observe 22% difference. (In this evaluation, we use a single inverter, with other parameters values being $t_{slew,in}$ = 0.4 ns, c_{load} = 0.002 pF, $size_{cell}$ = 1X, off_{noise} = 0 ns, and $temp$ = 25°C.) We conclude that to accurately model the impact of supply voltage noise on cell delay, we must consider both noise slew and noise magnitude parameters, and not simply the area under the noise waveform.

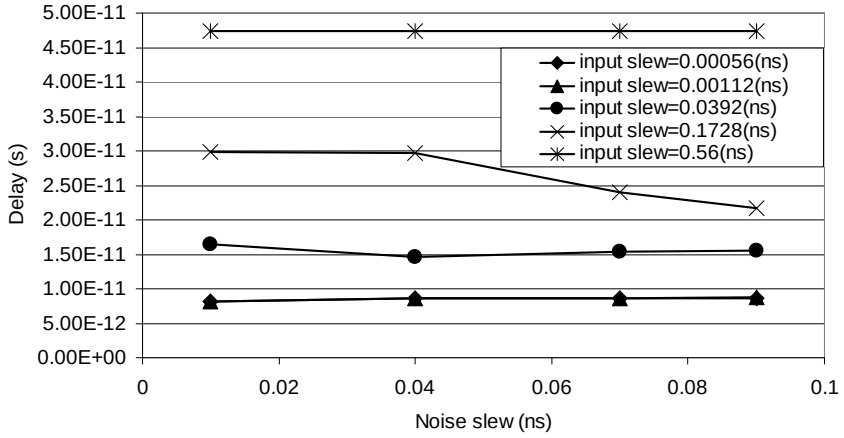

Figure 3.10: Delay of an inverter versus noise slew for different input slew values.

The other important supply voltage noise characteristic is *noise offset*, which denotes the time of the voltage noise transition relative to the time of the input signal transition. We expect that as long as the supply voltage noise waveform is outside of the input signal transition window, it should not have any impact on cell delay. However, when the noise waveform overlaps with the input signal transition, it affects the cell delay. Figure 3.11 shows the impact of noise offset on cell delay. In our experiment, input slew and noise slew are 0.09 ns and 0.1 ns, respectively. In our delay model, we explicitly consider noise offset as an input to the model.

In addition, cell characteristics are influenced by temperature. Temperature impacts cell delay through voltage threshold, mobility, and other parameters [51]. For example, as temperature decreases, both threshold voltage and mobility increase; the

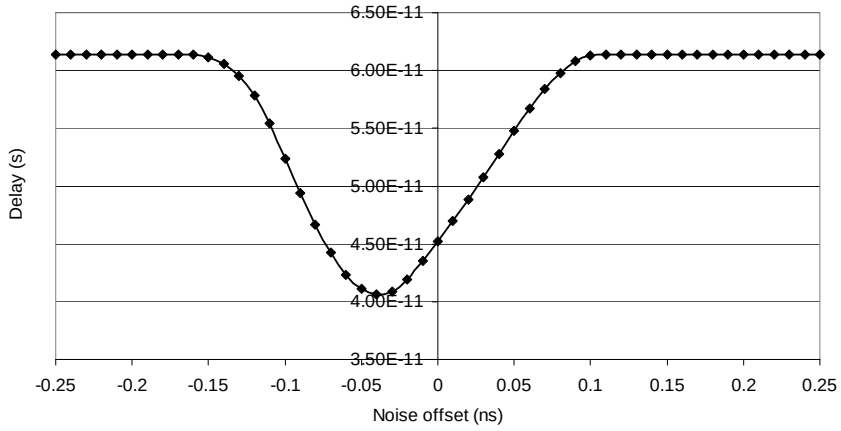

Figure 3.11: Impact of supply voltage noise offset on cell delay.

latter causes increased saturation current. However, the impact of temperature on cell delay depends on the gate voltage. The gate voltage at which the temperature shifts of threshold voltage and mobility exactly compensate each other's effects on delay is typically called the *zero-temperature-coefficient* [72]. Hence, cell delay can increase or decrease with the increase in temperature. These complex relationships between cell delay and the aforementioned parameters make delay modeling a nontrivial task.

Finally, since our gate delay model depends on the input slew we must also model output slew of the previous stage of the critical path. The above discussion indicates that approximating CMOS gate delay is a nontrivial task with non-obvious implications, as seen from Figures 3.10 and 3.11. This motivates our exploration of machine learning-based nonparametric regression techniques to develop accurate cell delay and output slew models. Figure 3.12 illustrates the form of the resulting inverter delay and output slew models using 65 nm foundry SPICE models.[7]

Worst-case Performance Model

We formalize the problem of finding the worst-case performance under dynamic supply voltage and temperature variations. We are interested in the specific configuration, i.e., the set of seven parameters (7-tuple) described in Table 3.4, that causes the

[7]Note that our methodology can be straightforwardly applied to future technologies, as long as necessary SPICE models and device-level netlists are available.

Delay Model
b_1 = max(0 , c_{load} – 0.0208);
b_2 = max(0 , 0.0208 – c_{load}); $\cdots$
b_{98} = max(0 , off_{noise} – 0.05)$\times b_{92}$;
b_{100} = max(0 , off_{noise} + 2.4e^{-12})$\times b_{37}$;
t_{cell} = 1.018e-11 + 7.353e-10$\times b_1$ – 5.890e^{-10} $\times$ b_2
– 2.172e^{-11} $\times$ b_3 + $\cdots$ – 1.708e^{-7} $\times$ b_{96} +
2.431e^{-7} $\times$ b_{98} – 3.031e^{-8} $\times$ b_{100}
Output Slew Model
b_1 = max(0 , c_{load} – 0.0009);
b_2 = max(0 , $size_{cell}$ – 4)$\times b_1$; $\cdots$
b_{99} = max(0 , 0.05 – $t_{slew,noise}$)$\times b_{55}$;
b_{100} = max(0 , off_{noise} + 0.15)$\times b_{94}$;
$t_{slew,out}$ = 1.227e^{-11} + 1.529 $\times b_1$ – 2.051e^{-10} $\times$ b_2
+ 2.050e^{-9} $\times$ b_3 + $\cdots$ – 1.081e^{-8} $\times$ b_{98}
– 4.327e^{-9} $\times$ b_{99} – 7.422e^{-9} $\times$ b_{100}

Figure 3.12: Sample inverter delay and output slew models in 65 nm.

delay of a given path with arbitrary number of stages to be maximum.[8] Note that we construct our path delay model using our basic cell delay and output slew models. Our proposed delay and output slew models are essentially mappings of f and g, respectively, from the set of all 7-tuples Q (cf. Table 3.4) to the positive reals, i.e., $f : Q \rightarrow \mathcal{R}^+$ and $g : Q \rightarrow \mathcal{R}^+$, where $Q = t_{slew,in} \times\ c_{load} \times size_{cell} \times v_{noise} \times\ t_{slew,noise} \times off_{noise} \times temp$.

For a single stage the problem of finding the worst-case configuration seeks $\vec{q}^* \in Q$ such that $f(\vec{q}^*)$ is maximized. With more than one stage in a path, i.e., $k\ >\ 1$, the output slew of the previous stage becomes the input slew to the current stage. In addition, the noise offset must be adjusted accordingly. Then, we seek $\vec{q_1}^*$ such that $f(\vec{q_1}^*) + \cdots + f(\vec{q_k}^*)$ is maximized, where $\vec{q_m}^* = \vec{q_1}^*$ for all stages $1 < m < k$, except that the

[8] In our experiments, a path consists of (1) only inverters, (2) only 2-input NAND gates or (3) a mix of inverters and 2-input NAND gates.

$t_{slew,in}$ component is replaced by $g(\vec{q}^{*}_{m-1})$, and the off_{noise} component is adjusted at the beginning of each stage. Note that the worst-case configuration is always going to be an element of the cross product of the various sets of parameter values. In other words, it is one of $|t_{slew,in}| \times |c_{load}| \times |size_{cell}| \times |v_{noise}| \times |t_{slew,noise}| \times |t_{slew,noise}| \times |off_{noise}| \times |temp|$ configurations. In our studies, the worst-case configuration is out of 30,720 different configurations.

3.4.4 Model Evaluation and Discussion

To generate our models, we randomly select 10% of our entire data set as training data, and we test the models on the other 90% of the data. To show that the selection of the training set does not substantially affect model accuracy, we randomly select 10% of the entire data set five times and show the corresponding models' maximum and average error values (Table 3.5).

Table 3.5: Model stability versus random selection of the training set.

Experiments	delay % error		output slew % error	
	max	avg	max	avg
Exp 1	56.993	5.660	55.117	6.012
Exp 2	53.342	5.458	56.896	5.976
Exp 3	53.661	5.401	56.237	5.526
Exp 4	55.419	5.552	54.883	5.311
Exp 5	55.015	5.609	55.614	5.672

To show the accuracy of our worst-case performance model, we compare our worst-case predictions with SPICE simulations. We construct three different paths with different number of stages, each consisting of (1) only inverters, (2) only 2-input NAND gates, or (3) a mix of inverter and 2-input NAND gates. For path (3), we construct the path starting with an inverter, and then alternate 2-input NAND gates with inverter gates. In our experiments, both of the NAND gate inputs are connected to each other. We evaluate our predictions using two metrics: (1) correlation of our predictions against SPICE results and (2) relative (%) difference in delays between our proposed model

and SPICE. For (1), we rank our model predictions (total of 30,720 data points) in descending order with respect to the delay of the given path. Each delay value corresponds to a set of parameters (i.e., a 7-tuple including all the parameters shown in Table 3.4). Next, we compare our predicted worst-case configuration with SPICE and find the rank ($rank_{SPICE}$) of our predicted worst-case configuration within the SPICE results. For multi-stage paths with $k > 1$ stages, we need to adjust the noise offset for each stage. To perform this, we need to identify the time at which the input to Stage i (where $i = 1, \cdots, k$) makes the transition. This value can be estimated by calculating the delay up to Stage $(i-1)$, and subtracting $\frac{t^i_{slew,in}}{1.6}$ from it, where $t^i_{slew,in}$ is the input slew to Stage i, and $\frac{t^i_{slew,in}}{1.6}$ determines the 50% output slew transition.[9]

Tables 3.6, 3.7, and 3.8 show the comparison of our worst-case performance model with SPICE for a path consisting of (1) only inverter gates, (2) only 2-input NAND gates, and (3) a mix of inverter and 2-input NAND gates, respectively. The second and third columns represent our (2) and (1) comparison metrics, respectively. The fourth column shows where the SPICE worst-case configuration is ranked according to our proposed model ($rank_{MARS}$). We observe that our path delay model is within 4.3% of SPICE simulations. In addition, our predictions are always ranked in the top3 (out of 30,720 configurations) of the SPICE list ($rank_{SPICE}$). We note that the ability of our worst-case performance model to correctly predict worst-case configuration is beneficial for early-stage design and optimization of power distribution networks. Finally, the SPICE-computed worst-case performance value is always among the top5 predictions of our model.

3.4.5 Extensibility to Other Metrics

In this subsection, we use machine learning-based nonparametric regression to model interconnect wirelength and fanout. Existing analytical interconnect wirelength and fanout models [39, 118] are not accurate due to not having accurate information about the netlist, and not taking into account the combined impacts of microarchitectural and implementation parameters. Final design outcomes are affected by optimization steps that are employed during design implementation, e.g., pre-placement, post-

[9] In our experiments, the 10%-90% transition time is taken as the slew value.

Table 3.6: Comparison of our worst-case performance model and SPICE for an inverter chain. Rank values are out of 30,720 configurations.

#Stage	delay % error	$rank_{SPICE}$	$rank_{MARS}$
1	1.08	1	1
3	3.54	3	2
5	4.29	1	1
10	3.26	2	4
20	2.42	1	1
30	2.88	1	1

Table 3.7: Comparison of our worst-case performance model and SPICE for a 2-input NAND chain. Rank values are out of 30,720 configurations.

#Stage	delay % error	$rank_{SPICE}$	$rank_{MARS}$
1	1.34	1	1
3	3.21	1	1
5	3.69	2	3
10	3.11	1	1
20	3.43	2	3
30	2.37	2	2

placement and pre-clock tree synthesis optimization steps. Hence, the choice of implementation parameters can significantly change the quality of results.

We use a similar implementation flow as shown in Figure 3.9 with two testcases: (1) an on-chip router and (2) a discrete Fourier transform (DFT) core. Using implementation data and the nonparametric regression technique explained in the previous subsections, we obtain architecture-level interconnect wirelength and fanout models. Figures 3.13 and 3.14 show our proposed average wirelength and fanout models for DFT core, respectively.

To validate our wirelength model, we compare four different models.

Table 3.8: Comparison of our worst-case performance model and SPICE for a mixed inverter-NAND chain. Rank values are out of 30,720 configurations.

#Stage	delay % error	$rank_{SPICE}$	$rank_{MARS}$
1	1.08	1	1
3	2.73	2	4
5	3.24	3	5
10	3.36	1	1
20	3.93	2	4
30	2.85	1	1

Basis Functions
b_1 = max(0, m_{DFT} – 16); b_2 = max(0, 16 – m_{DFT});
b_4 = max(0, 16 – dw) $\times b_1$; b_5 = max(0, $util$ – 0.5);
b_6 = max(0, n_{fifo} – 2); b_7 = max(0, dw – 16);
⋮
b_{31} = max(0, AR – 1.5) $\times b_7$; b_{35} = max(0, tw – 2) $\times b_{31}$;
Average Wirelength Model
WL_{avg} = 22.4886 + 0.056 $\times b_1$ – 0.328 $\times b_2$ + 0.013 $\times b_4$
– 5.891 $\times b_5$ – 0.226 $\times b_6$ – 0.194 $\times b_7$ – 0.271 $\times b_8$
– 0.018 $\times b_9$ + 0.001 $\times b_{11}$ + 0.017 $\times b_{12}$ + 0.0002 $\times b_{13}$
+ 0.001 $\times b_{15}$ + 0.002 $\times b_{16}$ – 9.104e^{-6} $\times$ b_{17} – 2.176e-5 $\times b_{18}$
– 0.051 $\times b_{19}$ – 0.017 $\times b_{21}$ – 2.228e^{-5} $\times$ b_{24} + 0.0003 $\times b_{25}$
+ 0.003 $\times b_{27}$ – 0.01284 $\times b_{35}$

Figure 3.13: Average wirelength model for DFT core in 65 nm.

Basis Functions
b_1 = max(0, m_{DFT} – 16); b_2 = max(0, 16 – m_{DFT});
b_3 = max(0, dw – 8); b_4 = max(0, n_{fifo} – 2);
b_5 = max(0, m_{DFT} – 16) $\times b_4$; b_6 = max(0, 16 – m_{DFT}) $\times b_4$;
$\vdots$
b_{30} = max(0, dw – 16) $\times b_9$; b_{33} = max(0, 16 – m_{DFT}) $\times b_{18}$;
Average Fanout Model
FO_{avg} = 3.707 + 0.003 $\times b_1$ – 0.034 $\times b_2$ – 0.011 $\times b_3$
– 0.016 $\times b_4$ + 8.602e^{-5} $\times$ b_5 + 0.002 $\times b_6$ + 7.051e^{-5} $\times$ b_7
+ 0.002 $\times b_8$ – 9.943e^{-5} $\times$ b_9 – 0.002 $\times b_{10}$ – 0.084 $\times b_{13}$
+ 7.989e^{-6} $\times$ b_{16} – 4.533e^{-6} $\times$ b_{17} + 0.0002 $\times b_{18}$
+ 1.011e^{-5} $\times$ b_{21} + 0.0005 $\times b_{22}$ + 0.006 $\times b_{23}$ + 0.003 $\times b_{25}$
– 0.0003 $\times b_{27}$ – 1.478e^{-5} $\times$ b_{28} – 1.492e^{-5} $\times$ b_{29}
– 8.567e^{-6} $\times$ b_{30} – 1.225e^{-5} $\times$ b_{33}

Figure 3.14: Average fanout model for DFT core in 65 nm.

- **Prop.:** Our proposed model.
- **Model 1:** Christie *et al.* [39] model with N, p and k modeled as a function of microarchitectural and implementation parameters, where N is the number of gates in a block, and p and k are empirical parameters.[10]
- **Model 2:** Modified Christie model with a correction factor.
- **Model 3:** Christie model with N, p and k derived from layout data for each configuration.

To validate our interconnect fanout model, we compare the same four models except that our reference model is Zarkesh-Ha *et al.* [118].

Our proposed model (Prop.) uses MARS to estimate interconnect wirelength and fanout as a function of microarchitectural and implementation parameters. For Model

[10] Rent's rule is a simple power-law relationship between the number of I/O terminals for a logic block, T, and the number of gates contained in that block, N [39]: $T = kN^p$.

1, we use the same regression approach to model N, p and k as a function of microarchitectural and implementation parameters. Then, we use the estimated *Rent* parameters in the Christie (Zarkesh-Ha) model to obtain wirelength (fanout) values. In the Christie model, unit distance between adjacent placement sites, i.e., $l = 1$, is modeled as $\sqrt{X_{die}Y_{die}/N}$, where X_{die} and Y_{die} are the width and height of the floorplan of each design, respectively. In Model 2, we first model N, p and k as a function of microarchitectural parameters only (i.e., with similar modeling approach as in Prop.). We then apply the estimated Rent parameters in the Christie (Zarkesh-Ha) model and introduce a correction factor α such that *Actual Wirelength* = $\alpha \times Model_{Christie}$ (*Actual Fanout* = $\alpha \times Model_{Zarkesh-Ha}$). We model α as a function of implementation parameters only, using the same modeling approach as in our proposed models. In the modified Christie's model (Model 2), we do not include the unit distance model as used in Model 1 because unit distance depends on implementation parameters. Finally, Model 3 uses the Christie (Zarkesh-Ha) model to estimate wirelength (fanout) using extracted N, p and k values from implemented designs.

To extract Rent parameters for Models 1, 2 and 3, we use layout reports to obtain N, and an internal Rent parameter evaluation program *RentCon*, to extract p and k values from a placed and routed design exchange format (DEF) [3]. To compute p and k, we use a circuit partitioning-based method that recursively applies min-cut bisection until the minimum number of cell instances over all partitions reaches two; each source-sink connection crossing the boundary is counted as one pin. For each level of the recursive bipartitioning, we compute the geometric mean values of the number of cell instances and the number of pins, which represent one data point in the fitted curve of the Rent parameter.[11]

To generate our models, we randomly select 10% of our entire data set (i.e., a total of 2187 data points for each testcase) and test the models on the other 90% of the data. To show that the selection of the training set does not substantially affect model accuracy we randomly select 10% of the entire data set five times and show the corresponding maximum and average error values (Table 3.9).

Tables 3.10 and 3.11 show comparisons of our proposed wirelength and fanout

[11]The multi-level circuit partitioner MLPart [33] is used to recursively partition the circuit netlist.

Table 3.9: Impact of random selection of the training set on model accuracy.

Experiments	average wirelength % error		average fanout % error	
	max	avg	max	avg
Exp 1	22.9	3.4	4.3	0.7
Exp 2	18.2	3.5	6.0	1.3
Exp 3	23.9	3.4	8.6	0.6
Exp 4	24.2	3.4	5.1	0.7
Exp 5	16.8	3.5	4.8	0.7

models with the above models, respectively. We observe significant accuracy improvement versus existing models (Model 3) with respect to layout data. Our estimated average wirelength values show an accuracy improvement of up to 14.7% (58.2%), and 24.9% (42%) in average (maximum) errors for DFT and router testcases, respectively. For average fanout, we observe accuracy improvement of up to 9.3% (17%), and 5.4% (16.8%) in average (maximum) errors for DFT and router testcases, respectively.

Table 3.10: Comparison of average wirelength derived from our proposed (Prop.), Model 1, Model 2 and Model 3 (Christie [39]) models with respect to actual implementation data.

Metric	DFT			
	Prop.	Model 1	Model 2	Model 3
maximum % error	**21.3**	76.4	98.2	79.5
average % error	**3.4**	17.9	22.7	18.1
Metric	Router			
	Prop.	Model 1	Model 2	Model 3
maximum % error	**17.9**	59.4	54.6	59.9
average % error	**2.3**	27.4	16.3	27.2

Finally, Figures 3.15, 3.16, 3.17 and 3.18 show scatter plots of our average wirelength and fanout estimations against corresponding Christie and Zarkesh-Ha models

Table 3.11: Comparison of average fanout derived from our proposed (Prop.), Model 1, Model 2 and Model 3 (Zarkesh-Ha [118]) models with respect to actual implementation data.

Metric	Discrete Fourier Transform (DFT)			
	Prop.	Model 1	Model 2	Model 3
maximum % error	**5.7**	23.7	18.7	22.7
average % error	**0.8**	10.1	7.3	10.1
Metric	Router			
	Prop.	Model 1	Model 2	Model 3
maximum % error	**1.4**	18.9	22.1	18.2
average % error	**0.2**	5.6	5.7	5.6

(i.e., Model 3) with respect to layout data.[12] These plots confirm the accuracy improvement of our proposed models versus existing models.

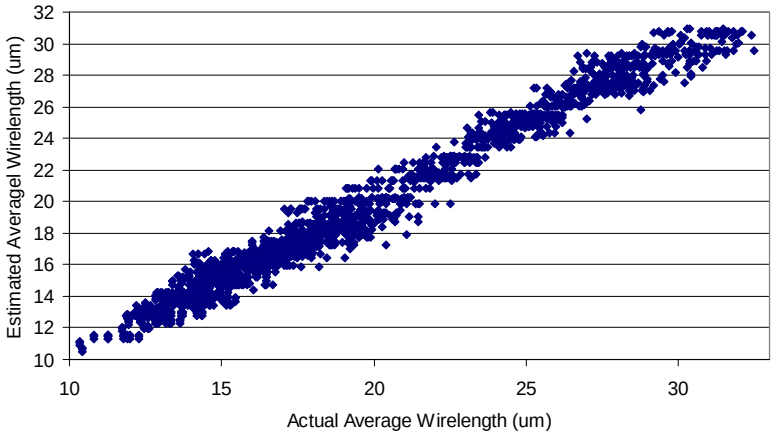

Figure 3.15: Our estimated average wirelength, plotted against layout data.

From Models 1 and 2, we understand that both microarchitectural and implementation parameters should be considered during model development. In addition, we can confirm that existing Rent's rule-based wirelength and fanout estimation models fail to correctly capture the impact of microarchitectural and implementation parameters, which can result in unrealistic estimates of wiring characteristics.

[12]These plots are for the DFT models. The router models show similar accuracy.

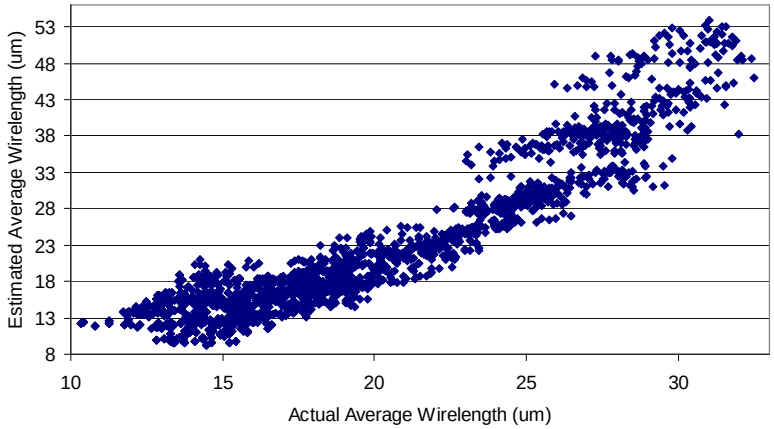

Figure 3.16: Christie's estimated average wirelength, plotted against layout data.

We also run our DFT core testcase through *Atrenta SpyGlass-Physical v4.2.1* [2], a fast physical simulator which provides early implementation feasibility analysis for digital blocks. We use the same design of experiments shown in the previous subsection. We collect corresponding power (dynamic and leakage), performance (maximum delay), and area (sum of all standard cells) of the DFT testcase reported by *SpyGlass*. Then, we use nonparametric regression to model power, performance and area with respect to architectural and implementation parameters. We observe that our models are within 3.5% of *SpyGlass* estimates. This experiment shows that the nonparametric regression techniques are robust with respect to different data sets coming from different sources.

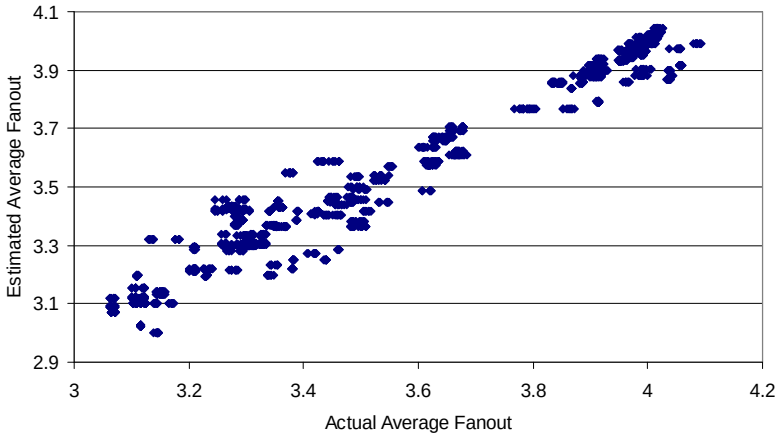

Figure 3.17: Our estimated average fanout, plotted against layout data.

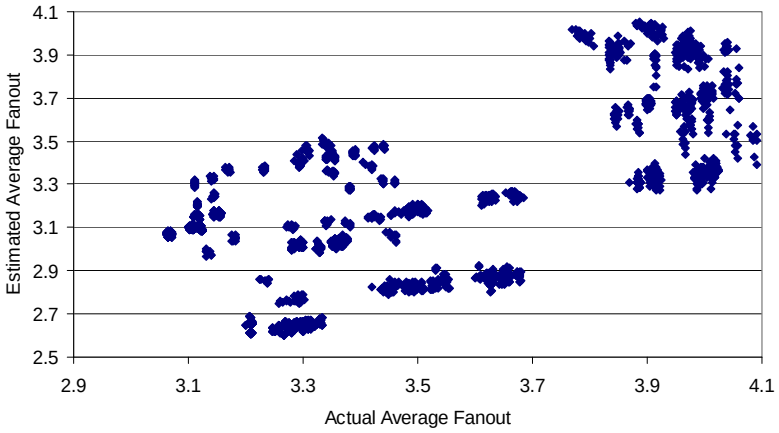

Figure 3.18: Zarkesh-Ha's estimated average fanout, plotted against layout data.

3.5 Conclusions

Accurate estimation of delay, power and area of interconnects early in the design phase can drive effective system-level exploration. Existing models of buffered interconnects are inaccurate for current and future technologies (due to deep-submicron effects) and can lead to misleading design targets. We propose accurate models for buffered interconnects that are easily usable by system-level designers. We present a reproducible methodology to extract inputs to our models from reliable sources. Existing delay-driven buffering techniques minimize interconnect delay without any consideration for power and area impact. This can often result in buffered interconnects that are infeasible during implementation, and motivates our proposed power-efficient buffering technique that minimizes total power with minimal delay impact.

To demonstrate the accuracy of our model, we evaluated its delay prediction for buffered interconnects in global and intermediate wiring layers across 90 nm, 65 nm and 45 nm technologies. Our results show that delay from our proposed model matches that from a commercial signoff tool within 11%. We integrate our model in an NoC topology synthesis tool (COSI-OCC) and show that accurate models substantially affect the explored topology solution.

In addition, we develop a methodology based on nonparametric regression to obtain accurate closed-form cell delay and output slew models under dynamic supply voltage and temperature variations. The proposed models are within 6%, on average,

of SPICE simulations. We show that our basic gate delay and output slew models can be used to construct delay estimates under supply noise for arbitrary critical paths. We also show that our models can accurately find the worst-case supply noise configuration that leads to worst-case delay performance. We believe that our proposed models enable accurate worst-case performance-driven power distribution network optimization as shown in Figure 3.8.

Finally, we show that the proposed nonparametric regression techniques are not restricted to modeling of power, performance and area. We use nonparametric regression to model interconnect wirelength and fanout with respect to architectural and implementation parameters, where our models are within 3.4% of layout data, on average. In an experiment with a third party physical simulator, we also show that nonparametric regression approaches are robust with respect to data sets from different sources.

3.6 Acknowledgments

This chapter is in part a reprint of:

- Chung-Kuan Cheng, Andrew B. Kahng, **Kambiz Samadi** and Amirali Shayan, "Worst-case Performance Prediction Under Supply Voltage and Temperature Variation", *Proc. ACM/IEEE International Workshop on System-Level Interconnect Prediction*, 2010, pp. 91–96.

- Luca Carloni, Andrew B. Kahng, Swamy Muddu, Alessandro Pinto, **Kambiz Samadi** and Puneet Sharma, "Accurate Predictive Interconnect Modeling for System-Level Design", *IEEE Transactions on Very Large Scale Integration Systems* 18(4) (2010), pp. 679–684.

- Kwangok Jeong, Andrew B. Kahng and **Kambiz Samadi**, "Architectural-Level Prediction of Interconnect Wirelength and Fanout", *Proc. IEEE International SOC Conference*, 2009, pp. 53–56.

- Luca Carloni, Andrew B. Kahng, Swamy Muddu, Alessandro Pinto, **Kambiz Samadi** and Puneet Sharma, "Interconnect Modeling for Improved System-Level

Design," *Proc. Asia and South Pacific Design Automation Conference*, 2008, pp. 258–264.

I would like to thank my coauthors Prof. Luca Carloni, Prof. Chung-Kuan Cheng, Kwangok Jeong, Dr. Swamy Muddu, Dr. Alessandro Pinto, Dr. Puneet Sharma, Amirali Shayan, and Prof. Andrew B. Kahng.

Chapter 4

On-Chip Router Power, Performance and Area Modeling

4.1 Introduction

Early-stage design exploration is essential to realize achievable power-delay-area tradeoffs. Existing methods for architecture-level estimation of on-chip routers can be broadly classified as *template-based*: in one way or another, they assume a specific architecture and underlying circuit implementation. The template-based approach is exemplified by the widely-used early-stage NoC power estimation tool ORION [114].

Other template-based approaches are based on parametric regression techniques [27, 30, 36, 77]. These approaches also assume a specific underlying router microarchitecture, and hence, require the development of new models for different microarchitectures. For parametric models, the modeler needs full comprehension of the underlying router microarchitecture in order to come up with a relevant model. Moreover, it is difficult to capture interactions between configuration parameters that may gain significance as the design complexity increases. Finally, most existing parametric modeling approaches fail to consider implementation flow options or settings in their models.

In this chapter, we introduce ORION 2.0, a set of accurate architecture-level on-chip router power and area models. To derive these models, we use a template-based approach. Template-based models are very beneficial because they can be developed

early in the design process and do not incur any implementation overhead. However, the template-based approach has two limitations. First, for power and area estimations to be accurate, the actual router microarchitecture used for implementation must closely match the microarchitecture assumed. Second, capturing the effects of different application-specific integrated circuit (ASIC) implementation flows and flow options is difficult.

The above shortcomings of existing template-based models have led us to explore new directions to improve the efficiency of on-chip router models for early-stage design space exploration. To accomplish this goal, we start from an existing RTL description of a router with any given architecture. We then create a library of fully-synthesizable router RTLs using different microarchitectural and implementation parameters. Using an industrial implementation flow, we take the RTL descriptions through the physical design steps and compute corresponding power and area values. We then apply nonparametric regression technique on the generated power and area data sets to develop accurate architecture-level router power and area models. The highlight of our modeling methodology is the decoupling of the router microarchitecture and underlying circuit implementation from the modeling effort. The contributions of this chapter are as follows.

