2017

High performance communication on reconfigurable clusters

BOSTON UNIVERSITY

COLLEGE OF ENGINEERING

Dissertation

HIGH PERFORMANCE COMMUNICATION ON RECONFIGURABLE CLUSTERS

by

JIAYI SHENG

B.S., Fudan University, 2012

Submitted in partial fulfillment of the

requirements for the degree of

Doctor of Philosophy

2017

Approved by

First Reader

Martin C. Herbordt, Ph.D.
Professor of Electrical and Computer Engineering

Second Reader

Michel Kinsy, Ph.D.
Assistant Professor of Electrical and Computer Engineering

Third Reader

Ayse K. Coskun, Ph.D.
Associate Professor of Electrical and Computer Engineering

Fourth Reader

Adrian Caulfield, Ph.D.
Principal Research Hardware Development Engineer
Microsoft Research

Step after step the ladder is ascended.

George Herbert

Acknowledgments

First and foremost, I want to thank my advisor, Prof. Martin Herbordt. During our five-year collaboration, he has offered me limitless help on my research. I feel respected because he has understood every single detail in my projects. I feel motivated because he has helped me come up with numerous brilliant research ideas. I feel touched because he has been thoughtful to my personal life. I feel honored and proud to have him as my Ph.D. advisor in my life.

I also want to thank other members in my dissertation defense committee. Prof. Michel Kinsy has helped me describe technical part of my dissertation more clearly and precisely. Prof. Ayse Coskun has assisted me to organize the big picture of my dissertation. Dr. Adrian Caulfield not only has offered invaluable three-month guidance in summer 2016, but also has helped me address several technical issues in my dissertation.

I sincerely thank my collaborators at the University of Florida, Prof. Alan George, Prof. Herman Lam, and Dr. Abhijeet Lawande. Without their substantial work on Novo-G#, I would not have my dissertation. Especially, I want to thank Dr. Abhijeet Lawande. During the five years, he has provided me all sorts of help including emails, documentation, code sharing, and even remote control to allow me to reproduce his work on the BU side. His assistance is one of the fundamental factors in my dissertation.

I am grateful to my supervisors at MediaTek, Inc, Dr. Yuan Lin and Dr. Henry Cox. I feel fortunate to spend two fantastic summers with them. Dr. Henry Cox is considerate and kind. He could always motivate and thrill me even if I made mistakes. Dr. Yuan Lin is sharp and wise. He could steer me in the right directions and provide me better research ideas.

I would also like to appreciate other professors at Boston University that help me

grow and make progress. I thank Prof. Douglas Densmore, Prof. David Starobinski, and Prof. Richard West for their remarkable classes. I thank Prof. Roscoe Giles for his great advice when I was his teaching assistant.

I am also grateful to my current CAAD lab mates, Chen Yang, Qingqing Xiong, Ahmed Sanaulah, and Rushi Patel. They gave me plenty of research advice and collaboration. I also want to thank the alumni including Tiansheng Zhang, Hao Chen, Chao Chen, Jie Meng, Raphael Landaverde, Hansen Zhang, and Ben Humphries for their help and encouragement.

Finally, I want to thank my family for their unconditional and limitless support on my Ph.D. career. My dear maternal grandpa passed away in my Ph.D. fourth year. I did not have the chance to go back to see him. I hope my Ph.D. degree could make him smile in paradise.

The research that forms the basis of this dissertation has been partially funded by NSF grants #CNS-1405695 and #CCF-1618303/7960, and through a grant from Microsoft Research.

HIGH PERFORMANCE COMMUNICATION ON RECONFIGURABLE CLUSTERS

JIAYI SHENG

Boston University, College of Engineering, 2017

Major Professor: Martin C. Herbordt, PhD
Professor of Electrical and Computer Engineering

ABSTRACT

High Performance Computing (HPC) has matured to where it is an essential third pillar, along with theory and experiment, in most domains of science and engineering. Communication latency is a key factor that is limiting the performance of HPC, but can be addressed by integrating communication into accelerators. This integration allows accelerators to communicate with each other without CPU interactions, and even bypassing the network stack. Field Programmable Gate Arrays (FPGAs) are the accelerators that currently best integrate communication with computation. The large number of Multi-gigabit Transceivers (MGTs) on most high-end FPGAs can provide high-bandwidth and low-latency inter-FPGA connections. Additionally, the reconfigurable FPGA fabric enables tight coupling between computation kernel and network interface.

Our thesis is that an application-aware communication infrastructure