- We introduce ORION 2.0, a set of on-chip router power and area models to provide accurate estimations with respect to different architectural parameters.
- We support accurate on-chip router power and area estimations down to the 32 nm technology node by providing corresponding scaling factors derived from multiple reliable sources (e.g., [6] and [13]).
- To enhance the accuracy of the existing template-based models, we consider decoupling the router microarchitecture and underlying circuit implementations from the modeling effort. This enables a modeling methodology in which both router microarchitecture and underlying circuit implementations are transparent to the system-level designer.
- We propose a new *framework* for modeling on-chip router power, performance, and area using machine learning-based nonparametric regression methods.

- We introduce a reproducible flow to further aid automatic generation of accurate architecture-level estimations.
- Separately, we have released both ORION 2.0 [12] and machine learning-based models [17] to enable further NoC research and design.

The remainder of this chapter is organized as follows. Section 4.2 describes ORION 2.0, a set of accurate on-chip router power and area models. Subsection 4.2.4 gives a detailed evaluation of ORION 2.0 models with respect to (1) microarchitectural parameters, (2) technology parameters, (3) synthesis of router RTLs, and (4) two Intel prototype chips. Section 4.3 describes a new modeling approach which enables automatic generation of on-chip router power, performance, and area models. Finally, Section 4.4 concludes the chapter.

4.2 Template-Based Model Generation

Wang *et al.* [114] propose ORION, a set of architectural power models for network routers, which has been widely used for early-stage NoC power estimation in academia and industry. Despite the increase in complexity of today's designs, ORION's original power models have not been updated or enhanced. In a comparison between ORION 1.0 and the Intel 80-core Teraflops chip, we notice up to 10× difference in reported total power values (see Subsection 4.2.4). This highlights the need for more accurate architectural power models to aid designers in making early-stage NoC design decisions.

In addition, since architectural design space exploration is typically done for current and future technologies, models must be derivable from standard technology files (Liberty format [8], LEF [7]), as well as extrapolatable process models (PTM [13], ITRS [6]). ORION 1.0 collects inputs from *ad hoc* sources to drive its internal power models. There is a clear need for a semi-automated flow (i.e., using shell scripting) to extract technology inputs from reliable sources, to ease the updating of models as new technology files become available.

The above factors prompt the development of ORION 2.0 and its two key goals: (1) to update and enhance ORION's power and area modeling accuracy, and (2) to en-

compass ORION 2.0 within a semi-automated flow so that ORION can be continuously maintained and easily updated. Figure 4.1 shows the usage model and modeling flow of ORION 2.0 with its main inputs and outputs.

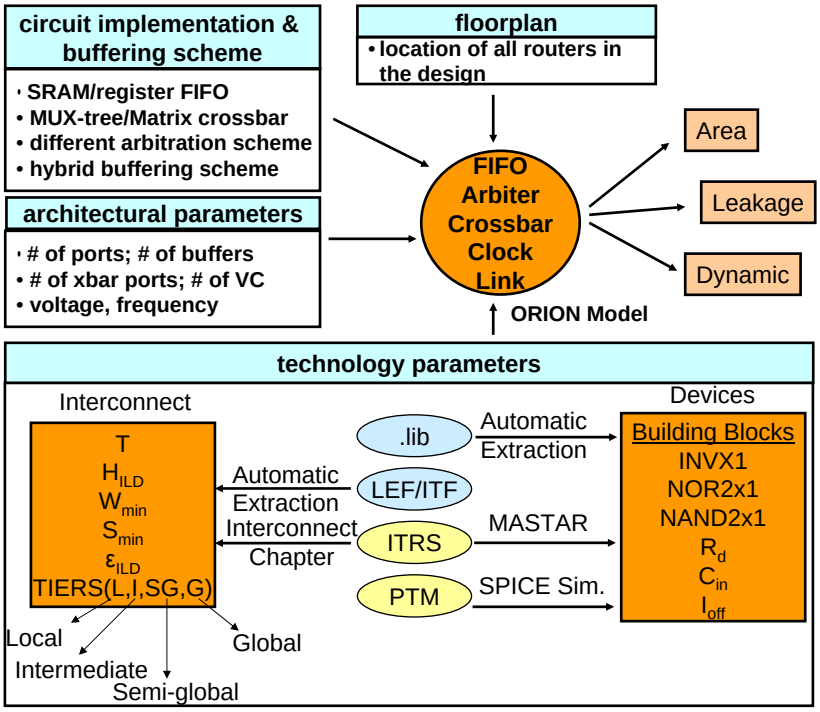

Figure 4.1: ORION 2.0 modeling methodology.

We substantially improve the original ORION 1.0 to more accurately estimate power for designs beyond the 65 nm technology node. ORION 2.0 surgically tackles various deficiencies of ORION 1.0 highlighted through validation with two Intel chips (Subsection 4.2.4) and our close interactions with both chip designers and the developers of ORION 1.0. Table 4.1 summarizes the contributions of ORION 2.0 beyond the original ORION 1.0.

New:

- Flip-flop (FF) and clock power models (both leakage and dynamic) are added. Flip-flop power models enable the faithful modeling of flip-flop-based FIFOs in addition to the SRAM-based implementation in ORION 1.0. Clock power is a ma-

Table 4.1: ORION 2.0 contributions versus ORION 1.0.

Component	App-Specific Sizing	Dynamic Power	Leakage Power	Area
SRAM-FIFO	Improved	Orig	Orig	New
Reg-FIFO	Improved	New	New	New
Crossbar	Improved	Orig	Orig	New
Arbiter	Improved	Orig	New	New
Clock	Improved	New	New	New
Link	Improved	New	New	New

jor component of overall chip power especially in high-performance applications [57], but was omitted in ORION 1.0.

- Link power models are added, leveraging accurate models that are proposed in Chapter 3. Prior existing works on link power and delay modeling [54, 55] focus on minimum-delay buffer insertion, whereas we adopt a hybrid solution which minimizes a weighted product of delay and power. ORION 1.0 did not have a parameterized link model.

- The virtual-channel (VC) allocator microarchitecture in ORION 1.0 is modified to optimize its power consumption. A new VC allocation model, based on the microarchitecture and pipeline proposed in [69], is added in ORION 2.0.

- Arbiter leakage power, previously not covered in ORION 1.0, is now modeled.

- An accurate area model is added, allowing for detailed router floorplanning which enhances the accuracy of early-stage power estimation.

- A semi-automatic flow for extracting technology parameters from standard technology files (e.g., Liberty format [8], LEF [7]), as well as extrapolatable models (PTM [13], ITRS [6], etc.) is added to allow ORION 2.0 to be easily and continuously updated in the future.

Improved:

- Application-specific technology-level adjustments (use of different V_{th} flavors and transistor widths) are used in ORION 2.0 to improve power estimation for system-on-chip (SoC) and high-performance applications. ORION 1.0 used a single set of parameters for all designs at a given technology node.

Updated:

- Transistor sizes and capacitance values are updated in ORION 2.0 with new process technology files – industry SPICE models and Interconnect Technology Format (ITF) – instead of ad hoc scaling factors as in ORION 1.0.

Our power model is validated against the Intel 80-core Teraflops chip [57] and the Intel Scalable Communications Core [62], and is within -7% and +11% of the corresponding total power values.

4.2.1 Dynamic Power Modeling

We derive parameterized equations for estimating switching capacitance of (1) clock network, (2) flip-flop-based FIFO buffers, (3) allocators and arbiters, and (4) physical links.

Clock Network

Clock generation and distribution comprise a major portion of power consumption in synchronous designs [47], representing up to 33% of power consumption in a high-performance router [57]. We estimate the term $c_{clk} = c_{sram-fifo} + c_{flip-flop-fifo} + c_{pipeline-registers} + c_{wiring}$, where $c_{sram-fifo}$, $c_{flip-flop-fifo}$, $c_{pipeline-registers}$, and c_{wiring} are capacitive loads due to SRAM-based FIFO buffers, flip-flop-based FIFO buffers, pipeline registers, and clock distribution wiring, respectively. Given that the load of the clock distribution network heavily depends on its topology, we assume an H-tree distribution style. Below, we show how to calculate each of the above capacitive components.

- *SRAM-Based FIFO Buffers.* We adapt the original ORION 1.0 model for SRAM buffers to determine the precharge circuitry capacitive load on the clock network. In an SRAM FIFO with flitwidth fw, the total capacitance due to precharging

circuitry, with n_{read} and n_{write} being the number of read and write ports, can be estimated as $c_{sram-fifo} = (n_{read} + n_{write}) \times fw \times c_{PMOS}$, where c_{PMOS} is the precharging capacitance.[1]

- *Flip-flop-Based FIFOs.* We assume a simple Dflip-flop (DFF) as the building block for flip-flop-based FIFOs. In a n_{buf}-entry flip-flop-based FIFO with flitwidth fw, the capacitive load on the clock can be estimated as $c_{flip-flop-fifo} = fw \times n_{buf} \times c_{ff}$.[2]

- *Pipeline Registers.* We also assume DFF as the building block of the pipeline registers. In a router with flitwidth fw and $n_{pipeline}$ pipeline registers, the capacitive load on the clock due to pipeline registers is $c_{pipeline-registers} = n_{pipeline} \times c_{ff}$, where $n_{pipeline} = n_{port} \times fw$ for buffers (i.e., input and output) and crossbar components, $n_{pipeline} = 2 \times (n_{port} \times n_{vc})^2$ for VC allocator, and $n_{pipeline} = n_{port} \times n_{vc} + n_{port}^2$ for switch allocator. c_{ff} is the flip-flop capacitance and is extracted from 65 nm HP (high-performance) and LP (low-power) libraries. n_{port} and n_{vc} are number of ports and number of virtual channels, respectively.

- *Wire Load.* We assume a buffered H-tree clock distribution within each individual router block. If the router block dimension is D (typically, tens of microns – e.g., D = 25 μm in the router block of each tile in the Intel 80-core chip), the total wire capacitance of an L-level H-tree is $\sum_{i=0}^{L-1} \frac{2^i \times D}{2^{\lfloor \frac{i}{2} \rfloor + 1}} \times c_{int}$ where each term is (number of segments per level) $\times$ (fraction of D per segment at that level) $\times$ (router dimension D) $\times$ (per unit length wire capacitance c_{int}). E.g., for a 5-level H-tree, the total wire capacitance is $(\frac{1\times D}{2} + \frac{2\times D}{2} + \frac{4\times D}{4} + \frac{8\times D}{4} + \frac{16\times D}{8}) \times c_{int}$. We use a fixed number of levels (equal to 5) in the H-tree; this can both overestimate clock tree wiring cost (since an H-tree is more expensive than skew-bounded Steiner constructions) and underestimate it as well (since some router configurations have significantly more than 32 leaves (sinks)). However, since the flip-flops in a router have strong spatial clustering (e.g., in FIFOs), we have opted to use the fixed number of levels. The small value of D reduces the impact of this modeling error.

[1] We use a conventional PMOS to model the precharging transistor.

[2] n_{buf} denotes buffer size in terms of number of flits per virtual channel.

Flip-flop-Based FIFO Buffers

FIFO buffers can be implemented using either SRAMs or registers. The ORION 1.0 model supports only the use of SRAM-based FIFOs. We use FFs as the building blocks of the registers. Register-based FIFOs can be implemented as a shift register or as a matrix of FFs.

- *Shift Register-Based FIFOs.* For an n_{buf}-entry FIFO, the shift register-based FIFO can be implemented as a series of n_{buf} flip-flops. We consider both read and write operations. The write operation occurs at the tail of the shift register. Assuming the new flit is f_{new} and the old flit is f_{old}, the number of switched flip-flops is the Hamming distance between them. Therefore, the write energy is $E_{write} = H(f_{new}, f_{old}) \times E^{ff}_{switch}$, where E^{ff}_{switch} is the energy to switch one bit. We simply estimate the average switching activity as $\overline{H} = \frac{fw}{2}$; then, the average write energy is $\overline{E}_{write} = \overline{H} \times E^{ff}_{switch}$. The read operation has two steps: (1) reading the head flit into the crossbar which does not consume any energy in the buffer, and (2) shifting all the subsequent flits one position toward the header. Hence, the average read energy is $\overline{E}_{read} = (fw - 1) \times \overline{E}_{write}$.

- *Matrix of FFs FIFOs.* A better approach to implement flip-flop-based FIFOs may be to use a matrix of FFs with write and read pointers as is done in SRAM-based FIFOs to avoid read and write energy consumption at every cycle due to shifting of flits. To implement this, we add a control circuitry to an existing matrix of FFs to handle the operation of write/read pointers. The *write pointer* points to the head of the queue, and the *read pointer* points to the tail of the queue. The pointer advances one position for each write or read operation. To model power, we can synthesize the RTL of the above implementation and obtain corresponding power numbers with respect to different buffer size and flitwidth values. To develop a closed-form power model, linear regression can be used to derive the power of the control unit as a function of buffer size and flitwidth. In this implementation, read energy is only due to pointer shifts, $\overline{E}_{read} = \overline{E}_{pointer}$, whereas write energy is due to pointer shifts and bit switches, $\overline{E}_{write} = \overline{H} \times \overline{E}^{ff}_{switch} + \overline{E}_{pointer}$, where $\overline{E}_{pointer}$ is the average energy to advance one position for read or write pointers.

Allocators and Arbiters

We modify the separable VC allocator microarchitecture in ORION 1.0 to optimize its power consumption. Instead of two stages of arbiters, we have a single stage of $n_{port} \times n_{vc}$ arbiters, each governing one specific output VC, where n_{port} and n_{vc} are the number of router ports and virtual channels, respectively. Instead of sending requests to all output VCs of the desired output port, an input VC first checks the availability of output VCs, and then sends a request for any available output VC. The arbiters will resolve conflicts where multiple input VCs request the same output VC. This design has lower matching probability, but uses only one stage of arbiters, and hence significantly reduces power. We also add a new VC allocator model in ORION 2.0 which models VC allocation as VC "selection" instead, as is proposed in [69]. Here, a VC is selected from a queue of free VCs, after switch allocation. Thus, the complexity (delay, power and area) of VC allocation does not grow with the number of VCs.

Physical Links

The dynamic power of links is primarily due to charging and discharging of capacitive loads (wire and input capacitance of the next-stage repeater). We use a hybrid buffering solution that minimizes a linear combination of delay and power. We exhaustively evaluate a given objective function for a given number and size of repeaters, while searching for the optimal (number, size) values. Dynamic power is given by $p_{dyn}^{link} = a \times c_{load} \times v_{dd}^2 \times f_{clk}$, and $c_{load} = c_{in} + c_g + c_c$, where p_{dyn}^{link}, a, c_{load}, v_{dd} and f_{clk} denote the link dynamic power, activity factor, load capacitance, supply voltage, and clock frequency, respectively. The load capacitance is the sum of the input capacitance of the next repeater (c_{in}), and the ground (c_g) and coupling (c_c) capacitances of the wire. Here, link power refers to power consumption of the links incident to the router (i.e., connecting ports of the given router to ports of adjacent routers). We count only the input link power, so that when composing router power models for an entire NoC, there is no double-counting.

4.2.2 Leakage Power Modeling

As technology scales to deep submicron processes, leakage power becomes increasingly important as compared to dynamic power. Thus, there is a growing need to characterize and optimize network leakage power as well. Chen *et al.* [38] propose an architectural methodology for estimation of leakage power. However, they only consider subthreshold leakage whereas from 65 nm and beyond gate leakage gains importance and becomes a significant portion of the leakage power. We follow the same methodology proposed in [38] with addition of gate leakage consideration.

To derive an architectural leakage model, we can separate the technology-independent variables such as transistor width from technology-dependent variables such as leakage current per unit transistor width. Total leakage current is calculated as, $i_{leak}(g, s)$ = $w(g, s) \times (i'_{sub}(g, s) + i'_{gate}(g, s))$, where i'_{sub} and i'_{gate} are subthreshold and gate leakage currents per unit transistor width for a specific technology, respectively, and $w(g, s)$ refers to the effective transistor width of gate g at state s. We measure i'_{sub} and i'_{gate} for a variety of circuit components, input states, operating conditions (i.e., voltage and temperature), and different V_{th} flavors, i.e., high V_{th} (HVT), normal V_{th} (NVT), and low V_{th} (LVT). We compose the architectural leakage power model in a bottom-up fashion for each building block [38].

Arbiter Leakage Power

In ORION 2.0, we add arbiter leakage power, and support matrix and round robin arbiters. Given a matrix arbiter with req requesters, the request priorities may be represented by an $req \times req$ matrix, with a '1' in row i and column j if requester i has higher priority than requester j, and 0 otherwise. Let req_i be the i^{th} request, gnt_n the n^{th} grant, and m_{ij} the element in the i^{th} row and j^{th} column in the matrix. The grant logic can be denoted as $gnt_n = req_n \times \prod_{i<n}(\overline{req_i} + \overline{m_{in}}) \times \prod_{i>n}(\overline{req_i} + \overline{m_{ni}})$. Then, we decompose the grant logic into elementary building blocks including NOR, INV, and DFFs, and compute the leakage current for the entire arbiter as $i_{leak}(arbiter) = i_{leak}(NOR2) \times ((2req - 1) \times req) + i_{leak}(INV) \times req + i_{leak}(DFF) \times \frac{req(req-1)}{2}$.[3] The

[3]For a given elementary building block, X, $i_{leak}(X)$ is calculated using the $w(X)$, $i'_{sub}(X)$, and $i'_{gate}(X)$.

previous equation can readily be obtained from the gate-level netlist of a given arbiter, if available. Hence, arbiter power can be computed as $p_{leak}(arbiter) = i_{leak}(arbiter) \times v_{dd}$, where v_{dd} is the supply voltage. Similarly, for a round-robin arbiter we break its corresponding grant logic into elementary building blocks (i.e., NOR and INV), and use DFFs to store the priority bits.

Physical Link Leakage Modeling

The leakage power of links is due to repeaters. In repeaters, leakage occurs in both output states. NMOS devices leak when the output is high, while PMOS devices leak when the output is low. This holds for buffers as well, because the second-stage devices are the primary contributors due to their large sizes. Leakage power has two main components, subthreshold leakage and gate-tunneling current. Both components depend linearly on device size and are modeled using linear regression with values obtained from SPICE simulations.

4.2.3 Area Modeling

With the increase in number of cores on a single chip, the area occupied by the communication components such as links and routers increases. As area is an important economic incentive in integrated circuit (IC) design, it must be estimated early in the design flow to enable design space exploration. In this subsection, we present accurate models for router and link area.

Router Area

To estimate router area, we basically compute the area of each building block, sum them up and add an additional 10% (rule of thumb) to account for global whitespace. For each building block, we first identify the implementation style of the block and then decompose the block into its basic logical elements. For example, for SRAM-based FIFOs we can compute wordline length using $l_{wordline} = fw \times (w_{mem,cell} + 2 \times (n_{read} + n_{write}) \times s_{int})$, and bitline length using $l_{bitline} = n_{buf} \times (h_{mem,cell} + (n_{read} + n_{write}) \times s_{int})$, where fw, n_{buf}, $w_{mem,cell}$, $h_{mem,cell}$, s_{int}, n_{read}, and n_{write} are flitwidth in bits, buffer

size, memory cell width, memory cell height, interconnect spacing, number of read ports, and number of write ports, respectively. The total area for an n_{buf}-entry buffer is then calculated as $A_{fifo} = l_{wordline} \times l_{bitline}$. For other router components, namely, crossbar and arbiter, we similarly decompose them into their circuit building blocks (i.e., gate-level netlist). By applying the gate area model, we estimate the area of individual circuit components and compute the area of the entire block. Link area models have been described in Subsection 3.3.4.

4.2.4 Model Evaluation and Discussion

In this subsection, we provide further insight into the proposed models with respect to (1) different microarchitectural parameters, (2) different technology nodes and transistor types, (3) synthesis of router RTLs, and (4) two recent NoC prototypes. ORION 2.0 models can be broadly classified as *template-based*, that is, derived from a mix of circuit templates, e.g., matrix crossbar, SRAM-based FIFO, etc.

Microarchitectural Parameters

We investigate the impact of different microarchitectural parameters on router power and area. We demonstrate that ORION 2.0 models behave as expected with respect to each parameter. Router microarchitectural components include (1) buffers, (2) crossbar, (3) virtual channel allocator, (4) switch allocator, (5) clock, and (6) link. The microarchitectural parameters for each router are: (1) buffer size per VC per port, (2) flitwidth, (3) number of VCs, and (4) number of ports.[4] For all the experiments, we use a supply voltage of 1.1 V, switching activity of 0.3, and a clock frequency of 3 GHz in 65 nm technology. In each experiment, we only vary one microarchitectural parameter of interest and keep the others fixed. Nominal values for buffer size, flitwidth, number of VCs and number of ports are four flits, 32 bits, one queue per port (i.e., wormhole configuration) and five, respectively.

Buffer. Buffer power and area are affected by buffer size, flitwidth, number of VCs, and number of ports. When we vary buffer size, we expect both dynamic and leak-

[4] We assume the crossbar has the same number of ports as the router.

age power of buffers to increase linearly. This is because buffer size linearly increases precharge capacitance load and the number of bitcell transistors. When we vary flitwidth, we again expect buffer dynamic and leakage power to increase linearly, since flitwidth linearly increases the precharge and bitline capacitances as well as the number of bitcell transistors.

On the other hand, as we increase the number of VCs, buffer dynamic power will not change, since the number of flits arriving at each input port is the same. However, we expect buffer leakage power to increase linearly, since VC routers have n_{vc} queues in each input port, where n_{vc} is the number of VCs. If we increase the number of ports, we expect buffer dynamic and leakage power to increase linearly; addition of a new port will add a new buffer set, with the same buffer size and flitwidth.

Buffer area follows similar trends as buffer power. As buffer size increases, we expect buffer area to increase linearly. This is because a buffer size increase of one unit increases the number of flits per buffer by one unit. In addition, buffer area changes linearly with flitwidth because flitwidth linearly increases the number of bitcells in each FIFO entry.

Crossbar. Crossbar power and area are affected by the number of router ports. If we increase the number of ports, we expect dynamic and leakage power to increase quadratically. This is because an $n_{port} \times n_{port}$ crossbar allows arbitrary one-to-one connections between n_{port} input ports and n_{port} output ports. Similarly, if we increase the number of ports, we expect crossbar area to increase quadratically.

VC and Switch Allocator. Dynamic and leakage power are expected to increase linearly and quadratically, respectively, with the number of VCs. This is because the number of arbiters increases linearly with the number of VCs. In addition, for each arbiter the request width increases linearly with the number of VCs. Hence, leakage power increases quadratically with the number of VCs. Since the utilization rate of each arbiter is assumed to be inversely proportional to the number of VCs, dynamic power is expected to change linearly with the number of VCs.[5] In our experiments, we have assumed a

[5]Note that VC allocator dynamic power is equal to arbiter utilization rate multiplied by the product of per-arbiter dynamic power and the total number of arbiters. Hence, VC allocator dynamic power is

two-stage separable VC allocator. For the switch allocator, if we increase the number of VCs, dynamic power and leakage power are expected to increase linearly: the switch allocator, the request width of each arbiter increases linearly with the number of VCs.

If we increase the number of ports, we expect VC allocator dynamic and leakage power to increase quadratically. This is because the request width for each arbiter in the second stage of arbitration increases linearly with respect to number of ports, and the number of such arbiters is also proportional to the number of ports. Similarly, VC allocator area is expected to increase quadratically with number of VCs and number of ports. Switch allocator area changes linearly and quadratically, respectively, with number of VCs and number of ports.[6]

In addition to the above 'sanity' checks, we evaluate the leakage power model by verifying that the leakage power density (defined as total leakage power / total gate width) remains the same as we change any of the microarchitectural parameters for different components. We observe that leakage power density for buffer, crossbar, and arbiter is 0.0003 mW/μm of gate width that we study in the 65 nm technology.

Technology Parameters

In ORION 2.0 we include transistor sizes and capacitance values for three combinations of V_{th} and transistor width: (1) large transistor size with LVT for high-performance, (2) nominal transistor size with NVT for general-purpose, and (3) small transistor size with HVT for low-power designs. When transistor type changes from HVT to NVT to LVT, dynamic power is expected to increase due to the increase in transistor width (i.e., assuming a fixed technology), and leakage power is expected to increase due to increase in transistor width, and decrease in threshold voltage, as confirmed in Figure 4.2. In the experiment for Figure 4.2, we use a router with five ports, two VCs, four-flit buffers, and 32-bit flitwidth; for HVT, NVT, and LVT we use (0.8 V, 0.2 GHz), (1.0 V, 1 GHz), and (1.1 V, 3 GHz), respectively.

Also, for a given transistor type, dynamic power is expected to decrease due to smaller gate areas as technology advances, and leakage power is expected to increase

linearly dependent on number of VCs (i.e., $\frac{1}{n_{vc}} \times n_{vc} \times n_{vc} = n_{vc}$).

[6] In addition, we observe that the clock and link power and area models follow expected trends.

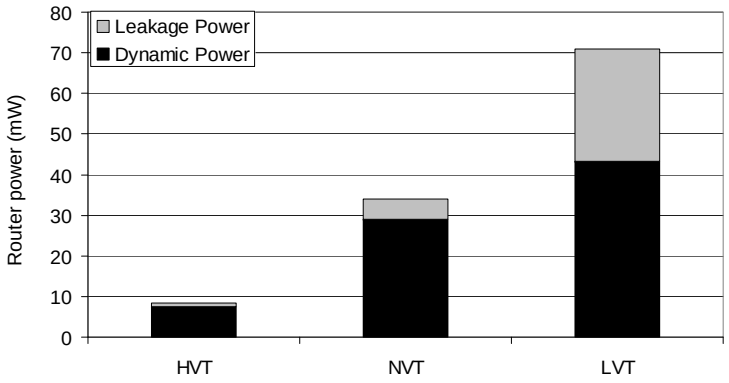

Figure 4.2: Power consumption versus transistor type.

due to leakier devices as confirmed in Figures 4.3(a), (b), and (c).[7] We use similar microarchitectural parameters and transistor types, but vary technology node from 90 nm down to 32 nm.

Router RTL Synthesis

We further validate the trend of the proposed models by comparing them against router RTL synthesis data. We use *Netmaker*, a library of fully-synthesizable parameterized NoC implementations [11]. A baseline VC router is used in which VC allocation and switch allocation are performed sequentially in one clock cycle.

Using automation scripts, we generate a corresponding RTL code for each combination of the above parameters. We then synthesize the RTL codes using the TSMC 65 nm GP cell library. Figures 4.4 and 4.5 show that ORION 2.0 models' trends (cf. Subsection 4.2.4) match those of synthesized routers. In our comparisons, we use a supply voltage of 0.9 V. We attribute differences between ORION 2.0 and the synthesized router results to the fact that ORION 2.0 does not capture the effects of the implementation flows. Modern IC implementation flows incorporate powerful logic synthesis and physical synthesis transformations (logic restructuring, gate sizing, etc.) to satisfy power and performance constraints. The detailed impacts of such transformations are difficult to capture at early stages of the design where not all the implementation information is available.

[7] Our estimations for 45 nm and 32 nm technologies are derived using scaling factors from ITRS [6].

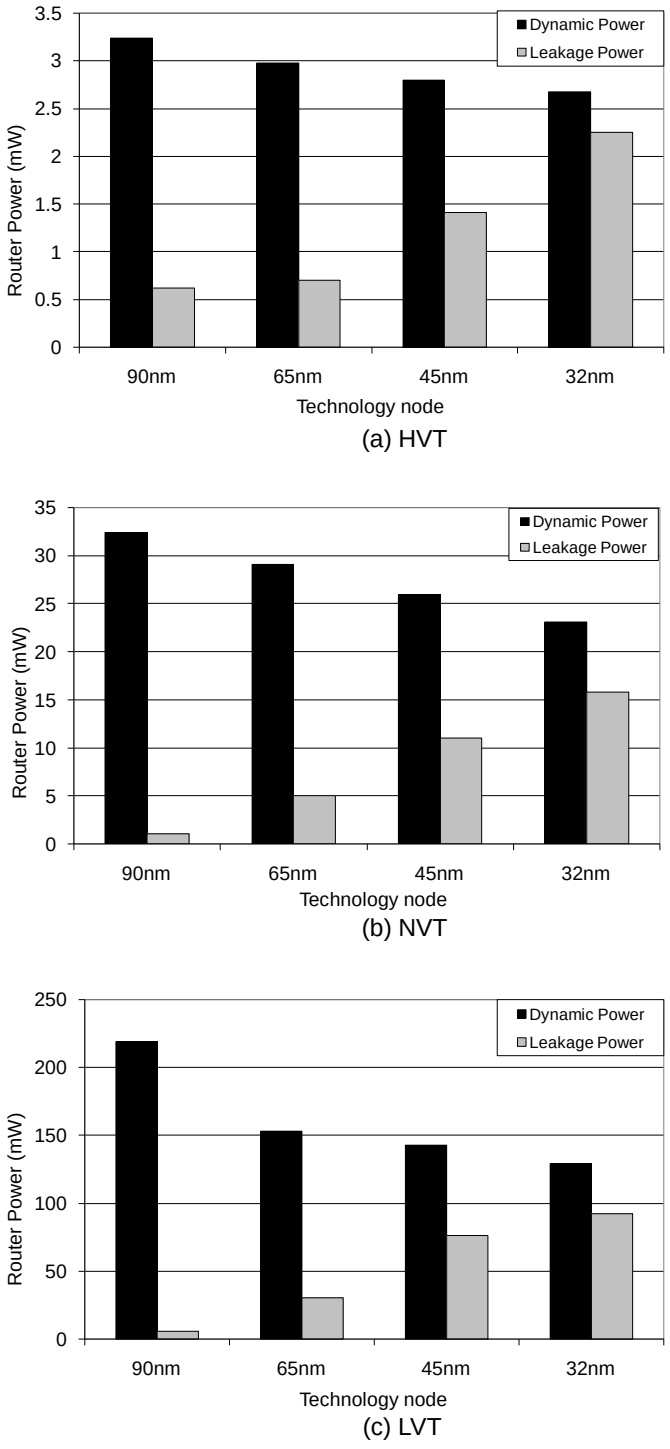

Figure 4.3: Router power versus technology node with (a) HVT, (b) NVT, and (c) LVT transistors.

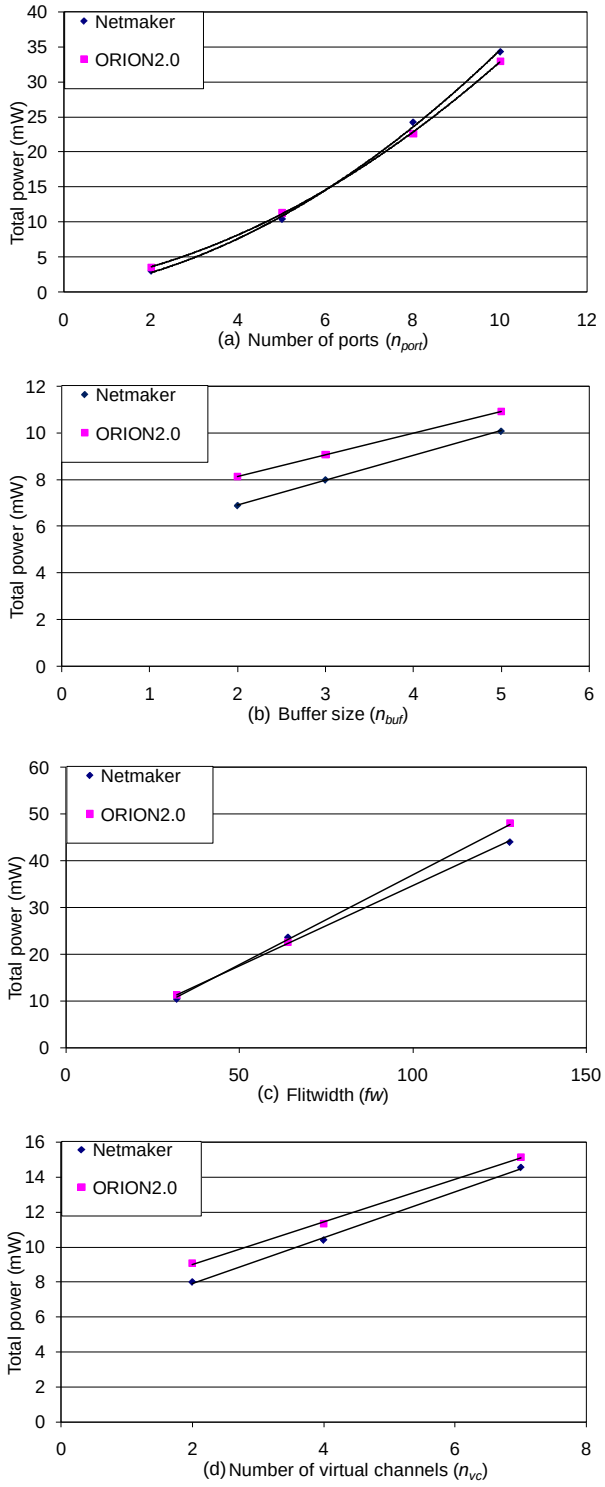

Figure 4.4: Router total power versus (a) number of ports, (b) buffer size, (c) flitwidth, and (d) number of virtual channels.

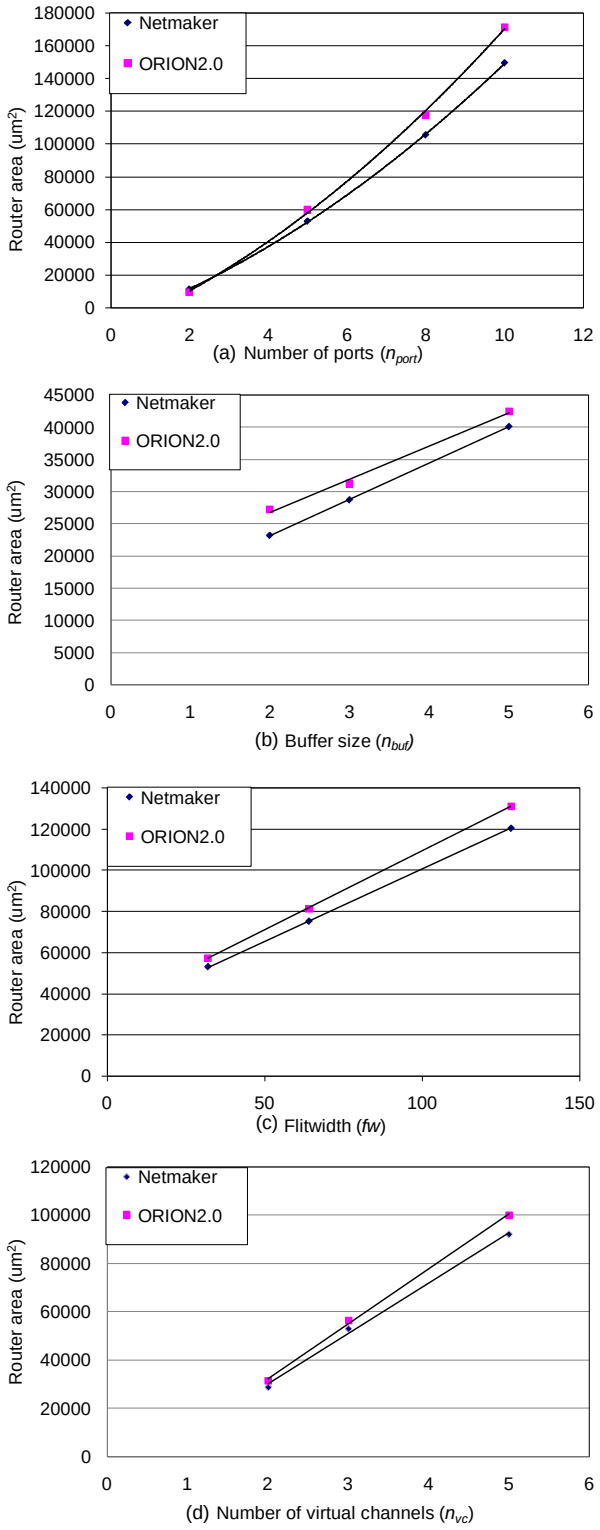

Figure 4.5: Router area versus (a) number of ports, (b) buffer size, (c) flitwidth, and (d) number of virtual channels.

Real Router Designs

Finally, we also validate ORION 2.0 models by comparing them to post-layout and pre-layout simulations of recent NoC prototypes: (1) the Intel 80-core Teraflops chip [57], targeted for high performance chip multiprocessors, and (2) the Intel Scalable Communications Core (SCC) chip [62], targeted for ultra low-power multiprocessor systems-on-chip. As noted in the introduction, there is up to 8× difference between ORION 1.0 estimations (per component) and the Intel 80-core chip silicon measurements. Also, the estimated total power is about 10× less than actual. Again, ORION 1.0 does not include clock and link power models. Figure 4.6 shows the percentage of each of the power components for the Intel 80-core chip, and the same statistics from ORION 1.0 and ORION 2.0 models. We observe that ORION 2.0 more accurately represents the impact of each individual component.[8]

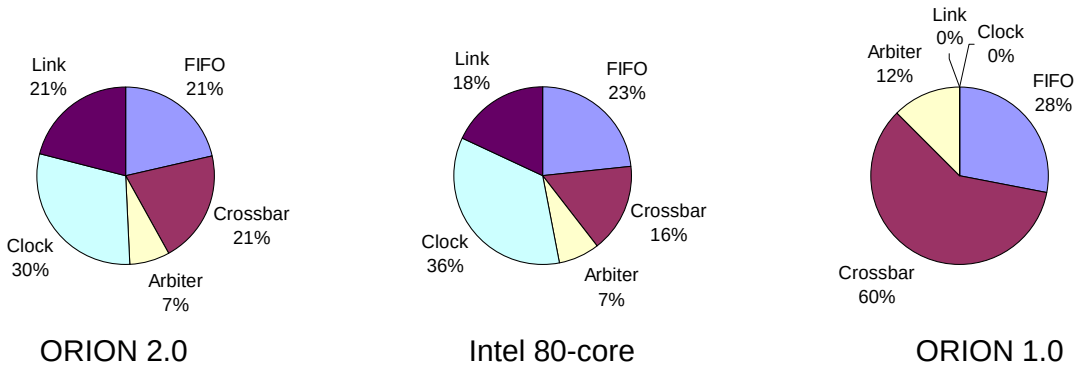

Figure 4.6: Power breakdown of the Intel 80-core chip versus estimations from ORION 1.0 and ORION 2.0 models.

The router configurations for the Intel 80-core and the Intel SCC chips are shown in Tables 4.2 and 4.3, respectively. We use switching activity of 0.15 for both testcases. The estimated total power consumption, using ORION 2.0 models, is within -7% and +11% of the Intel 80-core post-layout and the Intel SCC pre-layout power estimations, respectively. In addition, the estimated total area, using ORION 2.0 models, is within -23.5% and +25.3% of the Intel 80-core, and Intel SCC, respectively. We attribute the deviation of ORION 2.0 estimation from the Intel 80-core simulations to (1) difference

[8]We do not have access to the power breakdown for the Intel SCC design.

in library files (i.e., between Intel's and those used in our study), (2) our use of 6T SRAM versus Intel's use of 8T SRAM buffers, and (3) no consideration for the tile floorplan. In addition, for the Intel SCC chip, overestimation of area may be attributed to the fact that Intel SCC is an ultra low-power product with the majority of its components being custom; by contrast, we have developed our area models using gate area values from generic libraries.

Table 4.2: Intel 80-core router configuration.

Voltage	Frequency	Transistor type	Number of ports
1.2 V	5.1 GHz	LVT	5
Number of VCs	Input buffer	Output buffer	Flitwidth
2	16	0	39

Table 4.3: Intel SCC router configuration.

Voltage	Frequency	Transistor type	Number of ports
1.08 V	250 MHz	HVT	5
Number of VCs	Input buffer	Output buffer	Flitwidth
1	2	1	32

4.3 Machine Learning-Based Model Generation

To quantify the limitations of template-based models, we evaluate the accuracy of ORION 2.0 and parametric models against actual post-layout power results for different router configurations. We use *Netmaker* [11], a public-domain tool that generates fully-synthesizable RTL for parameterized input-buffered VC routers. Even when provided with the actual (TSMC 65 nm) library information, ORION 2.0 [63] and parametric models [27, 30, 36, 77] have significant deviation (40% and 28% on average) from the actual power values, respectively. The fact that the architecture underlying ORION 2.0 does not completely match the architecture assumed in [11] supports a

premise that the accuracy of template-based models degrades as the underlying architecture or circuit implementation changes. Such inaccuracy in estimation can lead to erroneous NoC design choices.

In the following subsections, we propose a new modeling approach which exploits the accuracy of post-layout analysis and machine learning-based nonparametric regression to develop on-chip router power, performance, and area models.

4.3.1 Implementation Flow and Scope of Study

Implementation Flow and Tools

Figure 4.7 shows our physical implementation flow, which includes the traditional synthesis, placement and routing steps plus static timing analysis and model generation, scripted for "push-button" use. At each step we require that the design meets timing requirements before it can pass on to the next step.

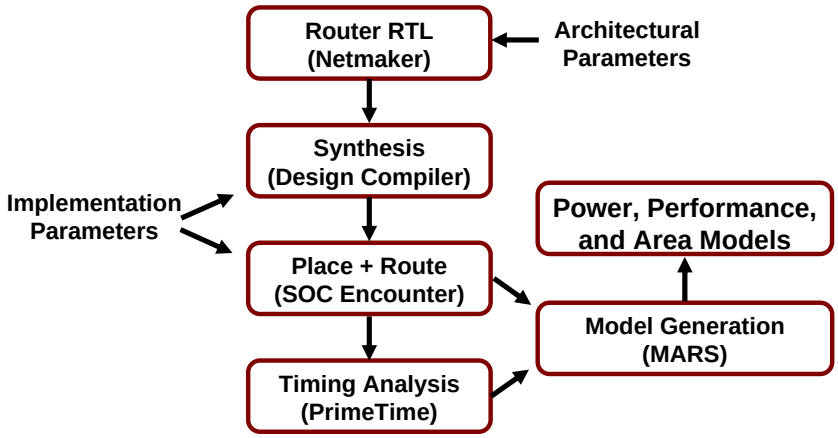

Figure 4.7: Implementation flow.

In our flow, we first synthesize corresponding RTL codes of each of our testcases with worst-case timing libraries. To mimic a typical industrial timing-driven implementation flow, we impose the target frequency as the primary constraint, with area and power minimization being optimization objectives.

Using synthesized netlists, we implement the designs through place and route steps using different row utilization and aspect ratio values at the floorplan stage. In

addition, we use a multi-V_{th} flow in which cells are chosen from a selection of HVT, NVT, and LVT to obtain a larger leakage versus frequency tradeoff envelope. After routing, we obtain power and area values which are used in power and area model generation. Finally, we perform static timing analysis to obtain the longest (critical) path delay values which are used for performance model generation.

We use *Netmaker v0.82* [11] to generate a library of synthesizable on-chip router RTL codes. We perform our experiments using multi-V_{th} libraries in TSMC 65 GP technology. We use *Synopsys Design Compiler v2009.06-SP2* [14] to synthesize the RTL codes, *Cadence SOC Encounter v7.1* [4] to execute the place and route flow, and *Synopsys PrimeTime v2007.12-SP3* [16] for static timing analysis. Finally, *MARS3.0* [9] is used for nonparametric modeling.

Scope of Study

We focus on the microarchitectural and implementation parameters that are of interest at the system level, and that significantly affect the quality of results. We use a baseline virtual channel (VC) router in which VC allocation and switch allocation are performed sequentially in one clock cycle. In a VC router, the microarchitectural parameters are: (1) flitwidth, fw; (2) number of virtual channels, n_{vc}; (3) number of input and output ports, n_{port}; and (4) buffer size, n_{buf}. Table 4.4 shows the microarchitectural parameters and the values they take on in our study.

Table 4.4: List of microarchitectural parameters used in our studies.

Parameter	Values
fw	{16, 24, 32}-bits
n_{vc}	{2, 3, 5}
n_{port}	{3, 5, 7}
n_{buf}	{3, 5, 8}-flit buffers

The implementation parameters include (1) clock frequency, f_{clk}; (2) aspect ratio, AR; (3) row utilization, $util$; and (4) type of V_{th} flavors (V_{th}). Target clock frequencies for the router design are 200 MHz, 400 MHz, and 700 MHz. We use three aspect

ratio values, 1.0, 2.25, and 3.75, and three row utilizations, 50%, 75%, and 90%. We also use single-V_{th} (NVT) and triple-V_{th} library flavors.

4.3.2 Modeling Methodology

We propose a new paradigm that uses machine learning for non-parametric model generation. A baseline model generation flow is as follows.

- We begin with a parameterized synthesizable RTL specification for a given router microarchitecture – i.e., a *configurable* router microarchitecture specification – which we use to generate actual router implementations across different configuration parameters.
- A small subset of configurations is selected for training, and we run each configuration in this training set through the ASIC implementation flow to obtain a detailed physical layout for each router instance.
- Finally, we apply machine learning-based nonparametric regression techniques to power, performance, and area data from the training set to derive the corresponding estimation models.

4.3.3 On-Chip Router Models

We model both dynamic and leakage power components. Dynamic power is due to charging and discharging of switching capacitances, and leakage power is due to subthreshold and gate leakage currents. Thus, the goal is to model dependence of switching capacitance and leakage current on microarchitectural and implementation parameters. To model performance we define maximum *implemented* clock frequency to be the reciprocal of the maximum path delay obtained for a given combination of implementation and microarchitectural parameters. Maximum implemented clock frequency primarily depends on the given cycle time constraint. However, it is also affected by the V_{th} flavors of the library used. Similarly, we model the dependence of the sum of standard cell areas on microarchitectural and implementation parameters. Figures 4.8, 4.9 and 4.10

illustrates the form of resulting router power, performance, and area models for a target router in 65 nm [17].

Basis Functions

b_1 = max(c_{clk} – 200); b_2 = max(n_{port} – 3) ×b_1; b_3 = max(n_{vc} – 2) ×b_2; b_4 = max(n_{buf} – 3) ×b_3; b_5 = max(n_{buf} – 3); b_6 = max(n_{vc} – 3); b_7 = max(3 – n_{vc}); b_8 = max(n_{port} – 5); b_9 = max(5 – n_{port}); b_{10} = max(fw – 16) ×b_1; b_{11} = max(fw – 16) ×b_4; b_{12} = max(f_{clk} – 400) ×b_5; b_{13} = max(400 – f_{clk}) ×b_5; b_{14} = max(n_{vc} – 2) ×b_1; b_{15} = max(n_{vc} – 2) ×b_8; b_{16} = max(n_{port} – 5) ×b_5; b_{17} = (5 – n_{port}) ×b_5; b_{18} = (n_{vc} – 2) ×b_5; b_{19} = (fw – 16) ×b_5; b_{20} = (f_{clk} – 400) ×b_8; b_{21} = (400 – f_{clk}) ×b_8; b_{22} = (fw – 16) ×b_{15}; b_{23} = (vth – 1) ×b_1; b_{24} = (n_{vc} – 3) ×b_{20}; b_{25} = (3 – n_{vc}) ×b_{20}; b_{26} = (fw – 16) ×b_6; b_{27} = (fw – 16) ×b_2; b_{28} = (f_{clk} – 200) ×b_{16}; b_{29} = (f_{clk} – 400) ×b_{18}; b_{30} = (400 – f_{clk}) ×b_{18}; b_{31} = (util – 0.5) ×b_2; b_{32} = (n_{port} – 5) ×b_{18}; b_{33} = (5 – n_{port}) ×b_{18}; b_{34} = (vth – 1) ×b_4; b_{35} = (f_{clk} – 200) ×b_{17};

Power Model

p_{router} = 5.02073 + 0.00906348 ×b_1 + 0.00347286 ×b_2 + 0.00205786 ×b_3 + 0.000192719 ×b_4 + 1.1676 ×b_5 + 0.772373 ×b_6 – 1.0305 ×b_7 + 1.49016 ×b_8 – 1.06622 ×b_9 + 0.000384038 ×b_{10} + 2.64503e-5 ×b_{11} + 0.0036758 ×b_{12} – 0.00360293 ×b_{13} + 0.00487664 ×b_{14} + 0.378489 ×b_{15} + 0.200063 ×b_{16} – 0.115736 ×b_{17} + 0.528612 ×b_{18} + 0.0409022 ×b_{19} + 0.0042898 ×b_{20} – 0.0016548 ×b_{21} + 0.0175677 ×b_{22} - 0.00174397 ×b_{23} + 0.002946 ×b_{24} – 0.00150447 ×b_{25} + 0.0632826 ×b_{26} + 0.000116609 ×b_{27} + 0.00118851 ×b_{28} + 0.00120678 ×b_{29} – 0.000994654 ×b_{30} - 0.00204467 ×b_{31} + 0.109626 ×b_{32} – 0.0608194 ×b_{33} – 9.94631e-5 ×b_{34} – 0.00062929 ×b_{35}

Figure 4.8: Power model of a router in 65 nm.

4.3.4 Model Evaluation and Discussion

Experimental Setup

To generate the models, we randomly select 10% of the entire data set as training data; we then test the models on the other 90% of the data. To show that the selection of the training set does not substantially affect model accuracy, we randomly select 10% of the entire data set five times and show the corresponding models' maximum and average error values (Table 4.5). Furthermore, to assess the generality of the models, we validate the models against 72 data points with different clock frequencies from those described in the scope of our study (Subsection 4.3.1). The clock frequencies include 50 MHz, 100 MHz, 1200 MHz and 2000 MHz. We observe that our power and area models have 7.9% (48.4%) average (maximum) error with respect to layout data.

Basis Functions

b_1 = max(f_{clk} - 400); b_2 = max(400 – f_{clk}); b_3 = max(f_{clk} – 700); b_5 = max(f_{clk} – 200); b_7 = max(V_{th} – 1) ×b_2; b_8 = max(n_{port} – 5) ×b_2; b_9 = max(5 - n_{port}) ×b_2; b_{10} = max(n_{vc} – 3) ×b_2; b_{11} = max(3 – n_{vc}) ×b_2; b_{12} = max(n_{vc} – 2) ×b_7; b_{13} = max(n_{port} – 5) ×b_{12}; b_{14} = max(5 – n_{port}) ×b_{12}; b_{15} = max(n_{buf} – 5) ×b_9; b_{16} = max(5 – n_{buf}) ×b_9; b_{17} = max(n_{port} – 3) ×b_1; b_{18} = max(n_{buf} – 5) ×b_{11}; b_{19} = max(5 - n_{buf}) ×b_{11}; b_{20} = max(f_{clk} – 1200); b_{23} = max(0.75 – $util$) ×b_{11}; b_{24} = max(AR – 2.25) ×b_{15}; b_{25} = max(2.25 – AR) ×b_{15}; b_{26} = max(n_{port} – 3) ×b_7; b_{27} = max(n_{vc} – 3) ×b_{26}; b_{29} = max(n_{vc} – 3) ×b_9; b_{30} = max(3 – n_{vc}) ×b_9; b_{31} = max(V_{th} – 1) ×b_{29}; b_{32} = max(n_{buf} – 3) ×b_{30}; b_{33} = max(n_{vc} – 2) ×b_{17}; b_{34} = max(V_{th} – 1) ×b_{33}; b_{35} = max(V_{th} – 1) ×b_{32};

Performance Model

t_{router} = 1 / (3.7783 + 0.00371148 ×b_1 + 0.003197 ×b_2 + 0.00272797 ×b_3 – 0.0072963 ×b_5 + 0.00457976 ×b_7 + 0.000334444 ×b_8 - 0.00186965 ×b_9 + 0.000586777 ×b_{10} - 0.00242344 ×b_{11} – 0.00208348 ×b_{12} – 0.000476011 ×b_{13} + 0.00170843 ×b_{14} – 0.000392809 ×b_{15} – 0.000210417 ×b_{16} + 5.66489e-5 ×b_{17} – 0.000212663 ×b_{18} – 0.000483294 ×b_{19} + 0.000737289 ×b_{20} + 0.00271441 ×b_{23} + 0.00012598 ×b_{24} + 7.53702e-5 ×b_{25} – 3.13016e-5 ×b_{26} + 0.000648084 ×b_{27} + 0.000524559 ×b_{29} + 0.000683698 ×b_{30} – 0.00137035 ×b_{31} + 0.000130404 ×b_{32} + 4.69492e-5 ×b_{33} – 4.96511e-5 ×b_{34} + 0.00019752 ×b_{35})

Figure 4.9: Performance model of a router in 65 nm.

Basis Functions

b_1 = (n_{port} – 5); b_2 = (5 – n_{port}); b_3 = (n_{vc} – 2); b_4 = (n_{buf} – 3); b_5 = (n_{port} – 5) ×b_3; b_6 = (5 – n_{port}) ×b_3; b_7 = (n_{buf} – 3) ×b_5; b_8 = (fw – 16); b_9 = (f_{clk} – 400) ×b_5; b_{10} = (400 – f_{clk}) ×b_5; b_{11} = (n_{vc} – 2) ×b_4; b_{12} = (n_{port} – 5) ×b_4; b_{13} = (5 – n_{port}) ×b_4; b_{14} = (fw – 16) ×b_7; b_{15} = (fw – 16) ×b_{11}; b_{16} = (f_{clk} – 400); b_{17} = (400 – f_{clk}); b_{18} = (n_{port} – 3) ×b_8; b_{19} = ($util$ – 0.75) ×b_9; b_{20} = (0.75 – $util$) ×b_9; b_{21} = (n_{vc} – 2) ×b_{13}; b_{22} = (n_{buf} – 3) ×b_{18}; b_{23} = (n_{vc} – 3) ×b_{18}; b_{24} = (3 - n_{vc}) ×b_{18}; b_{25} = (f_{clk} – 400) ×b_2; b_{26} = (400 – f_{clk}) ×b_2; b_{27} = (f_{clk} – 400) ×b_3; b_{28} = (400 – f_{clk}) ×b_3; b_{29} = (V_{th} – 1) ×b_{27}; b_{30} = (V_{th} – 1) ×b_1; b_{31} = (f_{clk} – 400) ×b_{12}; b_{33} = (f_{clk} – 400) ×b_6; b_{35} = (V_{th} – 1) ×b_{33};

Area Model

A_{router} = 0.019701 + 0.00763916 ×b_1 – 0.0048896 ×b_2 + 0.00965144 ×b_3 + 0.00449707 ×b_4 + 0.00295859 ×b_5 – 0.00139435 ×b_6 + 0.000636547 ×b_7 + 0.000434499 ×b_8 + 8.81772e-6 ×b_9 – 2.60311e-6 ×b_{10} + 0.00182555 ×b_{11} + 0.000992093 ×b_{12} – 0.000652932 ×b_{13} + 6.08512e-6 ×b_{14} + 6.36385e-5 ×b_{15} + 7.46568e-6 ×b_{16} – 3.72798e-6 ×b_{17} + 7.50136e-5 ×b_{18} – 2.86688e-5 ×b_{19} + 8.65157e-6 ×b_{20} – 0.000354326 ×b_{21} + 3.5547e-5 ×b_{22} + 5.83999e-5 ×b_{23} – 2.90962e-5 ×b_{24} – 2.22046e-6 ×b_{25} + 2.32869e-6 ×b_{26} + 8.46455e-6 ×b_{27} – 1.69488e-6 ×b_{28} – 5.23805e-6 ×b_{29} – 0.000454256 ×b_{30} + 9.11546e-7 ×b_{31} – 3.22553e-6 ×b_{33} + 2.13008e-6 ×b_{35}

Figure 4.10: Area model of a router in 65 nm.

We also investigate the impact of different microarchitectural and implementation parameters on router power, performance, and area. For our experiments, we use

Table 4.5: Model stability with respect to randomization of the training set.

Experiments	power % error		performance % error	
	max	avg	max	avg
Exp 1	42.208	3.251	39.113	3.384
Exp 2	44.732	3.369	42.503	3.893
Exp 3	36.769	3.571	42.927	3.279
Exp 4	39.782	3.665	37.446	3.472
Exp 5	40.092	3.315	43.011	3.523

a supply voltage of 0.9 V, switching activity of 0.2, clock frequency of 400 MHz and a single-V_{th} 65 nm timing library. In each experiment, we only vary one microarchitectural parameter of interest and keep the others fixed at these nominal values (for buffer size, flitwidth, number of VCs and number of ports are five flits, 32 bits, one VC (i.e., wormhole configuration) and five ports, respectively).

Figures 4.11(a) and (b) give 'sanity checks' for the models. When we vary buffer size, we expect router power to increase linearly, as confirmed in Figure 4.11(a), since buffer size linearly increases the number of registers required for the additional flits.[9] Another important microarchitectural parameter is the number of router ports. If we increase the number of ports, we expect router power to increase quadratically, as confirmed in Figure 4.11(b), since the baseline VC router uses a multiplexer tree crossbar which enables arbitrary one-to-one connections between n_{port} input ports and n_{port} output ports.

The takeaway from Figures 4.11(a) and (b) is that *automatic* discovery of accurate, and physically and architecturally meaningful, models is possible using nonparametric regression.

We also compare our machine learning-based models against (1) parametric and (2) ORION 2.0 models. To develop models using parametric regression, we use the same data set as in our machine learning-based model generation process. The drawback of the parametric (linear, quadratic, etc.) regression methods is their limited accuracy since there are no procedures in the modeling methodology to help the modeler

[9] Our baseline VC router uses register-based buffers using flip-flops.

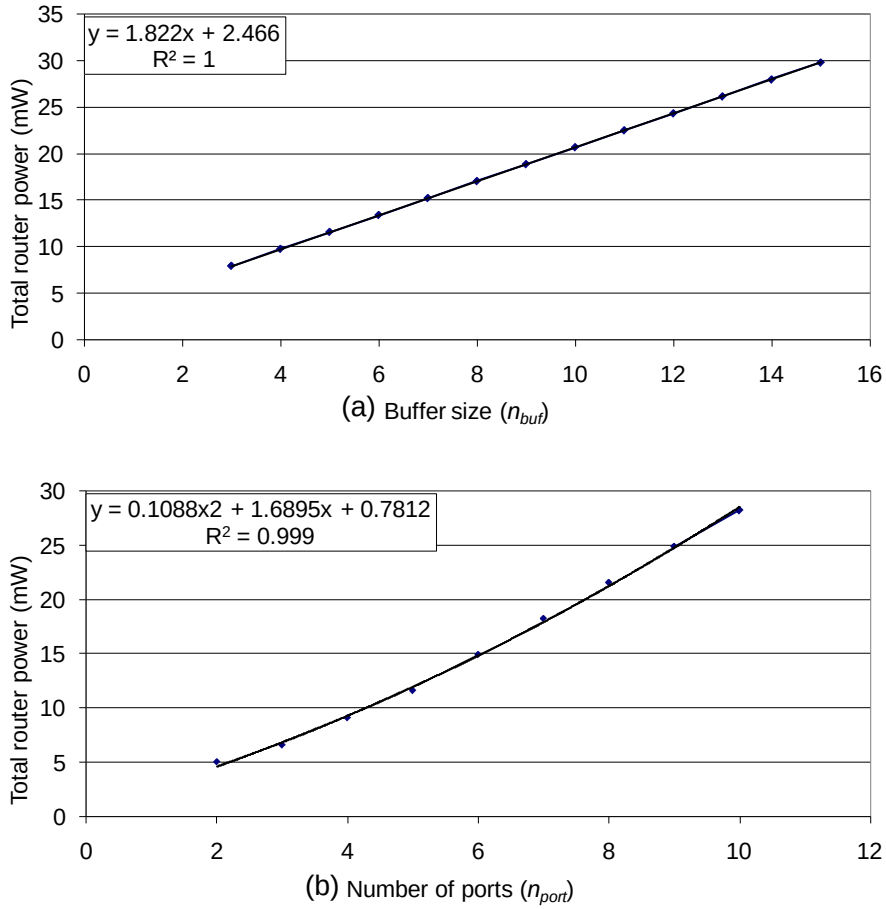

Figure 4.11: Total router power versus (a) buffer size and (b) number of ports.

understand the interactions between the individual parameters. Hence, to develop parametric models, comprehensive understanding of the underlying architecture and circuit implementation is needed. We also compare the machine learning-based models with ORION 2.0 using similar microarchitectural parameters. Figures 4.12(a) and (b) show the significant accuracy improvement of the new machine learning-based models relative to ORION 2.0, with respect to router implementation data. We believe that a key limitation of ORION 2.0 is that it does not capture the effects of the implementation flow. Modern IC implementation flows incorporate powerful logic synthesis and physical synthesis transformations such as logic restructuring or gate sizing to satisfy the power and performance constraints. The detailed impacts of such transformations are difficult to capture in static circuit templates such as that of ORION 2.0, as they depend

on implementation parameters such as process and library flavor, operating voltage, etc. Our results show that the machine learning-based models reduce the average estimation error by up to 86% and 89% versus parametric and ORION 2.0 models, respectively.

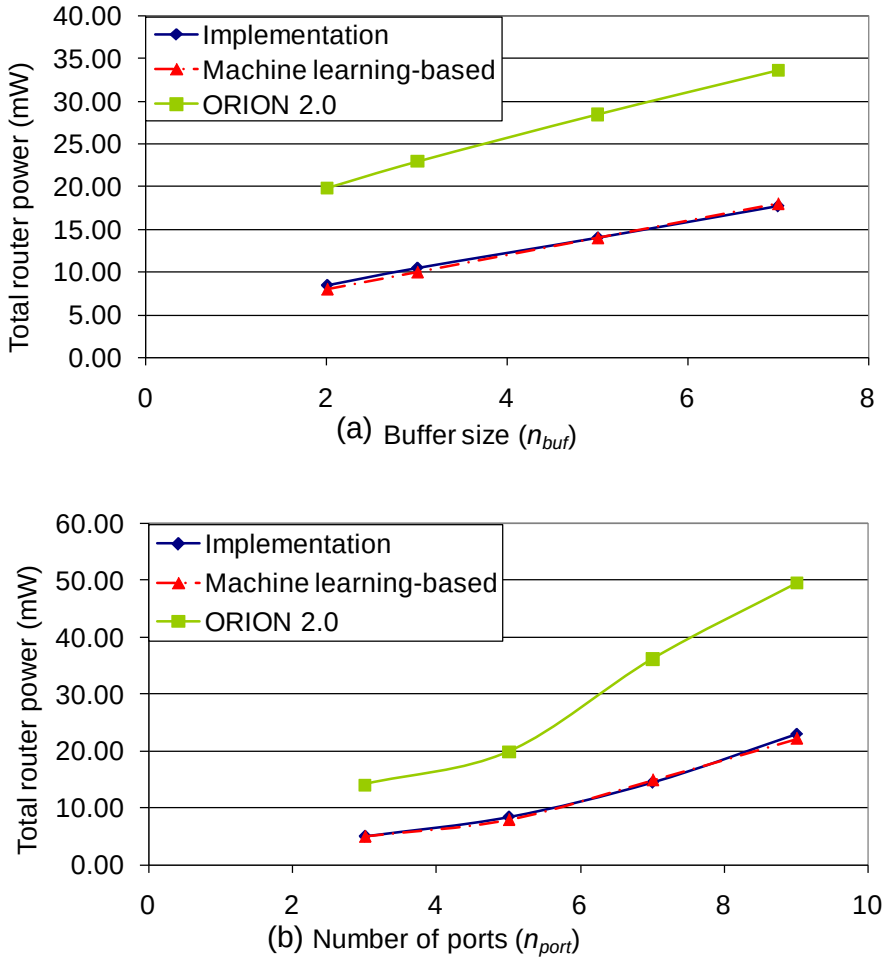

Figure 4.12: Comparison among implementation, the proposed machine learning-based models and ORION 2.0 showing total router power versus (a) buffer size and (b) number of ports.

Table 4.6 shows relative variable importance in our maximum implemented clock frequency model. *MARS 3.0* computes the variable importance based on the reduction in goodness of fit when the variable is removed. The right column in Table 4.6 shows the relative importance of variables, in order of decreasing percentage loss in *generalized*

cross-validation (GCV) [32].[10] From Table 4.6, we observe that target clock frequency f_{clk} is the main contributor to the maximum implemented frequency. Among the microarchitectural parameters, flitwidth does not affect maximum implemented frequency, since it only changes the bandwidth (i.e., number of simultaneous bits processed by the router). Buffer size and number of virtual channels do not noticeably affect the maximum implemented clock frequency, since buffer size only determines the amount of flit storage, and virtual channels only provide parallel, multiplexed paths to the crossbar switch (hence, they do not change the critical path and have no noticeable impact on maximum delay). However, the number of ports affects the maximum implemented frequency because it changes the crossbar switch interconnect grid, i.e., more ports result in a longer signal path within the crossbar switch.

Table 4.6: Relative variable importance for maximum implemented clock frequency modeling.

Parameter	Variable importance (%)
f_{clk}	100
V_{th}	38.85
n_{port}	22.78
n_{vc}	5.26
l_{buf}	4.95
$util$	2.09
AR	0.20

Experimental Results

In this subsection, we highlight the benefits of the proposed machine learning-based modeling methodology. First, the proposed methodology considers the interactions between different architectural and implementation parameters, and accurately captures their combined effect on the power and performance of the implemented on-

[10]Note that the relative variable importance depends on the existing set of implementation and microarchitectural parameters. If we remove any of the parameters, the relative variable importance and the associated rankings could change. GCV equation has been described in Equation (3.26) in Chapter 3.

chip routers. Our results show a close match (3.9% error on average) between the model estimates and layout data.

Second, the proposed methodology captures the impact of design optimization techniques (e.g., use of multi-V_{th} libraries). It is well known that HVT libraries can reduce leakage power, or LVT libraries can increase performance, etc. However, it is difficult to capture the impact of optimization techniques on design power and performance. As an example, the proposed methodology enables designers to quantify the impact on design power and performance of using triple-V_{th} libraries. Figure 4.13 shows maximum implemented frequency versus target clock frequency for single-V_{th} and triple-V_{th} libraries. Figure 4.14 shows router leakage power versus clock frequency for the two libraries. From such figures, we may estimate how much performance improvement or leakage power reduction can be obtained when we use triple-V_{th} libraries. Despite up to 60% difference between single-V_{th} and triple-V_{th} leakage power values, given the same implementation and microarchitectural parameters, the proposed leakage power model remains within 3.8% (on average) of the layout data. This confirms nonparametric regression techniques can accurately capture the impact of underlying optimization techniques in the implementation flow.

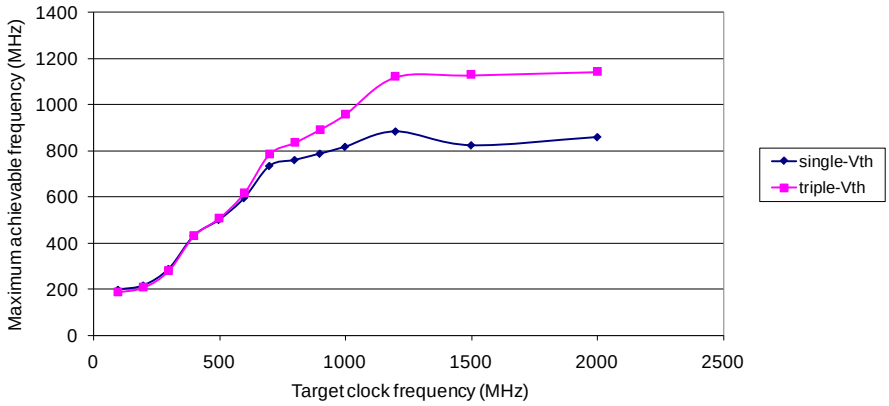

Figure 4.13: Maximum implemented clock frequency versus target clock frequency.

Third, the proposed models enable designers to optimize over different configurations and solutions using closed-form power and performance equations. As an example, we have performed a case-study design space exploration in which we show on-chip router energy efficiency with respect to architectural parameters. We assess the energy

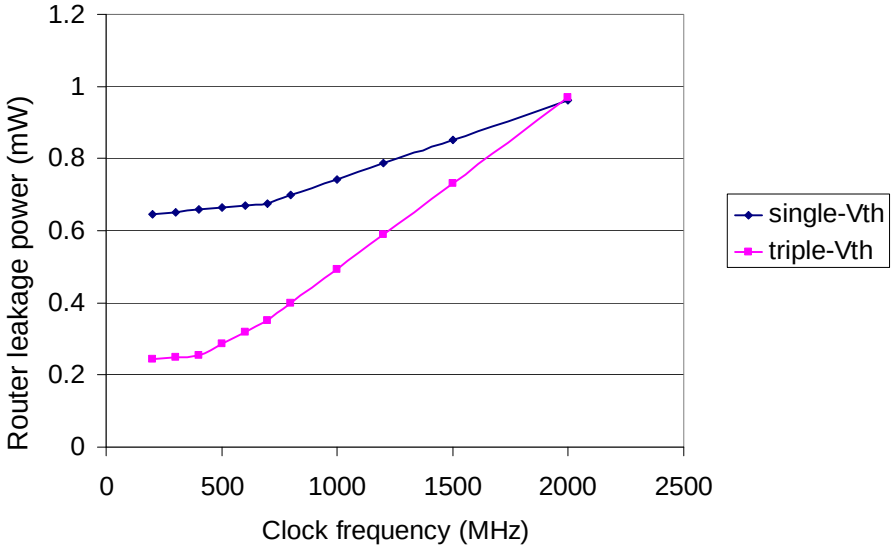

Figure 4.14: Router leakage power versus clock frequency.

efficiency of a single router at a fixed activity factor using the energy-per-bit metric, where the total number of simultaneous bits to a router is defined as $n_{port} \times n_{vc} \times fw$. In Figure 4.15, on the x-axis we show 27 data points corresponding to different combinations of flitwidth, number of ports, and number of virtual channels; on the y-axis we show the associated energy-per-bit values for each combination. The figure shows two sets of data: (1) the dotted line which represents the energy efficiency prediction (i.e., using the proposed closed-form models), and (2) the dashed line which represents the same metric derived from layout data. We observe that our model predictions closely match the layout data, again confirming that the proposed models can efficiently drive accurate design space explorations. In addition, Figure 4.15 shows the energy-per-bit as a function of n_{port}, n_{vc}, and fw. Note that buffer size does not change the total number of simultaneous bits to a router, but impacts the energy. Thus, for each combination of (n_{vc}, n_{port}, fw) we have a range of energy values corresponding to a range of different buffer sizes. From Figure 4.15 we can conclude that larger flitwidth and smaller number of ports increase router energy efficiency. On the other hand, buffer size can be an important knob in controlling the achievable energy-performance envelope, as it also directly impacts network latency.

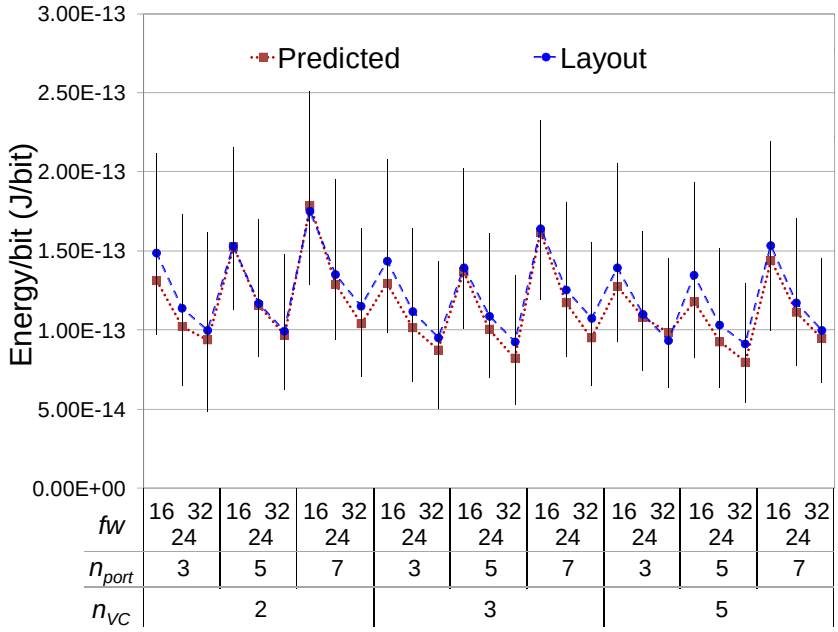

Figure 4.15: Router energy-per-bit versus choice of microarchitectural parameters.

4.3.5 Extensibility to Register File Modeling

Design decisions made at the architecture level have the largest impact on the power consumption of the final chip. Therefore, there is a need for architecture-level power, performance and area models for different building blocks of a design. Multi-ported register files (RFs) are commonly used in modern processors [21]. The large number of instances, and usually considerable size, of RFs make them an important modeling target with respect to power dissipation. Considerable effort has been spent on power models for RFs [21, 65, 122]. However, all previous models, in one way or other, are based on certain assumptions regarding the structure and design style of the underlying RFs; this limits the applicability of such models in efficient design space exploration.

In this subsection, we use machine learning-based nonparametric regression to model power (read and write power), performance (maximum clock frequency), and area of RFs with respect to their relevant microarchitectural parameters and clock frequency.

Modeling Methodology

We use a 65 nm industry memory generator to create register file instances. Using shell scripting, we call the memory generator to create the RF instances corresponding to given architectural parameters. We then report read and write power, maximum delay (i.e., maximum achievable clock frequency), and area of the RF instances. To illustrate the basic idea, consider the following baseline model generation flow.

- We begin with a parameterized RF configuration. We refer to this as a *configurable* RF specification, which will be used to generate the representative RF instances under different architectural and clock frequency parameters. The architectural parameters include (1) number of bits, (2) number of words, (3) number of ports, and (4) multiplexer width.
- Using a small subset of selected configurations for *training*, we run the memory generator and create the corresponding RF instances to obtain accurate power, performance (i.e., maximum clock frequency) and area for each instance.
- Finally, we apply machine learning-based nonparametric regression on the power, performance, and area training sets to derive the corresponding power, performance, and area models.

Table 4.7 shows the microarchitectural RF parameters and the values they take on in our study. Figure 4.16 shows an example write power model for a 65 nm dual-port register file.

Table 4.7: List of register file microarchitectural parameters used in our studies.

Parameter	Values
n_{bit}	{4, 8, 16, 32, 64, 128, 144}-bits
n_{word}	{8, 16, 32, 64, 128, 144, 256, 512, 1024}
n_{port}^{RF}	{1, 2}
mw	{1, 2, 4}

Basis Functions
b_1 = max(0, n_{bit} – 128); b_2 = max(0, 128 – n_{bit});
b_3 = max(0, n_{word} – 512); b_4 = max(0, 512 – n_{word});
b_5 = max(0, n_{bit} – 128)$\times b_4$; b_6 = max(0, 128 – n_{bit});
$\vdots$
b_{97} = max(0, n_{word} – 32) $\times b_1$; b_{99} = max(0, f_{clk} – 350) $\times b_{56}$;
Write Power Model
p_{write} = 0.066 – 0.0001 $\times b_1$ + 0.002 $\times b_2$ + 4.818e-5 $\times b_3$
+ 1.750e-6 $\times b_4$ + 1.272e-6 $\times b_5$ – 0.194 $\times b_7$ – 0.271 $\times b_8$
– 0.018 $\times b_9$ + 4.726e-6 $\times b_6$ + 0.0004 $\times b_7$ – 4.075e-10 $\times b_8$
+ 0.002 $\times b_9$ + 2.430e-5 $\times b_{10}$ – 1.342e-6 $\times b_{11}$ + 3.238e-6 $\times b_{12}$ $\cdots$
– 1.688e-8 $\times b_{91}$ + 1.334e-9 $\times b_{92}$ + 2.544e-7 $\times b_{93}$ – 2.670e-9 $\times b_{95}$
+ 1.342e-6 $\times b_{97}$ – 1.369e-7 $\times b_{99}$;

Figure 4.16: Write power model for a register file in 65 nm.

To generate our models, we randomly select 10% of the entire data set and test the models on the other 90% of the data. To show that the selection of the training set does not substantially affect model accuracy, we randomly select 10% of the entire data set five times and show the corresponding maximum and average error values (Table 4.8).

Table 4.8: Impact of training set randomization on write power model accuracy.

Experiments	Write power % error	
	max	avg
Exp 1	18.53	0.91
Exp 2	22.74	0.83
Exp 3	20.14	0.88
Exp 4	18.14	0.93
Exp 5	18.97	0.87

Our proposed register file models match well with estimates from the memory generator, with estimated read power, write power, area, and maximum delay (clock frequency) within 0.91%, 0.93%, 0.42%, and 0.31%, on average, of memory generator values. Finally, Figures 4.17(a) and (b) show scatter plots of our register file read and write power estimations against corresponding memory generator estimates, respectively.

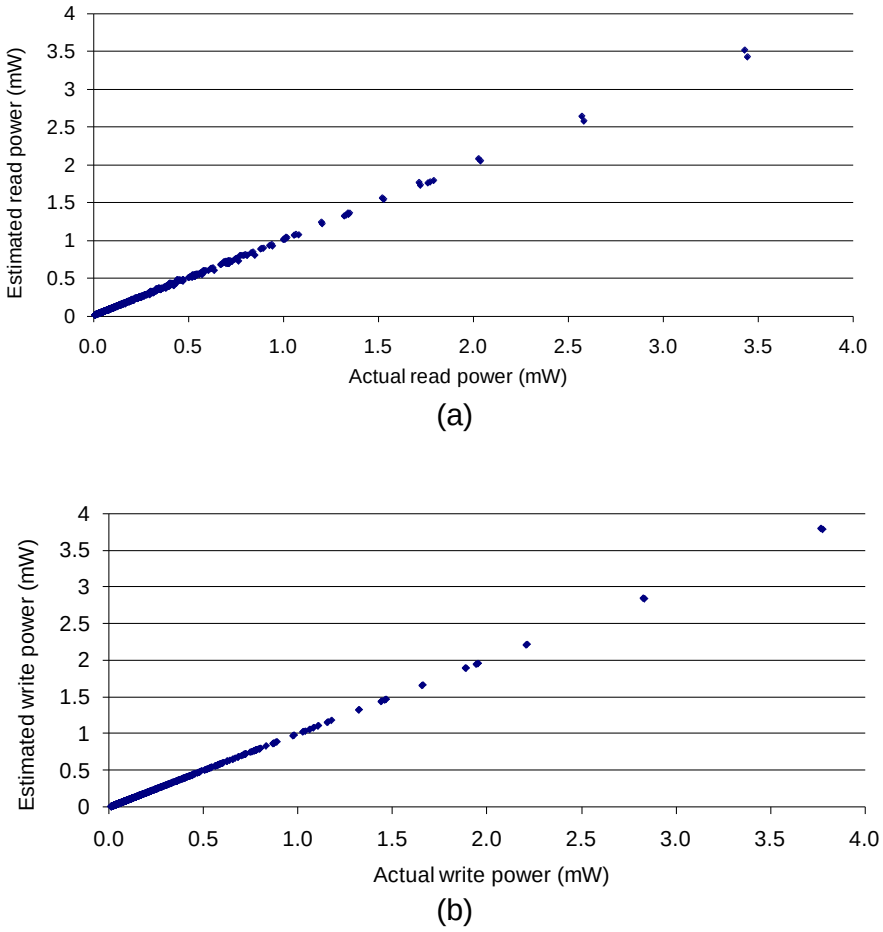

Figure 4.17: Comparison of (a) read power and (b) write power estimates against memory generator values.

4.3.6 3D NoC Power and Performance Modeling

Three-dimensional integrated circuits (3DIC) refers to an emerging technology which allows stacking of more than one silicon die. 3DIC potentially improves performance by replacing global wires with short vertical connections called "through-silicon vias" (TSVs). In this subsection, we extend our on-chip router and interconnect power and performance models to obtain power and performance estimations for NoCs in the 3DIC context. A mesh topology is the most widely used on-chip network topology in conventional 2D designs [57, 62]; we assume that on-chip routers are implemented in a single layer (i.e., 2DIC) and then stacked in the third dimension. On-chip routers in a 2D mesh network have five ports: four ports to communicate with the four neighboring routers, and one port to communicate with the corresponding processing element. In a 3D NoC design with a mesh topology, the router connects to two additional neighboring routers on the adjacent layers, which increases the number of router ports to seven. In our assumed 3D NoC design, k_x, k_y and k_z denote the number of routers in the x, y and z directions, respectively. Hence, the total number of on-chip routers in the network is $k_x \times\ k_y \times\ k_z$. We develop analytical performance and power models for a 3D NoC using the on-chip router models proposed in Section 4.3 and the interconnect models proposed in Chapter 3.

Performance Modeling

The *zero-load* network latency is widely used as a performance metric in conventional on-chip networks [44, 79]. The zero-load latency of a network is the latency where only one packet traverses the network. Even though the zero-load model does not consider contention among packets, it can be used to assess the impact of topology on performance of a network [86]. The network latency, $T_{network}$, is given as

$$T_{network} = hop_{avg} \times t_{router} + \frac{size_{packet}}{bandwidth} + t_{int} \tag{4.1}$$

The first term in Equation (4.1), $hop_{avg} \times t_{router}$, represents the *head latency* where hop_{avg} and t_{router} denote average hop count and router delay, respectively. Average hop count for a k-ary 1-mesh under uniform traffic and dimension-ordered routing

is given in Equation (4.2) [44]. We also assume that each node can send traffic to itself.

$$hop_{avg} = \frac{k^2 - 1}{3k} \tag{4.2}$$

We can extend the above equation considering the effects on average hop count of additional nodes in the y- and z-directions (Equation (4.3)).

$$hop_{avg} = \frac{k_x^2 - 1}{3k_x} + \frac{k_y^2 - 1}{3k_y} + \frac{k_z^2 - 1}{3k_z} \tag{4.3}$$

Router delay, t_{router}, is derived using layout data via the use of nonparametric regression methods as described in Section 4.3.

The second term in Equation (4.1) represents *serialization latency*, which is the time required for a packet of size $size_{packet}$ to cross a channel with bandwidth $bandwidth = fw \times f_{clk}$, where fw and f_{clk} denote the flitwidth and the clock frequency, respectively. Finally, the third term in Equation (4.1), t_{int}, represents the time to traverse the average distance between a source and a destination, and is due to wires in the x- and y-direction, and TSVs in the z-direction. To appropriately compute t_{int}, we first calculate the average distance in the x- and y-directions, d_{xy}, and in the z-direction, d_z, as shown in Equations (4.4) and (4.5), respectively.

$$d_{xy} = (\frac{k_x^2 - 1}{3k_x} + \frac{k_y^2 - 1}{3k_y}) \times l_{int} \tag{4.4}$$

$$d_z = \frac{k_z^2 - 1}{3k_z} \times h_{TSV} \tag{4.5}$$

In Equations (4.4) and (4.5), l_{int} and h_{TSV} respectively denote the interconnect length between two adjacent nodes and the TSV height. Finally, we compute $t_{int} = t_w + t_{TSV}$, where t_w is the delay due to horizontal wires, and t_{TSV} is the delay due to TSVs. We leverage accurate interconnect models described in Chapter 3 to compute t_w, and use $t_{TSV} = r_{TSV} \times c_{TSV}$, where r_{TSV} and c_{TSV} respectively denote TSV resistance and capacitance for a given height and diameter.

Power Modeling

To model power, we use the on-chip router power models proposed in Section 4.3 and the interconnect models proposed in Chapter 3. As noted above, individual routers in a 3D NoC have two additional ports to communicate with the neighbors on the adjacent layers. Dynamic power is primarily due to charging and discharging of capacitive loads (wire, input capacitance of the next-stage repeater and TSVs), and leakage power is due to the repeaters. Hence, total network power due to on-chip routers, interconnects and TSVs is given as follows.

$$p_{network} = (k_x \times k_y \times k_z) \times p_{router} + (\frac{k_x^2 - 1}{3k_x} + \frac{k_y^2 - 1}{3k_y}) \times p_{int} + \frac{k_z^2 - 1}{3k_z} \times p_{TSV} \quad (4.6)$$

2D NoC versus 3D NoC Comparisons

Using the third dimension for NoC implementation decreases the average number of hops that packets must traverse. Both the average number of hops on the same layer, hop_{2D}, and the average number of hops in the third dimension, hop_{3D}, are reduced. The distribution of the nodes k_x, k_y and k_z that yields minimum total number of hops is not always the same as the distribution that minimizes the number of intralayer hops [86]. This scenario occurs especially for small and medium-sized networks, while for large networks, the distribution of k_x, k_y and k_z that minimizes the number of hops also minimizes the number of intralayer hops [86]. Table 4.9 shows representative TSV dimensions and the corresponding resistance and capacitance values in 65 nm.

Table 4.9: TSV diameter, height, pitch and the corresponding resistance and capacitance values in 65 nm.

Parameter	Value
TSV diameter	6 μm
TSV height	50 μm
TSV pitch	12 μm
TSV resistance	40 mΩ
TSV capacitance	85 fF

Figures 4.18 and 4.19 compare 2D NoC and 3D NoC with respect to network latency and network power, respectively. A decrease in latency of 11% (16%) is observed when the total number of nodes is 128 (512). We assume that each tile has an area of 0.53 mm^2 (cf. the Intel Teraflops chip [57]) which results in $l_{int} = \sqrt{0.53}$ mm^2. The node distribution that results in the lowest latency varies with network size; it is a function of the reduction in number of hops from use of the third dimension. For small and medium networks, the decrease in the number of hops is small and cannot compensate the increase in the routing delay due to the increase in number of ports of a router in 3D NoC. On the other hand, as the global interconnect length increases, even a slight decrease in the number of hops significantly decreases the overall delay.

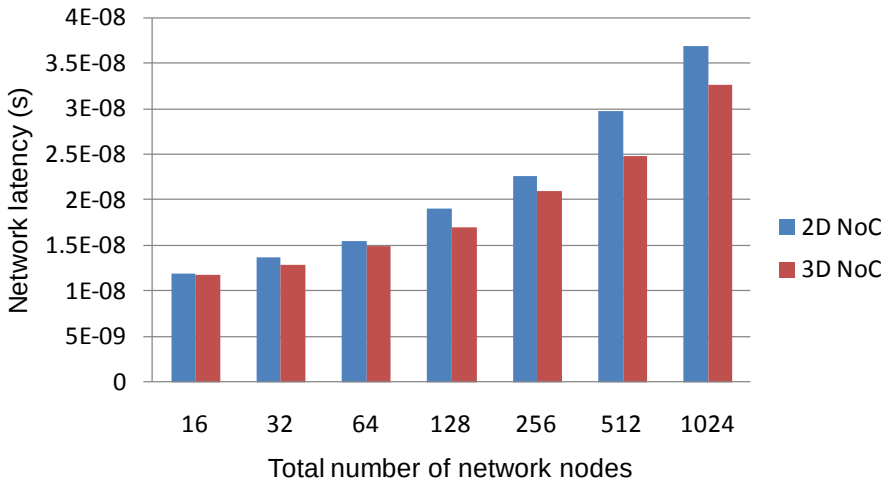

Figure 4.18: Network latency with respect to total number of nodes in the network for 2D NoC and 3D NoC.

Similar to network latency, power consumption is decreased by reducing the number of hops. The power impact from the increase in number of ports is less than the impact on latency. Hence, 3D NoC can reduce power even in small networks. Figure 4.19 shows 15% (19%) decrease in power consumption per bit when the total number of nodes is 128 (512).

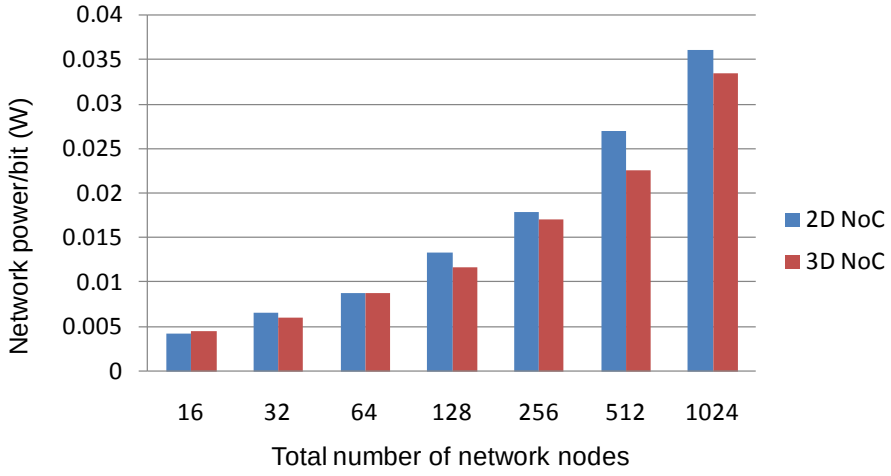

Figure 4.19: Network power with respect to total number of nodes in the network for 2D NoC and 3D NoC.

4.4 Conclusions

Accurate estimation of power and area of on-chip routers in early phases of the design process is required for effective NoC design space exploration. ORION 1.0 [114], a popular template-based power model for on-chip routers, is inaccurate for current and future technologies, and leads to poor design choices. We therefore develop ORION 2.0, which provides more accurate on-chip router power and area models that are easily usable by system-level designers. We also develop a reproducible methodology for extraction of inputs to our models from different reliable sources (e.g., industry technology files), so that ORION 2.0 can be easily and continuously updated in the future. Finally, we validate our new models with respect to different microarchitectural and technology parameters, synthesis of router RTLs, and two recent Intel chips. By maintaining the user interfaces of the original ORION 1.0 while substantially improving accuracy and fidelity, we see ORION 2.0 making a significant impact on future NoC research and design.

We also note that existing on-chip router models (e.g., ORION 2.0 [63], Xpipes [41], etc.) are based on specific architectures and circuit implementations. Hence, they cannot guarantee maximum accuracy within an architecture-specific computer-aided design flow. To address this problem, we propose an efficient on-chip router power and

area modeling methodology in which the underlying architecture and circuit implementation are decoupled from the modeling effort. We use machine learning-based non-parametric regression methods to accurately develop router power, performance and area models from final implementation data. We achieve a close match (3.9% error on average) between our machine learning-based models and actual layout data. We also apply our on-chip router and interconnect models to develop simple network power and latency estimates for 3D NoCs.

4.5 Acknowledgments

This chapter is in part a reprint of:

- Andrew B. Kahng, Bin Li, Li-Shiuan and **Kambiz Samadi**, "ORION 2.0: A Power-Area Simulator for Interconnection Networks", to appear in *IEEE Transactions on Very Large Scale Integration Systems*.
- Kwangok Jeong, Andrew B. Kahng, Bill Lin and **Kambiz Samadi**, "Accurate Machine Learning-Based On-Chip Router Modeling", *IEEE Embedded Systems Letters* 2(3) (2010), pp. 62–66.
- Andrew B. Kahng, Bin Li, Li-Shiuan Peh and **Kambiz Samadi**, "ORION 2.0: A Fast and Accurate NoC Power and Area Model for Early-Stage Design Space Exploration," *Proc. Design, Automation and Test in Europe*, 2009, pp. 423–428.

I would like to thank my coauthors Kwangok Jeong, Dr. Bin Li, Prof. Bill Lin, Prof. Li-Shiuan Peh, and Prof. Andrew B. Kahng. I would also like to thank Xiang Hu for providing 65 nm TSV values.

Chapter 5

Trace-Driven Optimization of Network-on-Chip Configurations

5.1 Introduction

As noted in Chapter 1, significant research effort has been spent to reduce router latency through modified router architectures and designs [48]. This chapter addresses the application-specific MPSoC context, wherein the configuration of each router in the network can be *non-uniformly* optimized to the traffic characteristics of the particular application. Though the problem of NoC configuration for application-specific MPSoCs is not new, prior approaches [37, 60, 61] have been *average-rate driven* in that the traffic characteristics have been modeled with *average data rates*. Recall from Figure 1.4 that average-rate models are poor representations of actual traffic characteristics for real applications. Our premise is that average-rate driven approaches may be misled by average traffic characteristics, resulting in poor design choices that are not well-matched to the actual traffic characteristics.

Among all router resources, input buffers consume a significant (e.g., 30% [57, 67, 74, 63]) portion of the total communication power. Hence, minimizing the number of buffers is important for reduction of router power consumption. Together, optimization of buffer size and VC allocation is of critical importance in maximizing performance and minimizing power consumption.

For buffer sizing, a number of methods have been proposed in the literature.

- Chandra *et al.* [37] propose a sizing algorithm based on bursty transmission rates of packets. A drawback of this work is its use of synthetic traffic models, which can potentially impact the relevance of the obtained solutions.

- Manolache *et al.* [73] propose a traffic shaping methodology in which packets are delayed at the source to avoid contention between different flows, so as to reduce the total amount of buffer usage along intermediate nodes.

- Hu *et al.* [60] propose a probabilistic approach for the sizing of input buffers along the intermediate nodes of a network. Their main goal is to minimize the average packet latency of all packet transmissions in the network while remaining within an overall buffer area budget. Their method assumes packets as atomic units of storage (i.e., store-and-forward). On the other hand, modern routers use flit-level flow control to achieve better latency and area.

For VC allocation, several approaches have also been proposed.

- Huang *et al.* [61] propose a queueing-based algorithm for VC allocation. Their focus is on determining the number of VCs to allocate to each link, with the assumption that the network topology, the mapping of tasks to the NoC, and the deterministic routing algorithm are all given. Their greedy algorithm increases the number of VCs allocated to a given link only if the addition reduces the average packet latency.

- Al Faruque *et al.* [22] propose a two-step VC allocation approach that seeks to minimize the number of VCs required to achieve a certain quality of service level. Their VC allocation approach is carried out during the task mapping stage. By assuming a Poisson packet arrival process, they try to estimate the size of each VC for each output port. A shortcoming of this approach is its reliance on Markovian assumptions for multimedia applications. These assumptions have been shown to be unreliable by other works (e.g., [112]).

The above buffer and VC allocation approaches are *static*, i.e., they are decided at design time. Other methods have been proposed to dynamically allocate buffers and VCs at runtime.

- The dynamically allocated multi-queue approach [106] uses linked lists to allocate VCs to each port. To update the logical pointer to the free list, a 3-cycle delay is incurred at every flit arrival/departure, making this method unsuitable for high-performance applications.

- Dally *et al.* [44] propose a similar approach in which buffers are built using linked lists. To control buffer allocation, the authors of [44] add to the state of each VC buffer a count register that keeps track of the number of flits in that buffer. An additional count register keeps track of the number of cells remaining on the free list. Using these counts, the authors of [44] propose a number of different buffer allocation policies.

- The fully-connected circular buffer approach [81] uses registers to selectively shift flits within buffers. This solution requires a large $P^2 \times P$ crossbar instead of a conventional $P \times P$ crossbar, where P is the number of ports. It also requires existing flits to be shifted when new flits arrive. Hence, this approach has significant latency and power overhead.

- Finally, Nicopolous *et al.* [82] propose a dynamic VC allocation approach, called ViChaR, in which the number of VCs and the depth of buffers per VC are dynamically adjusted based on the traffic load. Since there can be as many VCs as there are flit buffers, control logic becomes complicated.

In summary, runtime dynamic allocation of buffers seems more desirable for general-purpose and reconfigurable design platforms that execute different workloads. However, design-time allocation of VCs appears more desirable for application-specific NoCs that are intended to implement a specific application or a limited class of applications.

In this chapter, we propose a *trace-driven* approach that uses actual application traffic traces to drive the optimization of NoC configurations. To illustrate the benefit

of a trace-driven approach, we specifically consider the problem of application-specific VC allocation. The contributions of this chapter are listed below.

- We propose simple yet effective VC allocation heuristics within a trace-driven optimization paradigm that directly incorporates actual application traffic behavior and workloads into the optimization process.
- We compare our proposed approaches with an existing average-rate driven VC allocation method [61]. In this comparison, we achieve up to 35% reduction in the number of VCs for a given average packet latency, which demonstrates the benefits of a trace-driven approach over average-rate driven approaches.
- We improve scalability of our approach to larger networks via a new proposed metric, called *significant VC failure*, which efficiently captures runtime VC contentions.
- We also propose new metaheuristics that achieve O($|L|$) speedup with no degradation in the quality of results, where $|L|$ is the number of links in the network.
- Finally, we evaluate our proposed heuristics and metaheuristics on a set of real applications. In particular, we evaluate our methods on the PARSEC benchmark suite [31], which contains multi-threaded programs that are representative of emerging workloads.

The remainder of this chapter is organized as follows. Section 5.2 describes our trace-driven VC allocation problem formulation. Subsections 5.2.1 and 5.2.2 describe two simple yet effective greedy heuristics. Subsections 5.2.4 and 5.2.5 present two new metrics which allow us to speed up our method by O($|L|$). In Section 5.3, we propose two efficient metaheuristics based on Subsections 5.2.4 and 5.2.5, and show that they can provide significant runtime improvement with no loss of solution quality. Section 5.4 describes our experimental setup and testcases, and presents experimental results. Finally, Section 5.5 concludes the chapter.

5.2 Trace-Driven VC Allocation Problem Formulation

In a typical design of virtual channel routers, a fixed amount of hardware resources (i.e., queues) is set aside to implement the VC buffers. As noted above, we wish to satisfy given performance criteria while minimizing the number of VCs. When the application is known, we state the *trace-driven VC allocation problem* as follows.

Given:

- Application communication trace, App_{trace}
- Network topology, $T(P, L)$, where P is the set of processing elements and L is the set of physical links
- Deterministic routing algorithm, R
- Target latency, t_{target}

Determine:

- A mapping $\mathbf{q}_{vc}$ from the set of links L to a set of positive integers, i.e., $\mathbf{q}_{vc} : L \rightarrow Z^{+}$, where for any $l \in L$, $q_{vc}(l)$ gives the number of VCs associated with link l, such that $\sum_{l \in L} q_{vc}(l)$ is minimized while average packet latency (APL) using routing algorithm R, $t(\mathbf{q}_{vc}, R)$, is within a target latency constraint t_{target}

Objective:

- Minimize $\sum_{l \in L} q_{vc}(l)$

Subject to:

- $t(\mathbf{q}_{vc}, R) \leq t_{target}$

In the next two subsections, we propose two greedy heuristics whose performance will be discussed in Subsection 5.2.3.

5.2.1 Greedy Addition VC Allocation

Our first VC allocation heuristic is shown in Figure 5.1. The algorithm initializes every network link, l, with one VC (Lines 1-3); this is equivalent to wormhole routing.

Hence, the total number of VCs in the network, n_{vc}, is initialized to the total number of links, $|L|$ (Line 5). Then, the algorithm proceeds in a greedy fashion: in each iteration, the performance of every one of $|L|$ possible perturbations of the current VC configuration, $\mathbf{q}_{vc}^{current}$, are evaluated simultaneously. Each perturbation consists of adding exactly one VC to one link (Line 9). The average packet latencies of all perturbed VC configurations are examined, and the configuration with the smallest average packet latency, $\mathbf{q}_{vc}^{best}$, is chosen (Line 12). Subsequently, $\mathbf{q}_{vc}^{best}$ is set as the starting configuration of the next iteration. The algorithm stops if either the total number of allocated VCs exceeds the VC budget, $budget_{vc}$, in which case the VC budget needs to be increased to achieve the target latency, or a configuration with better performance than the target average packet latency, t_{target}, is found. Runtime analysis of our proposed greedy addition heuristic is given in Subsection 5.2.3.

Despite the effectiveness of the addition approach, it has an inherent disadvantage, namely, it bases its decision at each iteration on the impact (on average packet latency) of adding only a single VC. To demonstrate the drawback of this approach, consider the example of Figure 5.2, where two traffic flows F_1 and F_2 share links $A \rightarrow B$ and $B \rightarrow C$, both of which initially have only one VC (A, B, and C are network nodes). F_1 turns west and F_2 turns east at node C. In this case, adding a VC to either link $A \rightarrow B$ or link $B \rightarrow C$ may not have a significant impact on average packet latency of flows F_1 and F_2 because the other link with just one VC becomes the bottleneck. On the other hand, if VCs are added to both links $A \rightarrow B$ and $B \rightarrow C$, the average packet latency of the flows may be significantly reduced. This is because if one of the two flows (F_1 or F_2) is blocked at node C due to congestion, the other flow can still use the shared links $A \rightarrow B$ and $B \rightarrow C$. The greedy VC addition approach may fail to realize the benefits of these combined additions and not pick either of the links as candidates for VC allocation at a given iteration. To overcome this drawback, we propose another greedy VC allocation heuristic based on *deletion*, instead of addition, of VCs.

5.2.2 Greedy Deletion VC Allocation

Our greedy VC deletion algorithm is shown in Figure 5.3. The approach is structurally similar to greedy addition except that we delete a VC, instead of adding one,

Algorithm: Greedy Addition

Input: Application communication trace, App_{trace}; Network topology, $T(P, L)$; Deterministic routing algorithm, R; Target latency, t_{target}

Output: Vector $\mathbf{q}_{vc}$, which contains the number of VCs associated with each link $l \in L$

1. **for** i = 1 to $|L|$
2. $q_{vc}^{current}(l) = 1$;
3. **end for**
4. $\mathbf{q}_{vc}^{best} = \mathbf{q}_{vc}^{current}$;
5. $n_{vc} = |L|$;
6. **while** $(n_{vc} < budget_{vc})$
7. **for** l = 1 to $|L|$
8. $\mathbf{q}_{vc}^{new} = \mathbf{q}_{vc}^{current}$;
9. $q_{vc}^{new}(l) = q_{vc}^{current}(l) + 1$;
10. **run** trace simulation on $\mathbf{q}_{vc}^{new}$ and **record** $t(\mathbf{q}_{vc}^{new}, R)$;
11. **end for**
12. **find** $\mathbf{q}_{vc}^{best}$;
13. $\mathbf{q}_{vc}^{current} = \mathbf{q}_{vc}^{best}$;
14. $\mathbf{q}_{vc} = \mathbf{q}_{vc}^{best}$;
15. **if** $(t(\mathbf{q}_{vc}^{best}, R) \leq t_{target})$
16. break;
17. **end if**
18. n_{vc}++;
19. **end while**

Figure 5.1: Greedy addition heuristic.

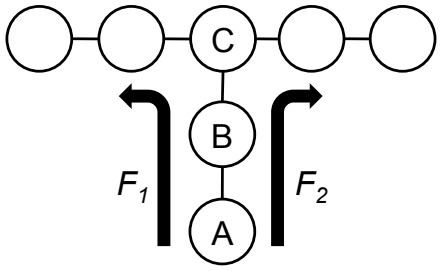

Figure 5.2: An example illustrating the drawback of greedy addition heuristic.

at each iteration. The algorithm starts with a given initial VC configuration, $\mathbf{q}_{vc}^{initial}$ (Line 1).[1] At the beginning of each iteration, the current VC configuration vector, $\mathbf{q}_{vc}^{current}$, is saved into a new vector, $\mathbf{q}_{vc}^{new}$ (Line 6). Next, single VC-deletion perturbations of the current VC configuration are generated by removing a VC from each link in the network that has more than one VC (Lines 7-8). The impact of each VC removal on the average packet latency is assessed and the one with minimum latency is selected for the next iteration (Lines 12-13).

The stopping condition for the algorithm is qualitatively different from that of the addition approach. We *generally* expect the average packet latency to increase at every iteration as VCs are removed. However, on rare occasions during the algorithm execution we may encounter a configuration with fewer VCs that has comparable or slightly lower packet latency than one with a higher number of VCs. This may be because the link for which a VC was removed was not the bottleneck link responsible for increased average packet latency. Hence, instead of stopping the algorithm as soon as the average packet latency exceeds t_{target}, it is better to continue exploring configurations even after exceeding the target latency value, and then return the minimum VC configuration that satisfies the latency constraint. The algorithm automatically stops once a wormhole configuration is reached.

The greedy deletion approach can overcome the drawback of greedy addition depicted in Figure 5.2. In the figure, if the initial configuration had two VCs for links $A \rightarrow B$ and $B \rightarrow C$, deletion of a VC on either of these links would expose the link as a bottleneck. At a given iteration, a VC is deleted from one of these links only if the

[1]Here, without the loss of generality, we assume that the algorithm starts with a given uniform VC configuration.

impact of the deletion on packet latency is less than the impact of deleting VCs from any other link in the network.

Figure 5.4 shows the average packet latency of each intermediate configuration using the greedy VC addition and deletion approaches for two different traces, *fluidanimate* and *vips* from the PARSEC benchmark suite [31]. The results presented are for a mesh network with 16 nodes and 64 links (details of the setup are explained in Section 5.4.1). In general, the average packet latency decreases as VCs are increased in the addition algorithm, and decreases as VCs are removed in the deletion algorithm. However, the change in packet latency is much smoother in the case of deletion of VCs, compared to addition. This is because adding a VC at a single link may not have a significant impact on average packet latency unless the added VC relieves congestion along the entire path of a traffic flow, as illustrated in the Figure 5.2 example. This explains the stepwise nature of the curve for VC addition, which shows periods of little improvement followed by steep descents. The intermediate solutions in the deletion approach are of a slightly higher quality because the deletion heuristic is better at detecting bottleneck links.

Figures 5.5 and 5.6 highlight the potential benefits of one of our VC allocation algorithms (greedy deletion) on two of the traffic benchmarks, *fluidanimate* and *vips*. In our experiments, we set a VC budget of 256 VCs or four VCs per link. We observe that a uniform configuration with four VCs per port has the best average packet latency among all uniform configurations within the VC budget. With our greedy deletion approach, we are able to achieve the same latency as that of a uniform configuration with four VCs per port, with 50% and 42% reduction in the number of VCs for the *fluidanimate* and the *vips* benchmarks, respectively.

Measuring the reduction in the number of VCs to achieve a given performance criteria shows the potential power and area benefits of trace-driven VC allocation. Another outlook would be to quantify the performance benefits while keeping the number of VCs fixed. With respect to this metric, we observe in Figures 5.5 and 5.6 that with a constraint of 128 VCs, the average packet latency of greedy deletion is better than that of a uniform-2VC configuration by 32% for the *fluidanimate* benchmark and by 74% for the *vips* benchmark. So, trace-driven non-uniform VC allocation can potentially be used

Algorithm: Greedy Deletion

Input: Application communication trace, App_{trace}; Network topology, $T(P, L)$; Deterministic routing algorithm, R; Target latency, t_{target}; Initial VC configuration, $\mathbf{q}_{vc}^{initial}$

Output: Vector $\mathbf{q}_{vc}$, which contains the number of VCs associated with each link $l \in L$

1. $\mathbf{q}_{vc}^{current} = \mathbf{q}_{vc}^{initial}$;
2. $\mathbf{q}_{vc}^{best} = \mathbf{q}_{vc}^{current}$;
3. n_{vc} = $\sum_{l \in L} \mathbf{q}_{vc}^{current}(l)$;
4. **while** $(n_{vc} \geq budget_{vc})$
5. **for** l = 1 to $|L|$
6. $\mathbf{q}_{vc}^{new} = \mathbf{q}_{vc}^{current}$;
7. **if** $(q_{vc}^{current} > 1)$
8. $q_{vc}^{new}(l) = q_{vc}^{current}(l) - 1$;
9. **run** trace simulation on $\mathbf{q}_{vc}^{new}$ and **record** $t(\mathbf{q}_{vc}^{new}, R)$;
10. **end if**
11. **end for**
12. **find** $\mathbf{q}_{vc}^{best}$;
13. $\mathbf{q}_{vc}^{current} = \mathbf{q}_{vc}^{best}$;
14. $\mathbf{q}_{vc} = \mathbf{q}_{vc}^{best}$;
15. **if** $(t(\mathbf{q}_{vc}^{best}, R) \leq t_{target})$
16. break;
17. **end if**
18. $n_{vc}--$;
19. **end while**

Figure 5.3: Greedy deletion heuristic.

to either reduce power within a given performance constraint or improve performance within a given power constraint.

Figures 5.7 and 5.8 compare our algorithm outputs to the solutions obtained using an average-rate driven VC allocation technique [61]. Our addition and deletion algorithms outperform the average-rate driven approach by 19% and 20%, respectively,

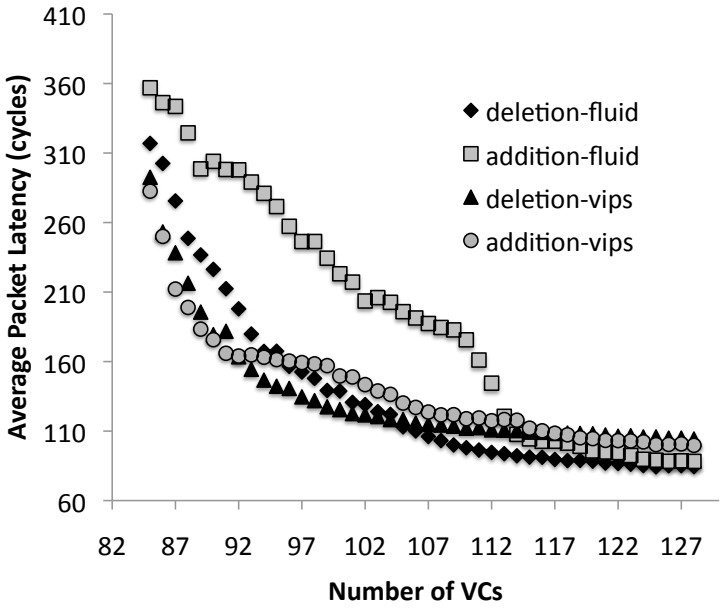

Figure 5.4: Performance of addition and deletion VC allocation heuristics for the *fluidanimate* and *vips* applications.

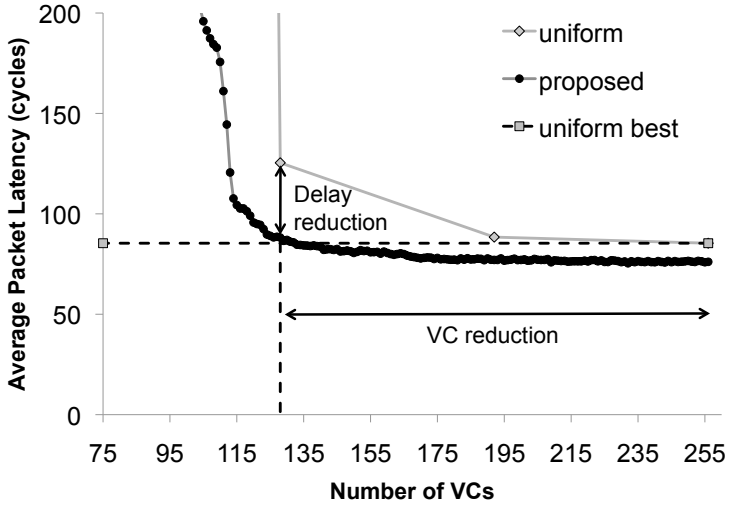

Figure 5.5: Average packet latency and VC reductions for the *fluidanimate* application.

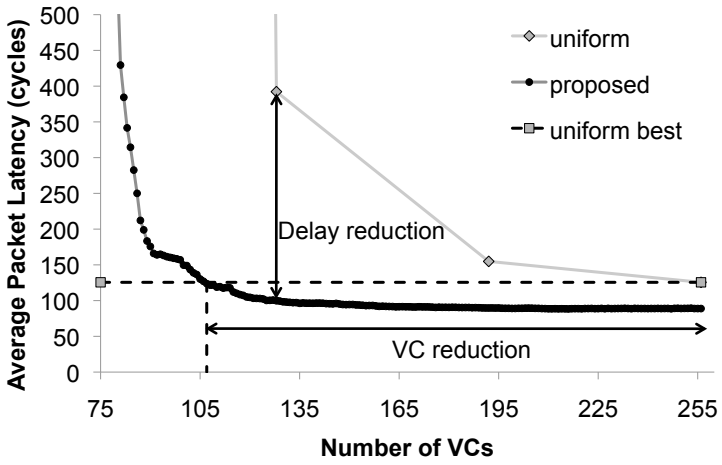

Figure 5.6: Average packet latency and VC reductions for the *vips* application.

over all traces when the target latency is set to be the latency of uniform-2VC and by 25% and 35%, respectively, when the target latency is the latency for uniform-3VC. The average-rate driven approach reduces the number of VCs over the uniform configurations for only two of the seven benchmarks. We believe that this is due to the fact that the average-rate characteristics of an application trace are not a very accurate representation of the actual traffic, which is bursty in nature for most traces (recall Figure 1.4). Our trace-driven techniques accurately comprehend the effects of these traffic bursts and obtain significantly better solutions.

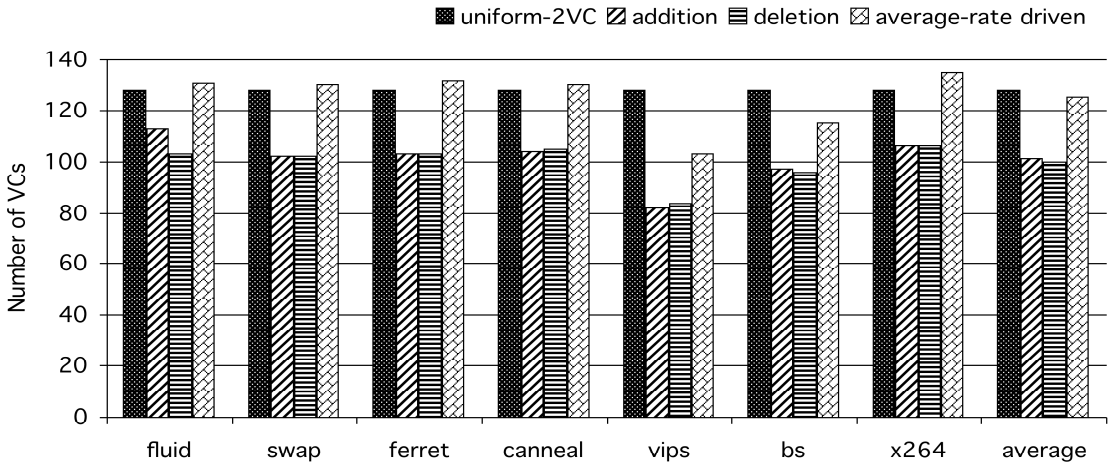

Figure 5.7: Performance of addition and deletion VC allocation methods versus the uniform-2VC configuration.

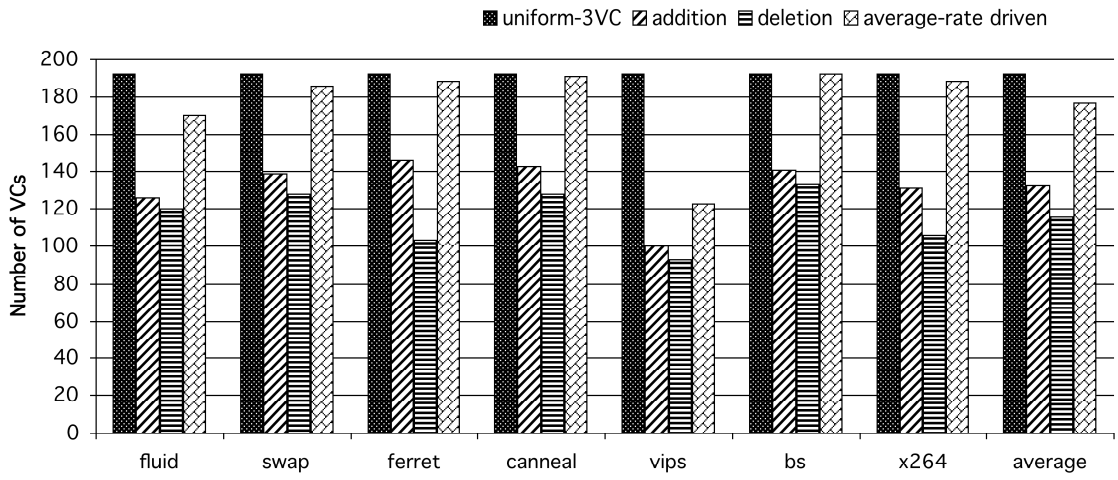

Figure 5.8: Performance of addition and deletion VC allocation methods versus the uniform-3VC configuration.

5.2.3 Runtime Analysis of Greedy Heuristics

Let m be the number of VCs added to (by greedy addition) or deleted from (by greedy deletion) an initial VC configuration. The runtime of the two proposed heuristics, t_{heur}, for any given input trace is

$$t_{heur} = m \times |L| \times t(\textit{trace}) \tag{5.1}$$

where $t(\textit{trace})$ is the average time to run trace simulation on all VC configurations explored by the algorithm. The above expression for runtime holds if the performance of each perturbation of the current VC configuration is evaluated sequentially. However, the two heuristics easily permit evaluation of all the perturbations in parallel. This results in an improved runtime of

$$t_{heur} = m \times t(\textit{trace})_{max} \tag{5.2}$$

Here, $t(\textit{trace})_{max}$ represents the average of the maximum runtimes of trace simulation at each iteration, where the maximum is taken over the runtimes of all perturbations in the iteration. We observe that this improved runtime scaling for our greedy heuristics requires O($|L|$) computing nodes to process the VC perturbations in parallel; this becomes impractical as $|L|$ increases. To enhance the scalability of our approach to larger

networks, we propose new metrics that can more efficiently capture the runtime VC contentions as explained in the following subsections.

5.2.4 SVCF-Driven VC Allocation

We propose to apply the concept of *significant VC failure* (SVCF) to implicitly capture the impact of virtual channels on average packet latency. A significant VC failure occurs when a new packet cannot acquire a virtual channel because all virtual channels that use the same output link are already held by packets that are "blocked" from proceeding further by downstream contentions. In this scenario, the output link unnecessarily remains idle until a packet that already holds a virtual channel can proceed.

To see what we mean by a significant VC failure, consider the example shown in Figure 5.9. Suppose we have three packets with the following ($source$, $destination$): (A, F), (B, D), and (E, D), with all three packets 10-flits in size. Packet (E, D) is injected at time t = 0, and Packets (A, F) and (B, D) are injected at time t = 1. We assume in this example that links only have a single VC (i.e., wormhole configuration) with 10 flits per VC. Link 2 will carry two packets from (A, F) and (B, D), and Link 3 will also carry two packets from (B, D) and (E, D). We observe that Packet (A, F) is "blocked" from proceeding because Link 2 is held by Packet (B, D) which itself is "blocked" from proceeding since Packet (E, D) has already held Link 3. Note that, as long as Packet (B, D) is blocked, Packet (A, F) is also blocked even though it is heading to a different destination. Only when all 10 flits of Packet (E, D) have traversed Link 3, then Packet (B, D) can proceed; however, Packet (A, F) needs to wait until all flits of Packet (B, D) have traversed Link 2 before it can proceed to its destination, F. However, if we add one VC to Link 2 then Packet (A, F) can bypass Packet (B, D) while Packet (B, D) is being blocked by Packet (E, D). Each time that Packet (A, F) tries to acquire a VC, a *significant* VC failure occurs until Packet (B, D) leaves Link 2. On the other hand, when Packet (B, D) requests for a VC on Link 3, the VC failure is *not* considered significant because Packet (E, D) is using Link 3.

Figure 5.10 shows the SVCF-driven VC allocation heuristic. The algorithm initializes every link, l, with one VC; this is equivalent to wormhole routing (Line 2).

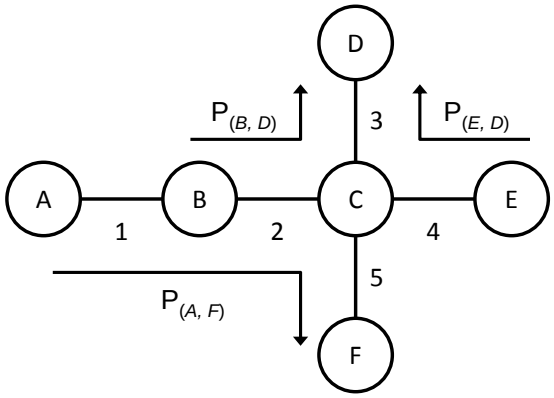

Figure 5.9: An example of significant VC failure.

Algorithm: SVCF-Driven

Input: Application trace, App_{trace}; Network topology, $T(P, L)$; Deterministic routing algorithm, R; Target latency constraint, t_{target}

Output: VC configuration vector, $\mathbf{q}_{vc}$, which contains the number of VCs associated with each link $l \in L$

---

1. **for** l = 1 to $|L|$
2. $\quad q_{vc}^{current}[l] = 1$;
3. $n_{vc} = |L|$;
4. **while** $(n_{vc} < budget_{vc})$ {
5. $\quad \mathbf{q}_{SVCF}^{current} = ComputeSVCF(\mathbf{q}_{vc}^{current})$;
6. $\quad$ **find** l^* that maximizes $n_{SVCF}[l^*]$;
7. $\quad q_{vc}^{current}[l^*] = q_{vc}^{current}[l^*] + 1$;
8. $\quad \mathbf{q}_{vc} = \mathbf{q}_{vc}^{current}$;
9. $\quad n_{vc}$++;
10. }

Figure 5.10: Significant VC failure-driven VC allocation heuristic.

Hence, the total number of VCs in the network, n_{vc}, is initialized to the total number of links, $|L|$. Then, the algorithm proceeds in a greedy fashion: in each iteration the significant VC failures of all the $|L|$ possible links in the current configuration, $\mathbf{q}_{vc}^{current}$,

are calculated using $ComputeSVCF$ (Line 5). We use trace simulations to evaluate $ComputeSVCF$. Subsequently, one VC is added to the link with the maximum number of significant failures, $n_{SVCF}[l^*]$. The algorithm stops when the total number of VCs exceeds the VC budget, $budget_{vc}$.

Figure 5.11 shows the average packet latency of each intermediate configuration using the SVCF-driven heuristic and the greedy addition approach for two different traces, *ferret* and *blackscholes (bs)*. The SVCF-driven heuristic achieves up to 20% and 23% reduction in number of allocated VCs, compared with 160 and 208 VCs for the uniform-2VC and uniform-3VC configurations, respectively. In addition, we achieve average packet latency values within 9% of those achieved by the greedy addition heuristic, with an O($|L|$) speedup. As network size increases, the greedy addition (resp. deletion) method requires significantly more computing resource, i.e., O($|L|$) simultaneous simulations, compared with only one simulation per iteration in our proposed SVCF-driven heuristic. The SVCF-driven heuristic does not reduce the average packet latency as much as the greedy addition heuristic, which we attribute to: (1) the fact that we are not directly minimizing average packet latency, and (2) links with highest SVCF count may not have the heaviest traffic load, and thus cannot reduce APL as much.

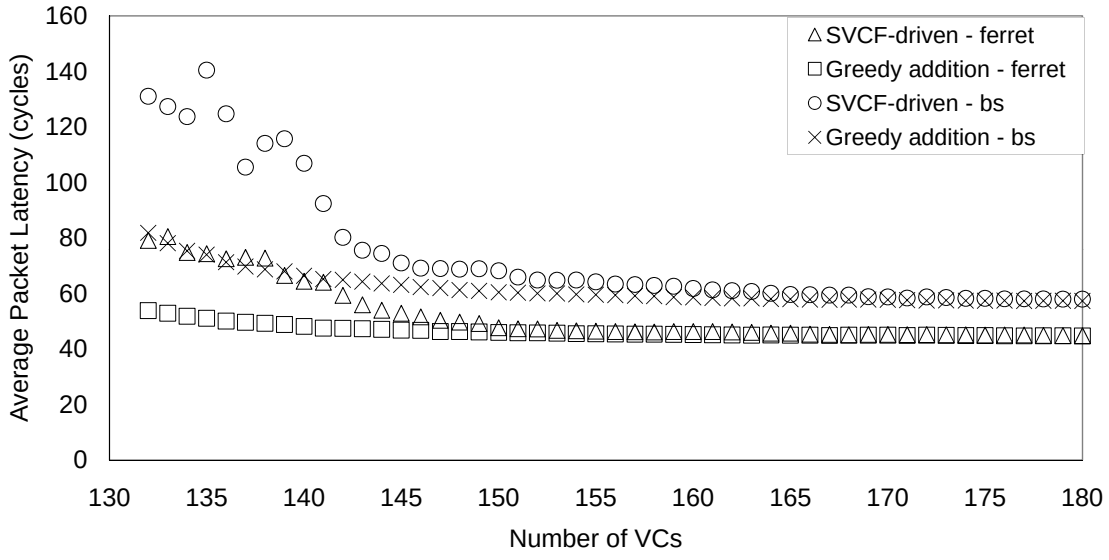

Figure 5.11: Performance of the SVCF-driven VC allocation heuristic on $ferret$ and $blackscholes$ traces.

5.2.5 Queueing Delay-Driven VC Allocation

In this subsection, we describe another simple VC allocation heuristic in which we use the queueing delays observed at each link to drive the VC allocation. In a conventional input-buffered router, queueing delay is the time a flit needs to wait in the buffers before it gets access to its designated output link. The queueing delay of a flit at link l of a router is measured as the difference between the time a flit enters the input buffer of the router until it departs the router through output link l. The queueing delay of a link l is the sum of the queueing delays of all flits passing through link l.

This approach is structurally similar to the SVCF-driven heuristic except that we use queueing delay as the driving metric. However, the queueing delay-driven approach can better capture bottleneck links early in the VC allocation process as shown in Figure 5.12. We note that our proposed queueing delay-driven heuristic fails to improve average packet latency after certain number of VCs have been allocated. This is because after a few VC allocations the bottleneck shifts from the nodes with heavy traffic to downstream nodes; however, VC failure at downstream nodes causes the queueing delay of the flits residing in the source nodes to increase. Hence, the queueing delay-driven approach will allocate VCs to links in source nodes instead of links in downstream nodes. Figure 5.12 shows the average packet latency of each intermediate configuration using queueing delay-driven, SVCF-driven heuristics, as well as the greedy addition heuristic for *canneal* trace from the PARSEC benchmark suite.

5.2.6 Top-k Selection Heuristic

In this subsection, we propose an efficient and yet simple approach for improving the performance of our SVCF- and queueing delay-driven heuristics. The hypothesis is that we can significantly improve the quality of solutions obtained by SVCF- and queueing delay-driven heuristics by simultaneously evaluating more than one of their suggested solutions. In other words, we will evaluate the top-k solutions and determine the best solution according to reduction of average packet latency.

Figure 5.13 shows our top-k SVCF-driven VC allocation heuristic. The algorithm initializes every link, l, with one VC (Line 2). Hence, the total number of VCs in

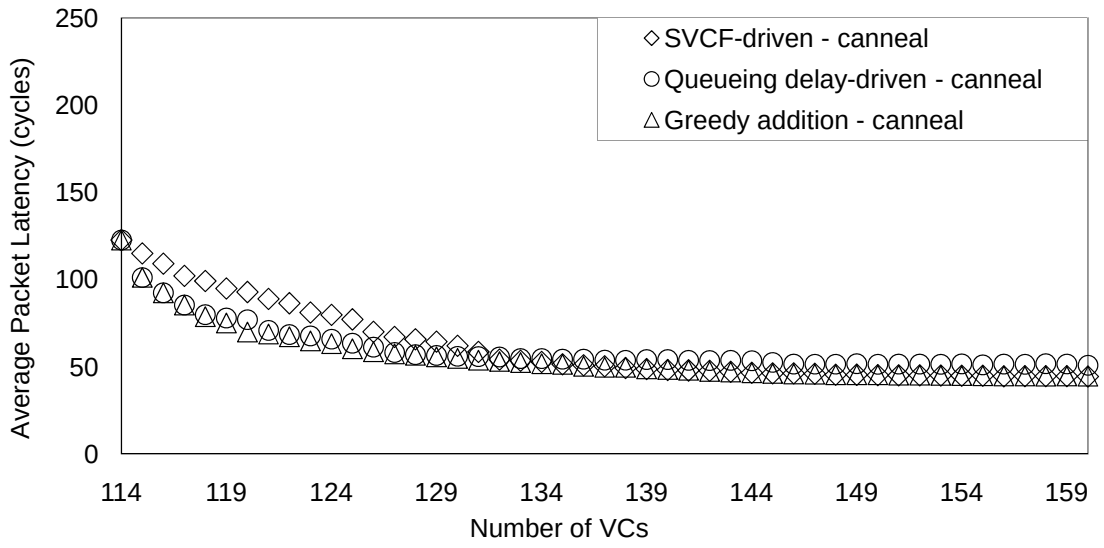

Figure 5.12: Comparison of SVCF-driven, queue delay-driven, and greedy addition VC allocation heuristics on $canneal$ trace.

the network, n_{vc}, is initialized to the total number of links, $|L|$. Then, the algorithm proceeds in a greedy fashion: in each iteration, significant VC failure values corresponding to each link in the current configuration, $\mathbf{q}_{vc}^{current}$, are computed using $ComputeSVCF$, where we use trace simulation to evaluate $ComputeSVCF$ (Line 5). We then find the top k links that have the highest number of significant VC failures (Line 6), and add one VC to each of these k configurations (Line 8). Next, we run k parallel trace simulations to evaluate the quality of each configuration, and pick the one that minimizes the average packet latency the most (Line 11). In the pseudocode of Figure 5.13, $ComputeAPL$ performs trace simulation on a given VC configuration and reports the average packet latency for that VC configuration. The algorithm stops when the total number of VCs exceeds $budget_{vc}$.

To determine an appropriate value for k, we perform a simple sensitivity analysis in which we run the heuristic for multiple k values with reduced trace length. In our sensitivity experiments, we are interested in the number of allocated VCs to satisfy an average packet latency constraint for a given VC configuration. We perform the sensitivity experiments for $k \in \{1, 3, 5, 8, 10, 15\}$ and assess at which point the increase in k does not improve the quality of the solutions. Figure 5.14 shows the sensitivity analysis results for all PARSEC traces. The x-axis shows the value of k, and the y-axis is

Algorithm: Top-k SVCF-Driven

Input: Application trace, App_{trace}; Network topology, $T(P, L)$; Deterministic routing algorithm, R; Target latency constraint, t_{target}

Output: VC configuration vector, $\mathbf{q}_{vc}$, which contains the number of VCs associated with each link $l \in L$

1. **for** l = 1 to $|L|$
2. $q_{vc}^{current}[l] = 1$;
3. $n_{vc} = |L|$;
4. **while** $(n_{vc} < budget_{vc})$ {
5. $\mathbf{q}_{SVCF}^{current} = ComputeSVCF(\mathbf{q}_{vc}^{current})$;
6. **find** top-k l_i^* corresponding to top-k $n_{SVCF}[l_i^*]$;
7. **for** i=1 to k {
8. $q_{vc}^{current}[l_i^*] = q_{vc}^{current}[l_i^*] + 1$;
9. $APL[i]$=$ComputeAPL(q_{vc}^{current}[l_i^*])$;
10. }
11. **find** m_i^* that minimizes $APL[m_i^*]$, where $l_1^* \leq m_i^* \leq l_k^*$
12. $\mathbf{q}_{vc}^{current} = q_{vc}^{current}[l_m^*]$;
13. $\mathbf{q}_{vc} = \mathbf{q}_{vc}^{current}$;
14. n_{vc}++;
15. }

Figure 5.13: Top-k significant VC failure-driven VC allocation heuristic.

number of VCs required to satisfy the average packet latency of a uniform configuration with two VCs per link. From Figure 5.14, we observe that for all of the traces, except for $vips$, k = 5 gives the best tradeoff between quality of results and runtime, i.e., the reduction in the number of allocated VCs is within 2% beyond k = 5. For the $vips$ trace, k = 10 gives an additional 8% improvement in average packet latency compared with k = 5; however, in our experiments we assume k = 5 for all the PARSEC benchmark traces. Similarly, we have performed sensitivity analysis for our queueing delay-driven heuristic and have determined that k = 15 offers the best tradeoff between quality of the

results and runtime. The results presented are for a mesh network with 16 nodes and 64 links (details of the setup are explained in Section 5.4.1).

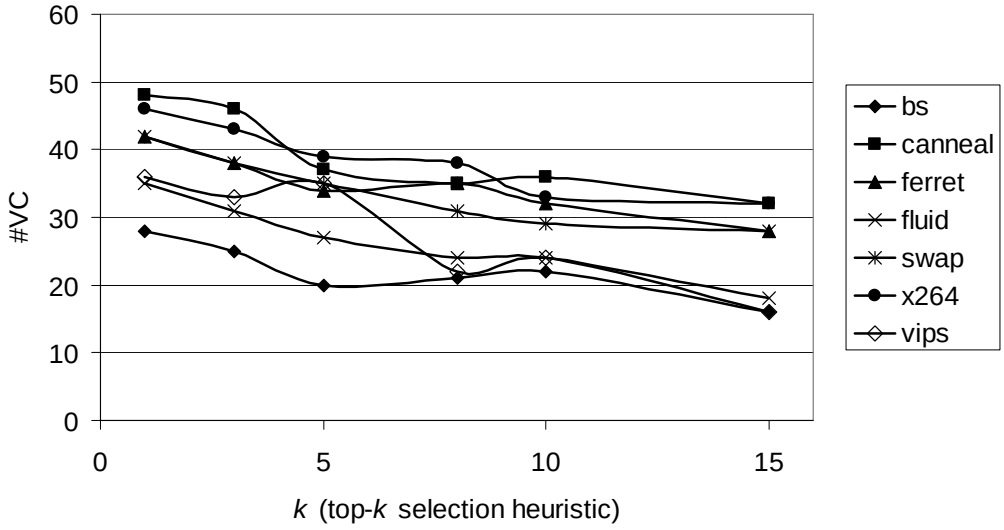

Figure 5.14: Sensitivity analysis of the k parameter for the PARSEC benchmark traces.

5.3 Efficient Metaheuristics

In this section, we propose two efficient and more robust VC allocation metaheuristics by combining the heuristics described in Subsections 5.2.4 and 5.2.5.

5.3.1 Hybrid Metaheuristic

In our proposed hybrid metaheuristic, we combine the top-k SVCF-driven and queueing delay-driven VC allocation heuristics such that in each iteration we pick the better result from these two heuristics. The key to our hybrid metaheuristic is that SVCF-driven and queueing delay-driven VC allocation heuristics each performs quite well in different VC regimes. In other words, the SVCF-driven heuristic seems to perform well once there are already a few VCs inserted in the network, whereas the queueing delay-driven heuristic performs well in the beginning.

Figure 5.15 shows our proposed hybrid metaheuristic. The algorithm initializes every link, l, with one VC; this is equivalent to wormhole routing (Line 2). Hence, the total number of VCs in the network, n_{vc} is initialized to the total number of links, $|L|$. Then, the algorithm proceeds in a greedy fashion: in each iteration, the significant VC failures associated with all of the $|L|$ possible links in the current configuration, $\mathbf{q}_{vc}^{current}$, are calculated using $ComputeSVCF$ (Line 5). In parallel, we also run $ComputeQD$ on $\mathbf{q}_{vc}^{current}$ to determine the total queueing delay associated with all of the $|L|$ possible links (Line 6). Subsequently, we find the corresponding top-k and top-k' links with the highest number of significant VC failure and queueing delay, respectively (Lines 7-8). Then, we run $k + k'$ parallel trace simulations in which we add one VC to each of the suggested $k+k'$ configurations and evaluate their impact on average packet latency (Lines 9-12 and 13-16). Finally, we pick the VC configuration with the lowest average packet latency and set that as the starting configuration for the next iteration. The algorithm stops when the total number of allocated VCs exceeds the VC budget, $budget_{vc}$.

5.3.2 Multi-Stage Metaheuristic

From previous subsections, we observe that our queueing delay-driven heuristic performs well starting from an initial configuration (i.e., wormhole configuration), and that our SVCF-driven heuristic performs well when there are already a number of VCs allocated. Knowing this, we propose a two-stage metaheuristic in which we start with our top-k queueing delay-driven algorithm and will switch to top-k SVCF-driven algorithm once the difference in average packet latency of two consecutive configurations falls below a certain threshold, s.

Figure 5.16 shows our proposed two-stage metaheuristic which is similar to our hybrid metaheuristic, but uses some constant number fewer trace simulations with no degradation in results. The algorithm initializes every link, l, with one VC; this is equivalent to wormhole routing (Line 2). Hence, the total number of VCs in the network, n_{vc} is initialized to the total number of links, $|L|$. Then, the algorithm proceeds in a greedy fashion: in each iteration the algorithm picks either top-k SVCF-driven or top-k queueing delay-driven heuristic based on the average packet latency improvement threshold, s, which is defined as the difference in average packet latency for two consecutive iter-

Algorithm: Hybrid Metaheuristic

Input: Application trace, App_{trace}; Network topology, $T(P, L)$; Deterministic routing algorithm, R; Target latency constraint, t_{target}

Output: VC configuration vector, $\mathbf{q}_{vc}$, which contains the number of VCs associated with each link $l \in L$

---

1. **for** l = 1 to $|L|$
2. $\quad q_{vc}^{current}[l] = 1$;
3. $n_{vc} = |L|$;
4. **while** $(n_{vc} < budget_{vc})$ {
5. $\quad \mathbf{q}_{SVCF}^{current} = ComputeSVCF(\mathbf{q}_{vc}^{current})$;
6. $\quad \mathbf{q}_{QD}^{current} = ComputeQD(\mathbf{q}_{vc}^{current})$;
7. $\quad$ **find** top-k l_i^* corresponding to top-k $n_{SVCF}[l_i^*]$;
8. $\quad$ **find** top-k' m_i^* corresponding to top-k $n_{QD}[m_i^*]$;
9. $\quad$ **for** i=1 to k {
10. $\quad\quad q_{vc}^{current}[l_i^*] = q_{vc}^{current}[l_i^*] + 1$;
11. $\quad\quad APL[i] = ComputeAPL(q_{vc}^{current}[l_i^*])$;
12. $\quad$ }
13. $\quad$ **for** i=1 to k' {
14. $\quad\quad q_{vc}^{current}[m_i^*] = q_{vc}^{current}[m_i^*] + 1$;
15. $\quad\quad APL[i+k] = ComputeAPL(q_{vc}^{current}[m_i^*])$;
16. $\quad$ }
17. $\quad$ **find** n_i^* that minimizes $APL[n_i^*]$, where $l_1^* \leq n_i^* \leq l_{k+k'}^*$
18. $\quad \mathbf{q}_{vc}^{current} = q_{vc}^{current}[l_n^*]$;
19. $\quad \mathbf{q}_{vc} = \mathbf{q}_{vc}^{current}$;
20. $\quad n_{vc}$++;
21. }

Figure 5.15: Hybrid metaheuristic using top-k SVCF-driven and queueing delay-driven VC allocation heuristics.

ations. Based on our previous findings (cf. Subsection 5.2.5), we start with queueing delay-driven heuristic first (Lines 7-15). Once, the APL improvement threshold falls below the defined value, the algorithm chooses the top-k SVCF-driven heuristic (Lines 19-26). We use trace simulations to evaluate both $ComputeSVCF$ and $ComputeQD$. The algorithm stops when the total number of allocated VCs exceeds the VC budget, $budget_{vc}$. To find an appropriate value for s, we have performed similar sensitivity analysis as described in Subsection 5.2.6, and have chosen s = 0.5.

5.3.3 Runtime Analysis of Metaheuristics

As we have noted earlier, simple greedy addition and deletion heuristics require relatively larger runtime compared with average-rate approaches. The runtime complexities of our proposed heuristics, using the new metrics, are as follows.

Let m be the number of VCs added to an initial VC configuration. The runtime of our SVCF- and queueing delay-driven heuristics, $t_{SVCF/QD}$, for any given input trace is expressed as follows:

$$t_{SVCF/QD} = m \times t(trace) \tag{5.3}$$

where $t(trace)$ is the average runtime of (cycle-accurate, flit-level) trace simulation simulator over all VC configurations explored by the algorithm. The above two heuristics do not require O($|L|$) simultaneous simulations as in the greedy addition (resp. deletion), which results in an O($|L|$) speedup.

The runtime for our proposed top-k SVCF- and queueing delay-driven heuristics to insert m VCs is

$$t_{\text{top-}k\ SVCF/\text{top-}k\ QC} = m \times k \times t(trace) \tag{5.4}$$

which also results in an O($|L|$) speedup (since k is a small constant) when compared with the greedy addition and deletion heuristics. Finally, the runtime of our proposed hybrid and two-stage metaheuristics to insert m VCs are

$$t_{hybrid} = m \times (k + k') \times t(trace) \tag{5.5}$$

Algorithm: Two-Stage Metaheuristic

Input: Application trace, App_{trace}; Network topology, $T(P, L)$; Deterministic routing algorithm, R; Target latency constraint, t_{target}

Output: VC configuration vector, $\mathbf{q}_{vc}$, which contains the number of VCs associated with each link $l \in L$

1. **for** l = 1 to $|L|$
2. $q_{vc}^{current}[l] = 1$;
3. $n_{vc} = |L|$;
4. $switch$ = false;
5. **while** $(n_{vc} < budget_{vc})$ {
6. **if**(!switch) {
7. $\mathbf{q}_{QD}^{current} = ComputeQD(\mathbf{q}_{vc}^{current})$;
8. **find** top-k m_i^* corresponding to top-k $n_{QD}[m_i^*]$;
9. **for** i=1 to k {
10. $q_{vc}^{current}[l_i^*] = q_{vc}^{current}[l_i^*] + 1$;
11. $APL[i] = ComputeAPL(q_{vc}^{current}[l_i^*])$;
12. }
13. **find** n_i^* that minimizes $APL[n_i^*]$, where $l_1^* \leq n_i^* \leq l_k^*$
14. $\mathbf{q}_{vc}^{current} = q_{vc}^{current}[l_n^*]$;
15. n_{vc}++;
16. **if**$(APL_{n_{vc}-1} - APL_{n_{vc}} < s)$
17. $switch$ = true;
18. }
19. **else** {
20. $\mathbf{q}_{SVCF}^{current} = ComputeSVCF(\mathbf{q}_{vc}^{current})$;
21. **find** top-k' l_i^* corresponding to top-k $n_{SVCF}[l_i^*]$;
22. **for** i=1 to k' {
23. $q_{vc}^{current}[m_i^*] = q_{vc}^{current}[m_i^*] + 1$;
24. $APL[i] = ComputeAPL(q_{vc}^{current}[m_i^*])$;
25. }
26. **find** n_i^* that minimizes $APL[n_i^*]$, where $l_1^* \leq n_i^* \leq l_{k'}^*$
27. $\mathbf{q}_{vc}^{current} = q_{vc}^{current}[l_n^*]$;
28. $\mathbf{q}_{vc} = \mathbf{q}_{vc}^{current}$;
29. n_{vc}++;
30. }
31. }

Figure 5.16: Two-stage metaheuristic using top-k SVCF-driven and queueing delay-driven VC allocation heuristics.

$$t_{two-stage} = r \times k \times t(trace) + (m - r) \times k' \times t(trace) \quad (5.6)$$

where k and k' are derived using sensitivity analysis as explained earlier (cf. Subsection 5.2.6), and r denotes the number of iterations that the two-stage metaheuristic chooses the queueing delay-driven heuristic.

5.4 Evaluation and Discussion

In this section, we describe our experimental setup, and then we evaluate the performance of our proposed metaheuristics with respect to (1) uniform VC configurations and (2) greedy addition heuristic. We show that our proposed metaheuristics can achieve the same quality of results with significant runtime improvement (i.e., O($|L|$)).

5.4.1 Experimental Setup

To determine the average packet latency of a given application trace, we use PopNet [98], a flit-level, cycle accurate on-chip network simulator. PopNet models a typical input-buffered VC router with four pipeline stages. Route computation and VC allocation are performed in the first pipeline stage, followed by switch arbitration, switch traversal and link traversal. The head flit of a packet proceeds through all four stages while the body flits bypass the first pipeline stage and inherit the output port and output VC reserved by the head flit. Non-uniform VC configurations are implemented by individually configuring the number of VCs at each router port. The routers are connected in a two-dimensional mesh topology and dimension-ordered routing is employed where packets are first routed in the X dimension followed by the Y dimension. The latency of a packet is measured as the delay between the time the header flit is injected into the network and the time the tail flit is consumed at the destination. The APL value reported by PopNet is the average latency over all packets in the input traffic trace.

To evaluate our VC allocation heuristics, we use seven applications from the PARSEC benchmark suite, namely, *canneal*, *blackscholes*, *fluidanimate*, *ferret*, *swaptions*, *vips*, and *x264* [31]. These benchmarks are multithreaded programs that are representative of emerging future workloads. The network traffic traces are generated by running these programs on *Virtutech Simics* [18], a full system simulator, and capturing

Table 5.1: Processor configuration for generation of PARSEC benchmark traces.

Number of Cores	16
Private L1 cache	32KB
Shared L2 cache	1MB distributed over 16 banks
Memory latency	170 cycles
Network topology	4×4 mesh
Packet sizes	72B data packets, 8B control packets

the program's memory trace. The GEMS toolset [75] runs on top of *Simics* and performs accurate timing simulation. We simulate a 16-core CMP architecture arranged in a 4×4 mesh, with parameters shown in Table 5.1. Each tile consists of an in-order SPARC core with private L1 and shared L2 cache. DRAM is attached to the chip using four memory controllers that are at the corners of the mesh. For the purpose of trace collection, the GARNET interconnect model [20] with 8 VCs per link is used.

In all seven traces, every node both sends and receives packets. A 4×4 mesh has 48 links and 16 injection ports at each node. The number of VCs for each of these links can be individually configured. To decouple the VC allocation and buffer space allocation problems, we assume that each VC has a fixed buffer length of 10 flits/VC, which is larger than the maximum packet size of the application. We statically allocate four VCs to each of the 16 injection ports in our heuristics to ensure that there is no head-of-line blocking and that the performance bottleneck is shifted to the regular links and not the injection ports. Hence, we start with a wormhole configuration with 112 VCs (1 VC/link + 4 VCs/injection port) and set our VC budget to 256 VCs, equivalent to a uniform configuration with four VCs at every link.

5.4.2 Experimental Results

In this subsection, we present the results of our proposed VC allocation meta-heuristics on the reduction in number of VCs under a target performance. We compare our proposed metaheuristics with uniform VC allocation, and the simple greedy addition heuristic using seven traces from PARSEC benchmark suites [31]. Figures 5.17 and 5.18

show that our proposed algorithms can reduce the number of VCs required to achieve the same target latency as uniform configurations with two VCs per port and three VCs per port, respectively. We also show that our metaheuristics can achieve higher reductions in the number of VCs compared with the simple greedy approaches.

Figure 5.17 shows the number of allocated VCs using our hybrid and two-stage metaheuristics to achieve the same average packet latency as uniform-2VC. We observe that both of our proposed metaheuristics can achieve up to 24.4% reduction in number of allocated VCs. On average, our hybrid and two-stage metaheuristics reduce the number of VCs by around 13.5% and 14.5% across all benchmarks with respect to the uniform-2VC and uniform-3VC configurations, respectively. Figure 5.18 shows the number of VCs required using our algorithms to achieve the same average packet latency as uniform-3VC. We see high reductions of up to 38% for both hybrid and two-stage metaheuristics, compared to the 208 VCs required by the uniform configuration.

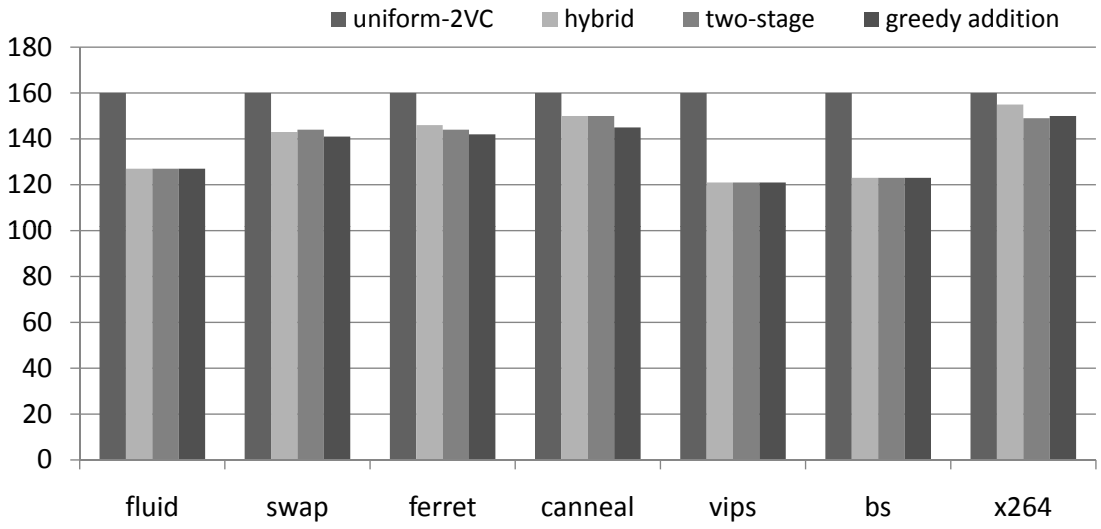

Figure 5.17: Comparison of hybrid and two-stage VC allocation metaheuristics versus the greedy addition heuristic and uniform-2VC configuration.

Figures 5.17 and 5.18 also show that our proposed metaheuristics match the solution quality of the greedy addition heuristic while reducing the runtime complexity by $O(|L|)$. Figure 5.19 shows the number of simulations required by our proposed metaheuristics and the simple greedy addition heuristic to achieve the same average

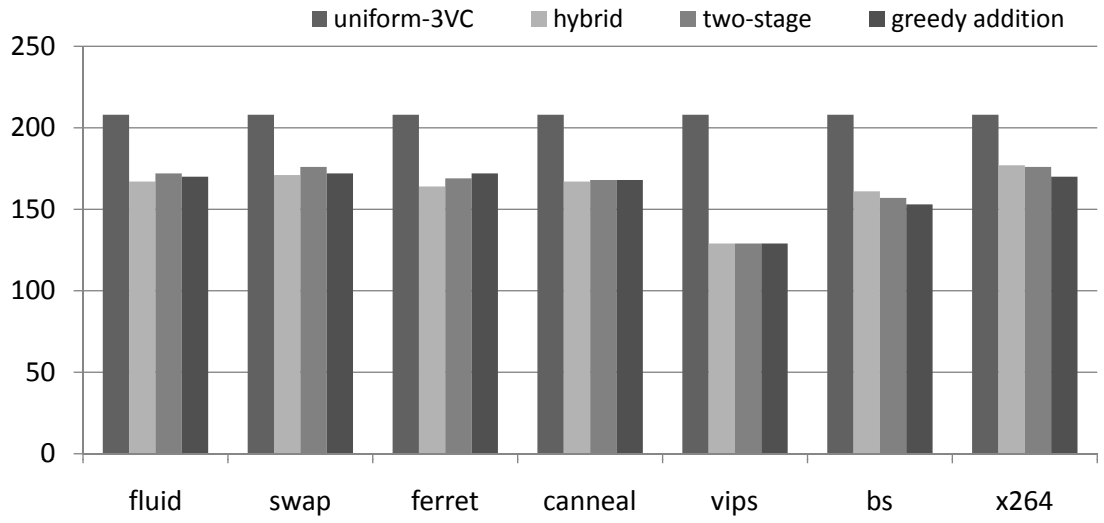

Figure 5.18: Comparison of hybrid and two-stage VC allocation metaheuristics versus the greedy addition heuristic and uniform-3VC configuration.

packet latencies as uniform-2VC and uniform-3VC. We observe up to 90% reduction in the total number of simulations compared to the greedy addition approach.

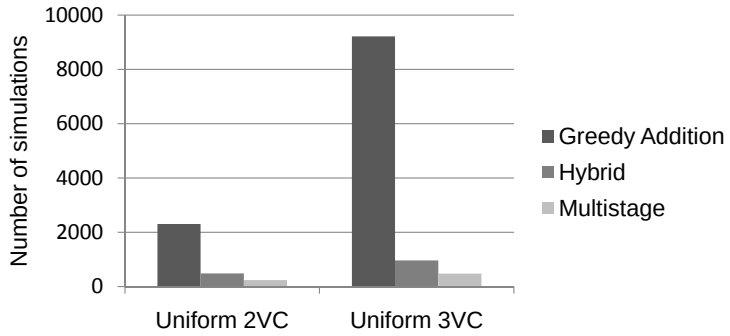

Figure 5.19: Comparison of number of simulations required for our proposed meta-heuristics versus the greedy addition heuristic.

Finally, to assess the impact of our proposed approach on network power and area, we use ORION 2.0 [63]. The ORION 2.0 router model assumes the same number of VCs at every port in the router. However, we need to compute the router power for non-uniform VC configurations. Hence, we first estimate the power overhead of adding a single VC to all router ports. This is done by computing the router power for

uniform configurations with one, two, and three VCs per port and averaging the power difference between the uniform-1VC (uniform-2VC) and uniform-2VC (uniform-3VC) configurations. This gives an estimate of the power overhead of adding one VC to all router ports. This value is divided by the number of router ports to determine the overhead of adding a single VC to just one port. We use a similar approach to find the area overhead of adding a single VC to one router port. The number of VCs saved by our trace-driven heuristics compared to uniform configurations is then multiplied by the power (area) overhead of adding a single VC to a router port to estimate our power (area) savings. From our experiments, we observe that our approach can reduce the network power by up to 7% and 14% compared to the uniform-2VC and uniform-3VC configurations, respectively, while achieving the same performance. Similarly, we achieve up to 9% and 16% area reduction compared to the uniform-2VC and uniform-3VC configurations, respectively.

5.5 Conclusions

In this chapter, we propose a trace-driven approach to the optimization of NoC configurations. We specifically consider the problem of application-specific VC allocation, which seeks to minimize the average packet latency of a given application while minimizing the VC resources required for the NoC implementation. Previous approaches to this problem have been based on the use of average rate-driven models to drive design choices, but fail to accurately capture the actual application traffic characteristics, which can lead to designs that are poorly matched to the application. In contrast, our proposed algorithms are driven by actual application traffic traces in which the selection of non-uniform VC allocations is based on the impact on actual performance. In comparison with uniform VC allocation, our methods achieve up to 51% and 74% reduction in number of VCs and average packet latency, respectively. We also compare our proposed approach against an existing average-rate driven method [61] and observe up to 35% reduction in number of VCs.

In addition, to enhance the scalability of our approach to larger networks, we propose a new metric called “significant VC failure” which efficiently captures run-

time VC contentions and allows us to speed up our method. Also, we propose new metaheuristics that achieve significant speedups (i.e., O($|L|$)) with no loss of solution quality. In comparison with uniform VC allocation, our metaheuristics achieve up to 38% reduction in number of VCs.

5.6 Acknowledgments

This chapter is in part a reprint of:

- Andrew B. Kahng, Bill Lin, **Kambiz Samadi** and Rohit Sunkam Ramanujam, "Efficient Trace-Driven Metaheuristics for Optimization of Networks-on-Chip Configurations", *Proc. IEEE/ACM International Conference on Computer-Aided Design*, 2010, pp. 256–263.

- Andrew B. Kahng, Bill Lin, **Kambiz Samadi** and Rohit Sunkam Ramanujam, "Trace-Driven Optimization of Networks-on-Chip Configurations", *Proc. ACM/ IEEE Design Automation Conference*, 2010, pp. 432–437.

I would like to thank my coauthors Prof. Bill Lin, Rohit Sunkam Ramanujam, and Prof. Andrew B. Kahng.

Chapter 6

Multi-Product Floorplan Optimization for Chip Multiprocessors

6.1 Introduction

Due to the diversity of market demands, modern chips are offered in multiple versions also known as SKUs (stock-keeping units). [10] shows an example of a modern 45 nm CPU design that proliferates to 10 different configurations (with different numbers of building blocks) and 21 distinct products in one of its market plans.

Chip multiprocessor (CMP) floorplanning can be performed at two levels: (1) tile level, and (2) chip level. In (1) the objective is to minimize area subject to performance constraints (i.e., wirelength) whereas in (2) the objective is to select a configuration (i.e., number of different resources and their corresponding placement) which satisfies certain power, area and performance constraints. Figure 6.1 shows an example of a tile floorplan, where $X_1, \cdots, X_6$ denote cores, communication blocks, caches, etc. and WS denotes white space. The tile floorplan is designed in such a way that communication between tiles can easily be achieved by placing them next to each other. Chip-level floorplanning of a single CMP product may seem to be a relatively simple task; however, simultaneous optimization of multiple product floorplans with varying resources and constraints is quite challenging.

Previous chapters focus on improved understanding of the design space for NoC

power, performance, and area. Since CMPs are a major consumer of NoCs, we now assume an existing on-chip communication network, and focus on simultaneous chip-level floorplan optimization of multiple CMP products. Our goal is to enable efficient exploration of achievable CMP floorplans given power and area constraints. Results of such an exploration will impact the on-chip communication network via available power and area budgets.

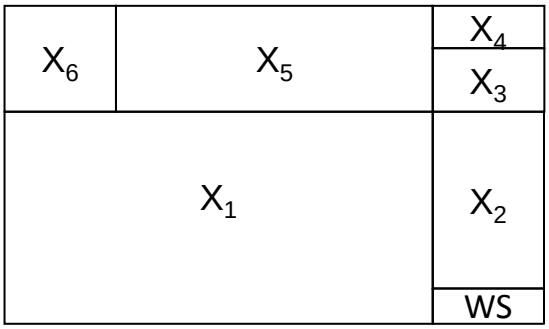

Figure 6.1: An example of a tile-level floorplan.

Typical CMP products include a low-cost, low-power model which targets mobile or low-end markets, a high-performance model which targets high-end markets, and medium-cost SKUs to fill the spectrum in between. These different product classes share the same building blocks, e.g., CPUs, memory controllers, common cache coherency protocol, interconnect topology, etc. However, they differ in the number of blocks, amount of on-chip memory, and I/O bandwidth. These differences lead to changes in the chip floorplan, and hence imply extra design effort which increases both design cost and time-to-market. We note that the common ground between different designs can be exploited to minimize design effort. To this end, in this chapter, we present a novel floorplanning approach that achieves two main objectives. (1) The floorplans of all product classes are simultaneously optimized. (2) The corresponding floorplans satisfy a new *choppability* requirement. This means that a smaller configuration can be obtained from a bigger configuration by simply chopping and shifting the floorplan as illustrated in Figure 6.2.

Figure 6.2 shows a simple example with three product classes. Product P_1 is comprised of three cores and one memory controller. Product P_2 is obtained from P_1 by

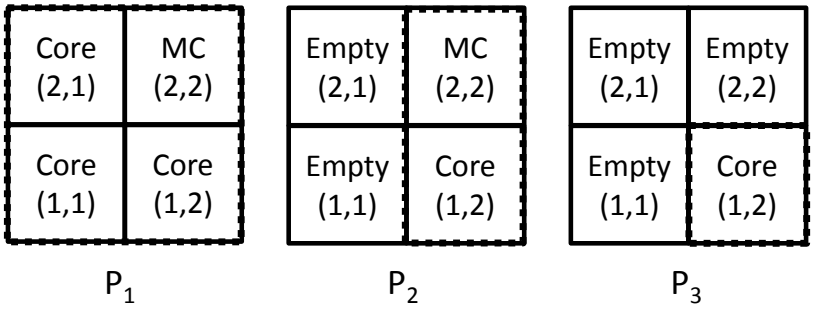

Figure 6.2: Example floorplans for three different CMP products. Chopped parts are labeled as *Empty* for illustration purposes.

chopping out Column 1 (i.e., the left-most column), and product P_3 is obtained from P_2 by chopping out the top row.

In general, the floorplan of the largest configuration is designed first. Then, the floorplan of each smaller configuration is obtained by literally chopping the biggest floorplan. We propose an efficient integer-linear programming (ILP) approach to solve the multi-product CMP floorplanning problem. Unlike traditional chip floorplanning approaches [19, 97, 101, 104, 116], our approach simultaneously optimizes the floorplan of multiple CMP products such that the floorplan of smaller products can be easily derived from those of the larger products via *chopping* operations (cf. Section 6.2). The contributions of this chapter are as follows.

- We define a *choppability property* for a given CMP product such that the floorplans of smaller CMP products can be derived by appropriate chopping operations.
- We propose an efficient ILP-based approach to simultaneously optimize the floorplans of multiple CMP products subject to design area and power constraints.
- We extend our baseline problem formulation to enable efficient design space exploration of floorplans under certain power and area budgets.
- We consider different width and height values for cores, memory controllers, and memory channel tiles to account for heterogeneous building block sizes.

- Finally, we provide several examples with varying resources and show that our approach efficiently provides choppable floorplans across all products.

The remainder of this chapter is organized as follows. Section 6.2 describes the definitions and notations used in our proposed approach. Section 6.3 proposes our basic problem formulation and then describes necessary adjustments to enable handling of additional building block types. Section 6.4 provides three extensions to our basic formulation which enable efficient design space exploration of viable floorplans. Section 6.5 describes our developed infrastructure and discusses experimental results. Finally, Section 6.6 concludes the chapter.

6.2 Preliminaries and Notations

CMPs consist of different building blocks, e.g., core, memory controller, memory channels, I/O, etc., which are laid out in rows and columns of a given grid. We focus on the chip-level floorplanning problem of CMPs, i.e., tile-level floorplanning is assumed to have already been done. Our goal is to simultaneously optimize the chip-level floorplans of multiple CMP products subject to area and power constraints. Let $P_1, P_2, \ \cdots, P_S$ denote S different product types, and let $k = 1, \ \cdots, K$ denote block types. Thus, each product type corresponds to a K-tuple $< n_1, \ \cdots, n_K >$, where n_i shows the number of resources of type i, and the products are denoted as

$$\begin{aligned} P_1 &=< n_{11}, \ \cdots, n_{1K} > \\ P_2 &=< n_{21}, \ \cdots, n_{2K} > \\ &\cdots \\ P_S &=< n_{S1}, \ \cdots, n_{SK} > \end{aligned}$$

We define the relation $\succ$ to be a total order on the set of products P if the following property holds.

$$P_i \succ P_j \Leftrightarrow n_{iK} \geq n_{jK} \ \ (k = 1, \ \cdots, K; \ \ \forall 1 \leq i, j \leq S) \tag{6.1}$$

Without loss of generality we index the products such that $P_1 \succ P_2 \succ \cdots \succ P_S$, and that means P_i contains P_{i+1} for all $i = 1, \cdots, S-1$. We assume four main types of building blocks in a CMP: cores, memory controllers (MCs), memory channels, and I/O devices (e.g., PCI Express, QPI, etc.). For technical reasosns, we also consider an empty tile to be a building block; in the following, we use $k = 1, 2, 3, 4, 5$ to respectively denote (1) empty, (2) core, (3) memory controller, (4) memory channel, and (5) I/O tile.

In a CMP product with R rows and C columns (i.e., $R \times C$ tiles), the binary variable $u_{r,c}$ indicates the occupying status of tile (r,c) where $1 \leq r \leq R$ and $1 \leq c \leq C$. $u_{r,c} = 1$ (resp. 0) means that tile (r,c) is occupied (resp. empty). To extend our definition to multiple products, $u^i_{r,c}$ denotes whether tile (r,c) in product P_i ($1 \leq i \leq S$) is occupied. Finally, we use $u^i_{r,c,k}$ to denote whether tile (r,c) in a product P_i contains a building block of type k. Below, we show a matrix representation of product P_S.

$$\mathbf{U^S_{RC}} = \begin{bmatrix} u^S_{1,1,1} & \cdots & u^S_{1,c,1} \\ \vdots & \ddots & \vdots \\ u^S_{R,1,1} & \cdots & u^S_{R,C,1} \end{bmatrix} \\ \vdots \\ \begin{bmatrix} u^S_{1,1,K} & \cdots & u^S_{1,C,K} \\ \vdots & \ddots & \vdots \\ u^S_{R,1,K} & \cdots & u^S_{R,C,K} \end{bmatrix} \tag{6.2}$$

The three products P_1, P_2 and P_3 in Figure 6.2 have three building blocks: (1) empty ($k = 1$), (2) core ($k = 2$), and (3) memory controller ($k = 3$). The respective variable encodings are:[1]

- P_1: $u^1_{1,1,2} = 1$; $u^1_{1,2,2} = 1$; $u^1_{2,1,2} = 1$; $u^1_{2,2,3} = 1$
- P_2: $u^2_{1,1,1} = 1$; $u^2_{1,2,2} = 1$; $u^2_{2,1,1} = 1$; $u^2_{2,2,3} = 1$

[1]Here, we only show the variables with the value '1'.

- P_3: $u^3_{1,1,1} = 1$; $u^3_{1,2,2} = 1$; $u^3_{2,1,1} = 1$; $u^3_{2,2,1} = 1$

Simultaneous optimization of the floorplan across multiple products seeks to derive the floorplans of smaller products from those of larger ones to minimize implementation efforts. If core or memory controller tiles are taken out arbitrarily from a given product, what remains is not necessarily an optimal floorplan for the smaller product, i.e., there may be white space or some part of the layout may change. Hence, the goal is to remove the resources in such a way that the final layout of the larger products is literally "chopped" to obtain those of the smaller products. A chopping operation simply removes an entire row or column (with occupied tiles converted to empty tiles) such that we can achieve the floorplan of the smaller products. In the following, we formally define chopping operations and the choppability property for a given product.

Definition 1. Chopping operation (column): for some $c^*, 1 \leq c^* \leq C$,

$$u^i_{r,c^*,k} = 1 \Rightarrow u^{i+1}_{r,c^*,k} = 0, \quad k > 1$$
$$u^i_{r,c^*,k} = 1 \Rightarrow u^{i+1}_{r,c^*,k} = 1, \quad k = 1$$

Definition 1′. Chopping operation (row): for some $r^*, 1 \leq r^* \leq R$,

$$u^i_{r^*,c,k} = 1 \Rightarrow u^{i+1}_{r^*,c,k} = 0, \quad k > 1$$
$$u^i_{r^*,c,k} = 1 \Rightarrow u^{i+1}_{r^*,c,k} = 1, \quad k = 1$$

Definition 2. P_i can be chopped to P_j ($i < j$), notated as $P_i \rightsquigarrow P_j$, if there exists a sequence of chopping operations that transforms P_i to P_j.

Definition 3. A set of floorplans is choppable if $P_i \rightsquigarrow P_{i+1}, \forall i = 1, \cdots, S-1$.

Returning to Figure 6.2, the left column of P_1 is chopped to obtain P_2. Then, the top row of P_2 is chopped to obtain P_3. To achieve *choppable* floorplans, we define a set of constraints to satisfy the above definitions and properties. In large products with $R, C > 2$, we need to accordingly shift the remaining rows and columns (i.e., after chopping operations) to achieve the final floorplan. In the next section, we formulate multi-product floorplan optimization via choppability constraints.

6.3 Multi-Product Floorplan Optimization

In this section, we propose a multi-product floorplan optimization framework for CMPs. We first introduce our basic problem formulation, and then present necessary enhancements to make it more practically relevant.

6.3.1 Basic Problem Formulation

We use binary variables to represent the occupancy state of any tile in any given product, as in Section 6.2. To maintain a linear objective, we minimize the sum of half-perimeter values of all products instead of area. Suppose $AR = \frac{max\{H,W\}}{min\{H,W\}} \leq R$, where AR, H, W, and R respectively denote aspect ratio, product (i.e., chip) height and width, and aspect ratio upper bound. A rectangle of given perimeter has minimum area when its sides are in the ratio $1 : R$. Without loss of generality let the side lengths be 1 and R. Then, the minimum-area rectangle (having perimeter $= 2 + 2R$) has area $= R$. On the other hand, a rectangle with perimeter $2 + 2R$ has maximum possible area $(\frac{1+R}{2})^2$. Thus, the deviation can be up to a ratio $(R + 1)^2 : 4R$. For example, if R = 2, the worst-case ratio is 9 : 8 which means minimizing half-perimeter can end up with 12.5% more area than minimizing area (for a given half-perimeter). To reduce deviation from area minimization, we can enforce an upper bound on aspect ratio.

We note that unlike tile-level floorplanning, we do not have any overlap constraints as CMP blocks (e.g., core, memory controller, memory channels, etc.) are laid out in a tiled fashion. For now, we assume that there are only three building blocks in a CMP product: (1) empty, (2) core, and (3) memory controller MC. Later, we will relax this assumption. Memory controller tiles can only be placed at the boundary of the design to communicate with the memory channels; however, core tiles can be placed anywhere in the design.

Minimize: $\sum_i (H^i + W^i)$

Subject to:

$$u^i_{r,c,k} \geq u^{i+1}_{r,c,k} \;\; \forall i, r, c; \;\; k > 1 \tag{6.3}$$

$$u^i_{r,c,k} = u^{i+1}_{r,c,k} \;\; \forall i, r, c; \;\; k = 1 \tag{6.4}$$

$$\sum_k u^i_{r,c,k} = 1 \;\; \forall i, r, c, k \tag{6.5}$$

$$\sum_r \sum_c u^i_{r,c,k} = N^i_k \;\; \forall i; \;\; 2 \leq k \leq 3 \tag{6.6}$$

$$u^i_{r,c,3} = 0 \;\; \forall i; \;\; 2 \leq r \leq R-1; \;\; 2 \leq c \leq C-1 \tag{6.7}$$

$$H^i = h \sum_r used^i_r \;\; \forall i \tag{6.8}$$

$$W^i = w \sum_c used^i_c \;\; \forall i \tag{6.9}$$

$$used^i_r \geq \sum_{2 \geq k \geq 3} u^i_{r,c,k} \;\; \forall i; \;\; r; \;\; 1 \leq c \leq C \tag{6.10}$$

$$used^i_c \geq \sum_{2 \geq k \geq 3} u^i_{r,c,k} \;\; \forall i; \;\; c; \;\; 1 \leq r \leq R \tag{6.11}$$

where

- N^i_k denotes total number of instances of the k^{th} building block in product P_i.
- $used^i_r$ and $used^i_c$ respectively denote used rows, and used columns. Used rows (columns) are rows (columns) that are not chopped through row (column) chopping operations. A row r (column c) in product P_i is chopped (i.e., $used^i_r = 0$ ($used^i_c = 0$)) only if all the tiles within that row (column) are empty.
- H^i and W^i respectively denote height and width of product P_i.
- h and w respectively denote tile height and width. In the above formulation, the assumption is that tile width and height are the same for all building blocks. Later, we will relax this assumption.

In the above formulation, Constraints (6.3), (6.4), and (6.5) capture the chopping operation definitions (Definitions 1 and 1′). Constraint (6.5) enforces the existence of

only one building block in a given tile. Constraint (6.6) enforces in each product the total number of each building block type, while Constraint (6.7) ensures that memory controller blocks are only located at the boundary tiles. Constraints (6.8) and (6.9) compute all product heights and widths, corresponding to the numbers of used rows and used columns captured through Constraints (6.10) and (6.10).

6.3.2 Handling More Tile Types

We now relax our previous assumption of having only empty, core, and memory controller tiles, and add *memory channels* (MCh). Memory channels are connected to cores through memory controllers; hence, their placement affects the placement of memory controllers. Memory channels are often a block of contiguous tiles. In a given CMP product, the boundary tiles are reserved for memory channels and I/O devices with the constraint that three out of four sides of the design are for memory channels, and one side is reserved for I/O. Hence, once memory channels are placed, I/O devices can be easily placed for each product in a post-processing step. Figure 6.3 shows the two possibilities for placement of memory channels and I/O devices can be placed at the boundary of the design.[2]

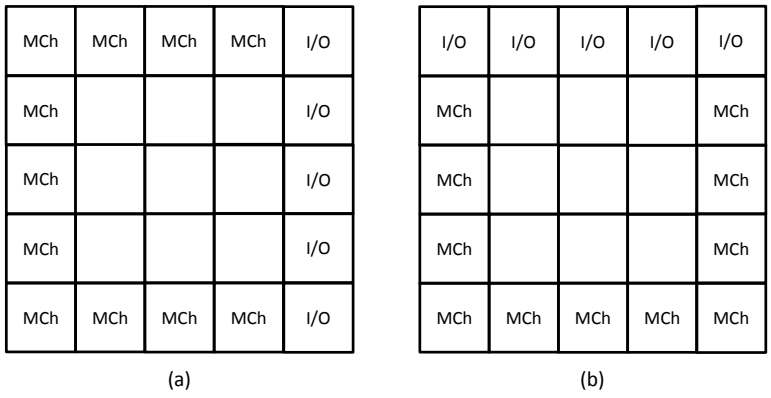

Figure 6.3: Two possible memory channel and I/O placements at the boundary of the design.

Let $g = 1, \cdots, G$ index the memory channel "groups", and let w_g denote the

[2]Note that other possible combinations are symmetric to these two configurations shown in Figure 6.3, and can be achieved by mirroring the design.

size of the memory channel group g. The size of a memory channel is the number of contiguous tiles that it occupies. We use $v^i_{r,c,d,g}$ to indicate that tile (r,c) is the starting tile of memory channel g in the direction d in product P_i, where $d \in \{\uparrow, \rightarrow\}$. Figures 6.3(a) and (b) show that there are two directions in which memory channels can be placed (i.e., $\uparrow, \rightarrow$). Note that the other directions (i.e., $\downarrow, \leftarrow$) are symmetric to the directions d. To ensure that different memory channel groups do not overlap with each other, we use the function f (Equation (6.12)) which determines whether two memory channels overlap. We then enumerate all the pairwise combinations of different memory channels, and apply Constraint (6.14) when there is an overlap between two memory groups.

$$f(< r', c', d', g' >, < r'', c'', d'', g'' >) \tag{6.12}$$
$$= \begin{cases} 1 & \text{if overlapping} \\ 0 & \text{if non-overlapping} \end{cases}$$
$$\forall i, r', c', d', g', r'', c'', d'', g'' \ \ (g' \neq g'')$$
$$v^i_{r',c',d',g'} \ + \ v^i_{r'',c'',d'',g''} \leq 1 \tag{6.13}$$
$$\forall i, r', c', d', g', r'', c'', d'', g'' \ \ (g' \neq g'') \ \ s.t. \ \ f = 1$$
$$\sum_r \sum_c \sum_d v^i_{r,c,d,g} = 1 \tag{6.14}$$
$$\forall i, g$$

Constraint (6.14) ensures that the correct number of memory channels are placed in each product. For example, Figure 6.4 shows a product with two memory channel groups of size one and size two (i.e., $w_1 = 1$ and $w_2 = 2$). If the start position of the memory channel group 2 is (3,1), then we will have the following constraints (there will be more constraints as we change the starting position of the given memory channel).

$$v^i_{3,1,\rightarrow,2} \ + \ v^i_{3,1,\rightarrow,1} \leq 1$$
$$v^i_{3,1,\rightarrow,2} \ + \ v^i_{3,2,\rightarrow,1} \leq 1$$
$$v^i_{3,1,\rightarrow,2} \ + \ v^i_{3,1,\uparrow,1} \ \leq 1$$

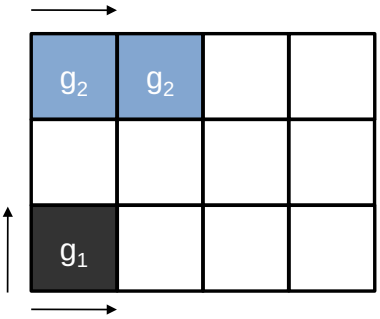

Figure 6.4: An example of a design with two memory channel groups.

Since the placement of memory channels affects the placement of memory controllers, we must add constraints to guarantee that a memory controller tile is adjacent to a memory channel group. Assuming the configuration shown in Figure 6.3(a), the following constraints guarantee adjacency between memory controller tiles and memory channel groups. To maintain a linear formulation, we use two different constraints files in which all the constraints are similar except for those enforcing adjacency of memory controller and memory channel. To implement a logical OR operation between the two given configurations would require nonlinear constraints which degrade solver performance. Hence, we construct two separate constraints files corresponding to the configurations in Figures 6.3(a) and (b), respectively.

$$\sum_{r}\sum_{c}\sum_{d}\sum_{g} v^{i}_{r,c,\rightarrow,g} \geq u^{i}_{r_{MC}c_{MC}3} \tag{6.15}$$
$$\forall i;\ \ r_{MC} = R;\ \ 2 \leq c_{MC} \leq C-1$$

$$\sum_{r}\sum_{c}\sum_{d}\sum_{g} v^{i}_{r,c,\uparrow,g} \geq u^{i}_{r_{MC}c_{MC}3} \tag{6.16}$$
$$\forall i;\ \ c_{MC} = 1;\ \ 2 \leq r_{MC} \leq R-1$$

$$\sum_{r}\sum_{c}\sum_{d}\sum_{g} v^{i}_{r,c,\rightarrow,g} \geq u^{i}_{r_{MC}c_{MC}3} \tag{6.17}$$
$$\forall i;\ \ r_{MC} = 1;\ \ 2 \leq c_{MC} \leq C-1$$

Here, r_{MC} and c_{MC} respectively denote the row and column in which a given memory controller is placed. Constraints (6.15), (6.16), and (6.17) ensure that memory channels are adjacent to memory controller tiles.

Due to placement of memory channels and I/O devices at the boundary of the design, core and memory controller blocks can only be placed in the inner tiles (cf. white tiles in Figures 6.3(a) and (b)). Hence, we add Constraint (6.18) to avoid the placement of cores and memory controllers on the boundary of the product. Also, the memory controller tiles can only be placed in rows and columns adjacent to the boundary, as permitted by the memory channel-I/O configuration (shown in Constraint (6.19)).

$$u^i_{r,c,2} = 0, \;\; u^i_{r,c,3} = 0 \tag{6.18}$$
$$\forall i; \;\; r = 1; \;\; r = R; \;\; c = 1; \;\; c = C$$

$$u^i_{r,c,3} = 0 \tag{6.19}$$
$$\forall i; \;\; 3 \leq r \leq R-2; \;\; 3 \leq c \leq C-2$$

In the next section, we propose additional constraints to enable effective floorplan design space exploration across multiple products.

6.4 Power- and Performance-Driven Floorplan Design Space Exploration

Our proposed multi-product floorplan optimization approach simultaneously optimizes floorplans of multiple CMP products subject to an upper bound on the sum of all product floorplan half-perimeters. To accomplish this, we extend the previous formulation as follows.

- We allow the number of cores and memory controllers for each product to vary in a given range.
- We add constraints on the maximum number of memory controllers in a given row or column.
- We consider different width and height values for different building blocks, to support heterogeneous building block sizes.

6.4.1 Extension 1: Power Exploration

Early in the design cycle, complete design specifications are often not available, and require efficient design space exploration to achieve convergence. Our first extension allows the numbers of cores and memory controllers for each product to vary in given ranges. To determine the numbers of cores and memory controllers, we define a power budget for product P_i, denoted as p^i_{budget}, as

$$p_{core} \sum_r \sum_c u^i_{r,c,2} \; + \; p_{MC} \sum_r \sum_c u^i_{r,c,3} \leq p^i_{budget} \qquad (6.20)$$
$$\forall r, c, i$$

where p_{core} and p_{MC} respectively denote core and memory controller power. In applying the above constraint, we ensure that all empty tiles are forced, i.e., we always add more resources if the power budget allows allows. For example, assume that $p_{core} = 2$ W, $p_{MC} = 1$ W, and $p^1_{budget} = 8$ W; Figures 6.5(a) and (b) show two possible configurations for P_1; however, we prefer the configuration of Figure 6.5(b) since it has an additional core without exceeding the power budget. With this in mind, we modify our original objective to minimize the sum of half-perimeters and number of empty tiles over all products.

$$\textbf{Minimize:} \sum_i (H^i \; + \; W^i) \; + \; \sum_r \sum_c u^i_{r,c,1}$$

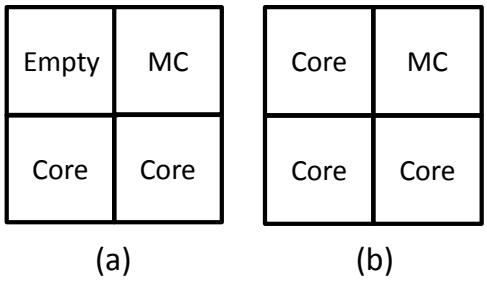

Figure 6.5: Two possible configurations for a given product.

6.4.2 Extension 2: Performance Enhancement

Since memory controllers receive all the traffic to the memory, they become hotspots when there are many cache misses. Hence, placing many memory controllers in a given row (column) can cause congestion on that row (column), resulting in performance degradation. To alleviate this problem, we add Constraints (6.21) and (6.22), which upper-bound the number of memory controllers that can be placed on a given row or column.

$$\sum_{c} u^i_{r,c,3} \leq N^i_{3,max-r} \;\; \forall i, r \tag{6.21}$$

$$\sum_{r} u^i_{r,c,3} \leq N^i_{3,max-c} \;\; \forall i, c \tag{6.22}$$

where $N^i_{3,max-r}$ and $N^i_{3,max-c}$ respectively denote maximum number of memory controllers in row r and column c of product P_i.

6.4.3 Extension 3: Heterogeneous Resource Support

Finally, to support heterogeneous resources, we consider different width and height values for cores, memory controllers, and memory channel tiles. We add Constraints (6.23) and (6.24) to find the maximum height (width) in a given row (column), and we modify our area computation to capture the difference in core and memory controller dimensions as shown below (Constraints (6.25) and (6.26)).

$$h^i_r \geq u^i_{r,c,2} \times h_{core} \;\; \forall i, r; \;\; 1 \leq c \leq C \tag{6.23}$$

$$h^i_r \geq u^i_{r,c,3} \times h_{MC} \;\; \forall i, r; \;\; 1 \leq c \leq C$$

$$w^i_r \geq u^i_{r,c,2} \times w_{core} \;\; \forall i, c; \;\; 1 \leq r \leq R \tag{6.24}$$

$$w^i_c \geq u^i_{r,c,3} \times w_{MC} \;\; \forall i, c; \;\; 1 \leq r \leq R$$

$$H^i = \sum_{r} h^i_r \;\; \forall i, r, c \tag{6.25}$$

$$W^i = \sum_{c} w^i_c \;\; \forall i, r, c \tag{6.26}$$

where h_{core}, w_{core}, h_{MC}, and w_{MC} denote core height, core width, memory controller height, and memory controller width, respectively. Further, h_r and w_c respectively denote the height of row r and width of column c.

6.5 Evaluation and Discussion

In this section, we describe our developed infrastructure for the proposed multi-product CMP floorplan optimization framework, and discuss our experimental results.

6.5.1 Experimental Setup

Our multi-product floorplan optimization framework (1) reads in a floorplan description file, (2) generates the corresponding ILP constraints, (3) feeds the constraints to CPLEX [5], and (4) generates visual representations of each product's floorplan. We use Perl scripting ($<$ 2000 lines of code) to read the floorplan description file and generate the corresponding ILP constraints.

The floorplan description file includes information about (1) grid size, (2) minimum and maximum numbers of core and memory controller tiles for each product, (3) maximum number of memory controller tiles in a given row or column of a product, (4) core and memory controller dimensions, and (5) building block power values, and power budget for each product. The grid size is independent of the number of tiles in the largest product (e.g., it is greater than or equal to the biggest product's R and C). Our script generates an ASCII file which contains the corresponding ILP constraints for the given floorplan description file. To show the user the chopping operations that have taken place from P_i to P_j, we generate visual representations of all the intermediate products between P_i and P_j. Our implementation also enables the designers to obtain all the possible solutions from the pool of solutions derived by CPLEX.

6.5.2 Experimental Results

To validate our approach, we investigate an example problem and show that our approach obtains the expected optimal solution. Figure 6.6 shows an example testcase

with two products: (1) P_1 with 14 cores and two MCs, and (2) P_2 with six cores and two MCs. In this example, h_{core} and h_{MC} are 3 and 1 units, and w_{core} and w_{MC} are both 4 units. We assume a configuration similar to that of Figure 6.3(a). For P_1, we must place the MCs on the boundary, but since memory controllers and cores have the same width, it will not help the area minimization to place both of them in the left column. In addition, it will not be beneficial to place one of them in either the bottom or top row and the other one in the left column. This is because we will not benefit from memory the controller's smaller height in minimizing area when going to smaller products. However, if we place both of them in either the top or bottom row (Figure 6.6(a)) we may achieve a configuration with a smaller row after corresponding chopping operations.

Figures 6.6(b), (c), and (d) show possible configurations for P_2. Note that there are other symmetric solutions which are identical to the configurations shown. Given core and memory controller width and height values, the configuration in Figure 6.6(b) achieves the smallest half-perimeter (resp. area) among all three configurations. We observe that our proposed method has appropriately picked the solution shown in Figure 6.6(b). We have also verified our approach against real industry prototypes, and have obtained the solutions that were manually developed by the designers in a few seconds, compared with weeks of designer effort. Due to the proprietary nature of these testcases, they cannot be shown them in this thesis.

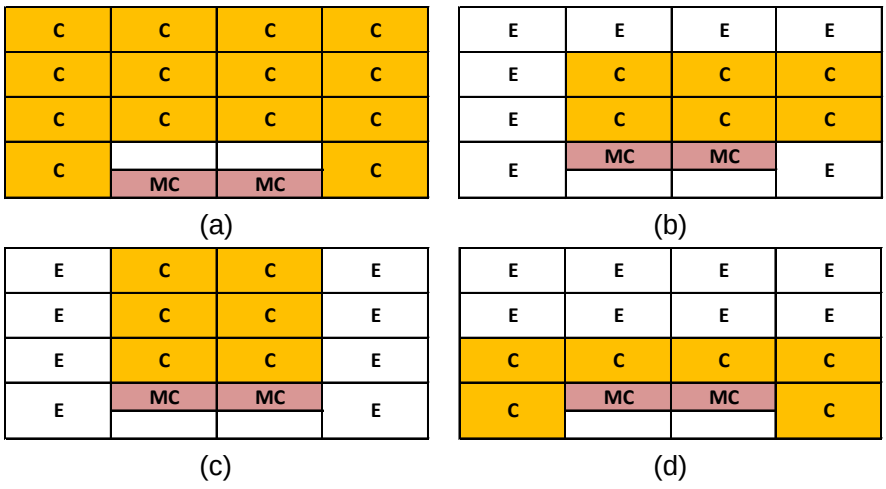

Figure 6.6: An example testcase with two products.

Our experiments use three testcases with varying numbers of cores, memory controllers and memory channels, as shown in Table 6.1. All of our testcases are representative of future-generation CMPs. The floorplans of all products are simultaneously obtained after solving the associated ILP problem. As noted above, in our problem formulation the grid size is independent from total number of tiles in the largest product. This means that, e.g., if the largest product has 20 tiles, the grid size need not be 4×5 or 5×4, but can be any size that contains the largest product.[3] This allows different solutions to be efficiently explored, which is of interest in light of the heterogeneous nature of the resources. In our testcases, core (MC) tiles have 3 (3) units of width and 2 (1) units of height. The width (height) of a column (row) is determined by the width (height) of the largest building block in that column (row). As mentioned above, memory controllers can reduce chip height if all of them are placed in the same row.

Figure 6.7 shows Testcase 2 with its corresponding products. In the figure, C, MC, MCH and E indicate core, memory controller, memory channel and empty tiles, respectively. The final floorplans of all three products are shown. To derive P_2 from P_1, Column 6 and three core tiles at (8,5), (8,7) and (2,7) are chopped from P_1. Subsequently, Column 5 and two memory controllers at (8,3) and (8,6) are chopped from P_2 to obtain P_3. After each product is chopped, necessary column and row shiftings are required to obtain the arrangements shown.

Table 6.1: Our experimental testcases.

Testcase	#products	#cores	#MCs	#MCHs
1	3	50	12	18
2	3	81	18	26
3	3	104	18	28

Table 6.2 shows our three testcases and their corresponding numbers of binary variables and constraints, as well as the CPU needed to solve the ILP instances. From Table 6.2, we observe that our approach has good scalability with respect to the number of building blocks in a given design. Runtimes for smaller testcases are on the order of

[3]Selecting a very large grid size will increase the runtime due to additional constraints for the extra tiles.

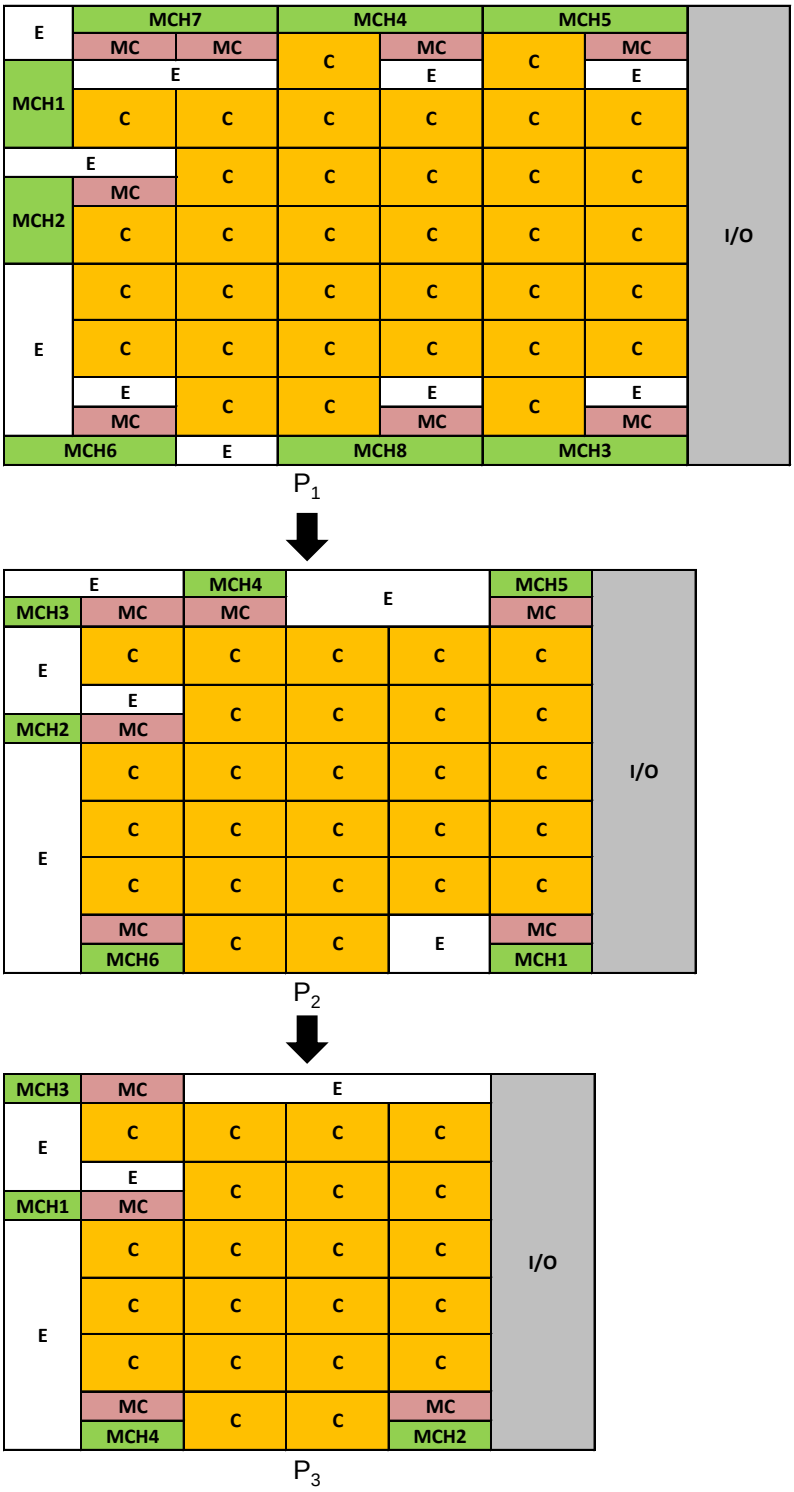

Figure 6.7: Testcase 2 with three different products and varying number of cores, memory controllers, and memory channels.

a few seconds to a few minutes. In addition, our method can be easily run on multiple computing resources if multiple product configurations need to be explored.

Table 6.2: Complexity and runtime of our approach.

Testcase	#binary variables	#constraints	CPU runtime (sec)
1	595	3014	687
2	896	6204	4744
3	1089	7218	14936

6.6 Conclusions

In this chapter, we propose a simultaneous floorplan optimization framework for multiple CMP products. We define the concept of a *choppable* floorplan, which enables us to easily derive the floorplan of smaller products from those of the larger ones through simple chopping operations. In our approach, we support (1) multiple building block types (i.e., core, memory controller, memory channel, etc.), (2) design space exploration of achievable floorplans (under a given power-performance budget), and (3) heterogeneous resources (i.e., different height or width values for each building block). We observe that our approach efficiently finds choppable floorplans across multiple products to reduce individual product re-design overheads and design turnaround times.

Looking into the future, CMPs are expected to grow in terms of both number of resources per product, and total number of products due to increased diversity of market demands. Hence, faster heuristics are needed to more efficiently perform floorplan optimizations. In addition, CMP designs will become more heterogeneous as newer building blocks are added, and the possibility of having more than one building block in a given tile must be enabled to avoid unnecessary white space and further minimize area.

6.7 Acknowledgments

This chapter is in part a reprint of:

- Marco A. Escalante, Andrew B. Kahng, Michael Kishinevsky, Umit Y. Ogras, **Kambiz Samadi** and Emily Shriver, "Multi-Product Floorplan Optimization Framework for Chip Multiprocessors", *draft in submission*, November 2010.

I would like to thank my coauthors Dr. Marco A. Escalante, Dr. Michael Kishinevsky, Dr. Umit Y. Ogras, Dr. Emily Shriver, and Prof. Andrew B. Kahng. I would also like to thank Peng Du for his help on extending the basic problem formulation.

Chapter 7

Conclusions

Networks-on-Chip (NoCs) are an important class of interconnection fabric for both general-purpose chip multiprocessors and application-specific multiprocessor systems-on-chip. Increased communication between cores to facilitate high core utilization requires high-performance NoCs. At the same time, power is a first-order design constraint, with more stringent future limits per the 2009 ITRS [6]. Hence, NoC power must be minimized while meeting performance objectives. High-quality, early-stage architectural design exploration – based on estimators as well as optimizers – is needed to understand the power-delay-area tradeoffs for on-chip networks. However, existing architectural estimation models, in one way or another, assume a specific architecture and underlying circuit implementation. Furthermore, existing NoC optimizations do not incorporate traffic behavior of the target applications. These two failings limit the quality of NoC design space exploration, and result in designs that are not well-matched to the corresponding applications.

To address the above two shortcomings, this thesis focuses on both the estimation and the optimization of on-chip networks. In the context of estimation, we propose reproducible methodologies to derive architecture-level power, performance and area models for on-chip routers and interconnects. In our modeling efforts, the goal is to close the accuracy gap between low-level (e.g., device- and circuit-level) and architecture-level models. Architecture-level models are usually very crude, but are relatively easy to use for early-stage design space exploration. On the other hand, low-level models are more accurate, but are not usable in high-level (e.g., architecture- and

system-level) design space exploration. Our proposed models incorporate low-level, technology-dependent) parameters to enhance the accuracy of the existing architecture-level models. We also provide a publicly available framework: our models can be freely downloaded and integrated in to any architecture- and system-level NoC optimization tool. We have shown in this thesis that the solution quality of high-level NoC optimization tools is very sensitive to the accuracy underlying models.

In another direction, we propose the use of machine learning-based nonparametric regression techniques to model on-chip routers and interconnect power, performance and area. The motivation behind this approach comes from the fact that accurate architecture-level models require comprehensive understanding of the underlying architecture and circuit implementation of digital blocks. However, as the number of parameters and the interactions between them increases, this becomes a nontrivial task. Nonparametric regression methods are data-driven in nature, which means they generate models whose functional form will be determined by the data itself. This allows designers to decouple understanding of the underlying architecture and circuit implementation from modeling efforts.

In the context of architecture-level optimization, we propose a trace-driven paradigm in which NoC configuration and other NoC optimizations are driven by application traces. We expose application specifics to obtain a better hardware solution. The design of an on-chip network spans various building blocks: topology, routing, flow control, router microarchitecture, and link architecture. Among these, the router microarchitecture is of primary importance since it directly impacts communication latency. We propose trace-driven virtual channel allocation heuristics that reduce buffering requirements with no penalty on performance.

Looking to the future, power will become the most pressing constraint in the design of on-chip networks. New low-power on-chip router and link architectures will be needed. An interesting direction is to explore other machine learning-based approaches that exploit active learning to model complex architectures and circuit implementations with high-dimensional parameter spaces. Furthermore, to proliferate trace-driven techniques in early-stage design exploration, computationally efficient and easily parallelizable optimization algorithms should be developed.

Bibliography

[1] *ARM Integrated Multiprocessor Core*, http://www.arm.com/ .

[2] *Atrenta SpyGlass-Physical User's Manual*, 2010, http://www.atrenta.com/ .

[3] *Cadence Design Exchange Format Reference Manual*, http://www.cadence.com/ .

[4] *Cadence SOC Encounter User's Manual*, http://www.cadence.com/ .

[5] *ILOG CPLEX User's Manual*, http://www.ilog.com/products/cplex/ .

[6] *International Technology Roadmap for Semiconductors*, http://www.itrs.net/ .

[7] *Cadence LEF/DEF Language Reference*, http://openeda.si2.org/projects/lefdef/ .

[8] *Synopsys Liberty File Format User's Guide*, http://www.synopsys.com/ .

[9] *MARS User's Guide*, http://www.salfordsystems.com/mars.php/ .

[10] *Nehalem Microarchitecture*, http://en.wikipedia.org/wiki/Nehalem_(microarchitecture)/ .

[11] *Netmaker*, http://www-dyn.cl.cam.ac.uk/∼rdm34/wiki/index.php/ .

[12] *ORION 2.0*, http://vlsicad.ucsd.edu/ORION/ .

[13] *Predictive Technology Model*, http://www.eas.asu.edu/∼ptm/ .

[14] *Synopsys Design Compiler User's Manual*, http://www.synopsys.com/ .

[15] *Synopsys HSPICE User's Manual*, http://www.synopsys.com/ .

[16] *Synopsys PrimeTime User's Manual*, http://www.synopsys.com/ .

[17] *UCSD VLSI CAD Laboratory: NoC Modeling Resources*, http://vlsicad.ucsd.edu/NOCModeling/ .

[18] *Virtutech AB. Simics* full system simulator, http://www.virtutech.com/ .

[19] S. N. Adya and I. Markov, "Fixed-Outline Floorplanning: Enabling Hierarchical Design", *IEEE Transactions on Very Large Scale Integration Systems* 11(6) (2003), pp. 1120-1135.

[20] N. Agarwal, L.-S. Peh and N. K. Jha, "GARNET: A Detailed Interconnection Network Model Inside a Full-System Simulation Framework", *Technical Report CE-P08-001*, Dept. of Electrical Engineering, Princeton University, 2008.

[21] K. M. B. Ahin, P. Patra and F. N. Najm, "ESTIMA: An Architectural-Level Power Estimator for Multi-Ported Pipelined Register Files", *Proc. International Symposium on Low Power Electronics and Design*, 2003, pp. 294-297.

[22] M. Al Faruque and J. Henkel, "Minimizing Virtual Channel Buffer for Routers in On-Chip Communication Architectures", *Proc. Design, Automation and Test in Europe*, 2008, pp. 1238-1243.

[23] R. Arunachalam, F. Dartu and L. Pileggi, "CMOS Gate Delay Models for General RLC Loading", *Proc. IEEE International Conference on Computer Design*, 1997, pp. 224-229.

[24] H. Bakoglu, *Circuits, Interconnections and Packaging for VLSI*, Addison-Wesley, 1990.

[25] A. Banerjee, R. Mullins and S. Moore, "A Power and Energy Exploration of Network-on-Chip Architectures", *Proc. ACM/IEEE International Symposium on Networks-on-Chip*, 2007, pp. 163-172.

[26] K. Banerjee, S. J. Souri, P. Kapur and K. C. Saraswat, "3-D ICs: A Novel Chip Design for Improving Deep-Submicrometer Interconnect Performance and Systems-on-Chip Integration", *Proc. IEEE* 89(5) (2001), pp. 602-633.

[27] N. Banerjee, P. Vellanki and K. S. Chatha, "A Power and Performance Model for Network-on-Chip Architectures", *Proc. Design, Automation and Test in Europe*, 2004, pp. 1250-1255.

[28] L. Benini and G. D. Micheli, "A New SoC Paradigm", *IEEE Computer* 35(1) (2002), pp. 70-78.

[29] S. Bhat, "Energy Models for Network-on-Chip Components", *M.S. Thesis*, Dept. of Mathematics and Computer Science, Royal Institute of Technology, Eindhoven, 2005.

[30] A. Bona, V. Zaccaria and R. Zafalon, "System Level Power Modeling and Simulation of High-End Industrial Network-on-Chip", *Proc. Design, Automation and Test in Europe*, 2004, pp. 318-323.

[31] C. Bienia, S. Kumar, J. P. Singh and K. Li, "The PARSEC Benchmark Suite: Characterization and Architectural Implications", *Technical Report TR-811-08*, Dept. of Electrical Engineering, Princeton University, 2008.

[32] L. C. Briand, B. Freimut and F. Vollei, "Using Multiple Adaptive Regression Splines to Support Decision Making in Code Inspection", *Journal of Systems and Software* 73(2) (2004), pp. 205-217.

[33] A. E. Caldwell, A. B. Kahng and I. L. Markov, "Improved Algorithms for Hypergraph Bipartitioning", *Proc. IEEE Asia and South Pacific Design Automation Conference*, 2000, pp. 661–666.

[34] Y. Cao, C. M. Hu, X. J. Huang, A. B. Kahng, S. Muddu, D. Stroobandt and D. Sylvester, "Effects of Global Interconnect Optimizations on Performance Estimation of Deep Submicron Design", *Proc. IEEE/ACM International Conference on Computer-Aided Design*, 2000, pp. 56-61.

[35] L. P. Carloni, A. B. Kahng, S. Muddu, A. Pinto, K. Samadi and P. Sharma, "Interconnect Modeling for Improved System-Level Design Optimization", *Proc. IEEE Asia and South Pacific Design Automation Conference*, 2008, pp. 258-264.

[36] J. Chan and S. Parameswaran, "NoCEE: Energy Macro-Model Extraction Methodology for Network on Chip Routers", *Proc. IEEE/ACM International Conference on Computer-Aided Design*, 2005, pp. 254-259.

[37] V. Chandra, A. Xu, H. Schmit and L. Pileggi, "An Interconnect Channel Design Methodology for High-Performance Integrated Circuits", *Proc. Design, Automation and Test in Europe*, 2004, pp. 1138-1143.

[38] X. Chen and L.-S. Peh, "Leakage Power Modeling and Optimization in Interconnect Networks", *Proc. International Symposium on Low Power Electronics and Design*, 2003, pp. 90-95.

[39] P. Christie and D. Stroobandt, "The Interpretation and Application of Rent's Rule", *IEEE Transactions on Very Large Scale Integration Systems* 8(6) (2000), pp. 639-648.

[40] J. Cong and D. Z. Pan, "Interconnect Delay Estimation Models for Synthesis and Design Planning", *Proc. IEEE Asia and South Pacific Design Automation Conference*, 1999, pp. 507-510.

[41] M. Dall'Osso, G. Biccari, L. Giovannini, D. Bertozzi and L. Benini, "Xpipes: A Latency Insensitive Parameterized Network-on-Chip Architecture for Multiprocessor SoCs", *Proc. IEEE International Conference on Computer Design*, 2003, pp. 536-539.

[42] W. J. Dally and C. L. Seitz, "The Torus Routing Chip", *Journal of Distributed Computing* 1(3) (1986), pp. 187-196.

[43] W. J. Dally and B. Towles, "Route Packets, Not Wires: On-Chip Interconnection Networks", *Proc. ACM/IEEE Design Automation Conference*, 2001, pp. 684-689.

[44] W. J. Dally and B. Towles, *Principles and Practices of Interconnection Networks*, Morgan Kaufmann, 2004.

[45] F. Dartu, N. Menezes and L. Pileggi, "Performance Computation for Precharacterized CMOS Gates with RC Load", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 15(5) (1996), pp. 544-553.

[46] P. Du, X. Hu, S.-H. Weng, A. Shayan, X. Chen, A. Ege Engin and C.-K. Cheng, "Worst-Case Noise Prediction With Non-Zero Current Transition Times for Early Power Distribution System Verification", *Proc. International Symposium on Quality Electronic Design*, 2010, pp. 624-631.

[47] D. E. Duarte, N. Vijaykrishnan and M. J. Irwin, "A Clock Power Model to Evaluate Impact of Architectural and Technology Optimizations", *IEEE Transactions on Very Large Scale Integration Systems* 10(6) (2002), pp. 844-855.

[48] N. Enright-Jerger and L.-S. Peh, *On-Chip Networks*, Synthesis Lectures, Morgan-Claypool Publishers, 2009.

[49] N. Eisley, V. Soteriou and L. S. Peh, "High-Level Power Analysis for Multi-Core Chips", *Proc. International Conference on Compilers, Architecture and Synthesis for Embedded Systems*, 2006, pp. 389-400.

[50] C. J. Glass and L. M. Ni, "The Turn Model for Adaptive Routing", *Proc. International Symposium on Computer Architecture*, 1992, pp. 874-902.

[51] M. Graziano, M. R. Casu, G. Masera, G. Piccinini and M. Zamboni, "Effects of Temperature in Deep-Submicron Global Interconnect Optimization in Future Technology Nodes", *Microelectronics Journal* 35(10) (2004), pp. 849-857.

[52] J. H. Friedman, "Multivariate Adaptive Regression Splines", *Annals of Statistics* 19(1) (1991), pp. 1-66.

[53] M. Hashimoto, J. Yamaguchi and H. Onodera, "Timing Analysis Considering Spatial Power/Ground Level Variation", *Proc. IEEE/ACM International Conference on Computer-Aided Design*, 2004, pp. 814-820.

[54] S. Heo and K. Asanovic, "Power-Optimal Pipelining in Deep Submicron Technology", *Proc. International Symposium on Low Power Electronics and Design*, 2004, pp. 218-223.

[55] S. Heo and K. Asanovic, "Replacing Global Wires With an On-Chip Network: A Power Analysis", *Proc. International Symposium on Low Power Electronics and Design*, 2005, pp. 369-374.

[56] R. Ho, K. W. Mai and M. A. Horowitz, "The Future of Wires", *Proc. IEEE* 89(4) (2001), pp. 490-504.

[57] Y. Hoskote, S. Vangal, A. Singh, N. Borkar and S. Borkar, "A 5-GHz Mesh Interconnect for a Teraflops Processor", *IEEE Micro* 27(5) (2007), pp. 51-61.

[58] J. Hu and R. Marculescu, "DyAD – Smart Routing for Networks-on-Chip", *Proc. ACM/IEEE Design Automation Conference*, 2004, pp. 260-263.

[59] J. Hu and R. Marculescu, "Energy- and Performance-Aware Mapping for Regular NoC Architectures", *IEEE Transactions on Computer Aided-Design of Integrated Circuits and Systems* 24(4) (2005), pp. 551-562.

[60] J. Hu, U. Y. Ogras and R. Marculescu, "System-Level Buffer Allocation for Application-Specific Networks-on-Chip Router Design", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 25(12) (2006), pp. 2919-2933.

[61] T.-C. Huang, U. Y. Ogras and R. Marculescu, "Virtual Channel Planning for Networks-on-Chip", *Proc. International Symposium on Quality Electronic Design*, 2007, pp. 879-884.

[62] D. A. Ilitzky, J. D. Hoffman, A. Chun and B. P. Esparza, "Architecture of the Scalable Communications Core's Network on Chip", *IEEE Micro* 27(5) (2007), pp. 62-74.

[63] A. B. Kahng, B. Li, L.-S. Peh and K. Samadi, "ORION 2.0: A Fast and Accurate NoC Power and Area Model for Early-Stage Design Space Exploration", *Proc. Design, Automation and Test in Europe*, 2009, pp. 423-428.

[64] A. B. Kahng, B. Lin, K. Samadi and R. Sunkam Ramanujam, "Trace-Driven Optimization of Networks-on-Chip Configurations", *Proc. ACM/IEEE Design Automation Conference*, 2010, pp. 437-442.

[65] M. B. Kamble and K. Ghose, "Analytical Energy Dissipation Models for Low Power Caches", *Proc. International Symposium on Low Power Electronics and Design*, 1997, pp. 143-148.

[66] A. K. Kodi, A. Sarathy and A. Louri, "Design of Adaptive Communication Channel Buffers for Low-Power Area-Efficient Network-on-Chip Architectures", *Proc. ACM/IEEE Symposium on Architecture for Networking and Communications Systems*, 2007, pp. 47-56.

[67] A. K. Kodi, A. Sarathy and A. Louri, "Adaptive Channel Buffers in On-Chip Interconnection Networks – A Power and Performance Analysis", *IEEE Transactions on Computers* 57(9) (2008), pp. 1169-1181.

[68] P. Kongetira, K. Aingaran and K. Olukotun, "Niagara: A 32-Way Multithreaded SPARC Processor", *IEEE Micro* 25(2) (2005), pp.21-29.

[69] A. Kumar, P. Kundu, A. Singh, L.-S. Peh and N. K. Jha, " A 4.6Tbits/s 3.6GHz Single-cycle NoC Router with a Novel Switch Allocator in 65nm CMOS", *Proc. IEEE International Conference on Computer Design*, 2007, pp. 63-70.

[70] A. Kumar, L.-S. Peh, P. Kundu and N. K. Jha, "Express Virtual Channels: Towards the Ideal Interconnection Fabric", *Proc. International Symposium on Computer Architecture*, 2007, pp. 150-161.

[71] S. E. Lee and N. Bagherzadeh, "A High Level Power Model for Network-on-Chip (NoC) Router", *Computers and Electrical Engineering* (35) (2009), pp. 837-845.

[72] E. Long, W. R. Daasch, R. Madge and B. Benware, "Detection of Temperature Sensitive Defects Using ZTC", *Proc. VLSI Test Symposium*, 2004, pp. 185-190.

[73] S. Manolache, P. Eles and Z. Peng, "Buffer Space Optimization with Communication Synthesis and Traffic Shaping for NoCs", *Proc. Design, Automation and Test in Europe*, 2006, pp. 95-98.

[74] R. Marculescu and P. Bogdan, "The Chip is Network: Toward a Science of Network-on-Chip Design", *Foundations and Trends in Electronic Design Automation* 2(4) (2007), pp. 371-461.

[75] M. M. K. Martin, D. J. Sorin, B. M. Beckmann, M. R. Marty, M. Xu, A. R. Alameldeen, K. E. Moore, M. D. Hill and D. A. Wood, "Multifacet's General Execution-Driven Multiprocessor Simulator (GEMS) Toolset", *SIGARCH Computer Architecture News*, 33(4) (2005), pp. 92-99.

[76] F. Martorell, M. Pons, A. Rubio and F. Moll, "Error Probability in Synchronous Digital Circuits Due to Power Supply Noise", *Proc. International Conference on Design and Technology of Integrated Systems in Nanoscale Era*, 2007, pp. 170-175.

[77] P. Meloni, I. Loi, F. Angiolini, S. Carta, M. Barbaro, L. Raffo and L. Benini, "Area and Power Modeling for Networks-on-Chip With Layout Awareness", *Proc. International Conference on VLSI Design*, 2007, pp. 1-12.

[78] G. D. Micheli and L. Benini, "Networks On Chip: A New Paradigm for Systems on Chip Design", *Proc. Design, Automation and Test in Europe*, 2002, pp. 2-6.

[79] G. D. Micheli and L. Benini, *Networks on Chip*, Morgan Kaufmann, 2006.

[80] S. Murali and G. D. Micheli, “Bandwidth-Constrained Mapping of Cores onto NoC Architectures”, *Proc. Design, Automation and Test in Europe*, 2004, pp. 896-901.

[81] N. Ni, M. Pirvu and L. Bhuyan, “Circular Buffered Switch Design with Wormhole Routing and Virtual Channels”, *Proc. IEEE International Conference on Computer Design*, 1998, pp. 466-473.

[82] C. A. Nicopoulos, D. Park, J. Kim, N. Vijaykrishnan, M. S. Yusif and C. R. Das, “ViChaR: A Dynamic Virtual Channel Regulator for Network-on-Chip Routers”, *Proc. IEEE/ACM International Symposium on Microarchitecture*, 2006, pp. 333-346.

[83] P. R. Nuth and W. J. Dally, “The J-Machine Network”, *Proc. IEEE International Conference on Computer Design*, 1992, pp. 420423.

[84] T. Okumura, F. Minami, K. Shimazaki, K. Kuwada and M. Hashimoto, “Gate Delay Estimation in STA Under Dynamic Power Supply Noise”, *Proc. IEEE Asia and South Pacific Design Automation Conference*, 2010, pp. 775-780.

[85] G. Palermo and C. Silvano, “PIRATE: A Framework for Power/Performance Exploration of Network-on-Chip Architectures”, *Proc. International Workshop on Power and Timing Modeling, Optimization and Simulation*, 2004, pp. 521531.

[86] V. F. Pavlidis and E. G. Friedman, “Interconnect-Based Design Methodologies for Three-Dimensional Integrated Circuits”, *Proc. IEEE* 97(1) (2009), pp. 123-140.

[87] D. Pamunuwa, L.-R. Zheng and H. Tenhunen, “Maximizing Throughput Over Parallel Wire Structures in the Deep Submicrometer Regime”, *IEEE Transactions on Very Large Scale Integration Systems* 11(2) (2003), pp. 224-243.

[88] C. S. Patel, S. M. Chai, S. Yalamanchili and D. E. Schimmel, ”Power Constrained Design of Multiprocessor Interconnection Networks”, *Proc. IEEE International Conference on Computer Design*, 1997, pp. 408-416.

[89] D. Pham, S. Asano, M. Bolliger, M. N. Day, H. P. Hofstee, C. Johns, J. Kahle, A. Kameyama, J. Keaty, Y. Masubuchi, M. Riley, D. Shippy, D. Stasiak, M. Suzuoki, M. Wang, J. Warnock, S. Weitzel, D. Wendel, T. Yamazaki and K. Yazawa, “The Design and Implementation of a First-Generation CELL Processor”, *Proc. IEEE International Solid-State Circuits Conference*, 2005, pp. 184-185.

[90] L. Pillage and R. Rohrer, “Asymptotic Waveform Evaluation for Timing Analysis”, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 9(4) (1990), pp. 352-366.

[91] A. Pinto, A. Bonivento, A. L. Sangiovanni-Vincentelli, R. Passerone and M. Sgroi, "System Level Design Paradigms: Platform-Based Design and Communication Synthesis", *ACM Transactions on Design Automation of Electronic Systems* 11(3) (2006), pp. 537-563.

[92] A. Pinto, L. P. Carloni and A. L. Sangiovanni-Vincentelli, "A Methodology and an Open Software Infrastructure for Constraint-Driven Synthesis of On-Chip Communications", *Technical Report UCB/EECS-2007-130*, Dept. of Electrical Engineering and Computer Science, UC Berkeley, 2007.

[93] A. Raghunathan, N. K. Niraj and S. Dey, *High-Level Power Analysis and Optimization*, Kluwer, 1998.

[94] C. Ratzlaff and L. Pillage, "RICE: Rapid Interconnect Circuit Evaluation using AWE", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 13(6) (1994), pp. 763-776.

[95] S. M. Rossnagel and T. S. Kuan, "Alteration of Cu Conductivity in the Size Effect Regime", *Journal of Vacuum Science and Technology B* 22(1) (2004), pp. 240-247.

[96] M. Saint-Laurent and M. Swaminathan, "Impact of Power-Supply Noise on Timing in High-Frequency Microprocessors", *IEEE Transactions on Advanced Packaging* 27(1) (2004), pp. 135-144.

[97] K. Sankaranarayanan, S. Velusamy, M. Stan and K. Skadron, "A Case for Thermal-Aware Floorplanning at the Microarchitectural Level", *Journal of Instruction-Level Parallelism* (8) (2005), pp. 1-16.

[98] L. Shang, L.-S. Peh and N.-K. Jha, "Dynamic Voltage Scaling with Links for Power Optimization of Interconnection Networks", *Proc. IEEE International Symposium on High-Performance Computing Architecture*, 2003, pp. 91-102.

[99] M. Shao, M. Wong, H. Cao, Y. Gao, L.-P. Yuan, L.-D. Huang and S. Lee, "Explicit Gate Delay Model for Timing Evaluation", *Proc. ACM International Symposium on Physical Design*, 2003, pp. 32-38.

[100] S. X. Shi and D. Z. Pan, "Wire Sizing with Scattering Effect for Nanoscale Interconnection", *Proc. IEEE Asia and South Pacific Design Automation Conference*, 2006, pp. 503-508.

[101] A. M. Smith, G. A. Constantinides and P. Cheung, "Integrated Floorplanning, Module-Selection, and Architecture Generation for Reconfigurable Devices", *IEEE Transactions on Very Large Scale Integration Systems* 16(6) (2008), pp. 733-744.

[102] V. Soteriou, N. Eisley, H. Wang, B. Li and L.-S. Peh, "Polaris: A System-Level Roadmap for On-Chip Interconnection Networks", *Proc. IEEE International Conference on Computer Design*, 2006, pp. 134-141.

[103] V. Soteriou, R. Sunkam Ramanujam, B. Lin and L-S. Peh, "A High-Throughput Distributed Shared-Buffer NoC Router", *IEEE Computer Architecture Letters* 8(1) (2009), pp. 21-24.

[104] S. Sutanthavibul, E. Shragowitz and J. B. Rosen, "An Analytical Approach to Floorplan Design and Optimization", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 10(6) (1991), pp. 761-769.

[105] D. Sylvester and K. Keutzer, "A Global Wiring Paradigm for Deep Submicron Design", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 19(2) (2000), pp. 242-252.

[106] Y. Tamir and G. L. Frazier, "High-Performance Multi-Queue Buffers for VLSI Communications Switches", *Proc. International Symposium on Computer Architecture*, 1988, pp. 343-354.

[107] Y. Tamir and G. L. Frazier, "Dynamically-Allocated Multi-Queue Buffers for VLSI Communication Switches", *IEEE Transactions on Computers* 41(6) (1992), pp. 725-737.

[108] M. B. Taylor, J. Psota, A. Saraf, N. Shnidman, V. Strumpen, M. Frank, S. Amarasinghe, A. Agarwal, W. Lee, J. Miller, D. Wentzlaff, I. Bratt, B. Greenwald, H. Hoffmann, P. Johnson and J. Kim, "Evaluation of the Raw Microprocessor: An Exposed-Wire-Delay Architecture for ILP and Streams", *Proc. International Symposium on Computer Architecture*, 2004, pp. 2-13.

[109] L. P. Tedesco, N. Calzans and F. Moraes, "Buffer Sizing for Multimedia Flows in Packet-Switching NoCs", *Journal of Integrated Circuits and Systems* 3(1) (2008), pp. 46-56.

[110] S. R. Vangal, J. Howard, G. Ruhl, S. Dighe, H. Wilson, J. Tschanz, D. Finan, A. Singh, T. Jacob, S. Jain, V. Erraguntla, C. Roberts, Y. Hoskote, N. Borkar and S. Borkar, "An 80-Tile Sub-100-W TeraFLOPS Processsor in 65nm CMOS", *IEEE Journal of Solid-State Circuits* 43(1) (2008), pp. 29-41.

[111] G. Varatkar and R. Marculescu, "Traffic Analysis for On-Chip Networks Design of Multimedia Application", *Proc. ACM/IEEE Design Automation Conference*, 2002, pp. 795-800.

[112] G. Varatkar and R. Marculescu, "On-Chip Traffic Modeling and Synthesis for MPEG-2 Video Applications", *IEEE Transactions on Very Large Scale Integration Systems* 12(1) 2004, pp. 108-119.

[113] H. Wang, L.-S. Peh and S. Malik, "A Power Model for Routers: Modeling Alpha 21364 and InfiniBand Routers", *IEEE Micro* 23(1) (2003), pp. 26-35

[114] H. Wang, X. Zhu, L.-S. Peh and S. Malik, "Orion: A Power-Performance Simulator for Interconnection Networks", *Proc. IEEE/ACM International Symposium on Microarchitecture*, 2002, pp. 294-395.

[115] H. Wilson and M. Haycock, "A Six-Port 30-GB/s Non-Blocking Router Component Using Point-to-Point Simultaneous Bidirectional Signaling for High-Bandwidth Interconnect", *IEEE Journal of Solid-State Circuits* 36(12) (2001), pp. 1954-1963.

[116] D. F. Wong and C.-L. Liu, "A New Algorithm for Floorplan Design", *Proc. ACM/IEEE Design Automation Conference*, 1986, pp. 101-107.

[117] J. Xiang and L. He, "Full-Chip Multilevel Routing for Power and Signal Integrity", *INTEGRATION, the VLSI Journal* 40(3) (2007), pp. 226-234.

[118] P. Zarkesh-Ha, J. A. Davis, W. Loh and J. D. Meindl, "Prediction of Interconnect Fanout Distribution Using Rent's Rule", *Proc. ACM/IEEE International Workshop on System-Level Interconnect Prediction*, 2000, pp. 107-112.

[119] W. Zhang, Y. Zhu, W. Yu, A. Shayan, R. Wang, Z. Zhu and C.-K. Cheng, "Noise Minimization During Power-Up Stage for a Multi-Domain Power Network", *Proc. IEEE Asia and South Pacific Design Automation Conference*, 2009, pp. 391-396.

[120] W. Zhang, L. Zhang, A. Shayan, W. Yu, X. Hu, Z. Zhu, E. Engin and C.-K. Cheng, "On-Chip Power Network Optimization with Decoupling Capacitors and Controlled-ESRs", *Proc. IEEE Asia and South Pacific Design Automation Conference*, 2010, pp. 119-124.

[121] Y. Zhou and H. Leung, "Predicting Object-Oriented Software Maintainability Using Multivariate Adaptive Regression Splines", *Journal of Systems and Software* 80(8) (2007), pp. 1349-1361.

[122] V. Zyuban and P. Kogge, "The Energy Complexity of Register Files", *Proc. International Symposium on Low Power Electronics and Design*, 1998, pp. 305-310.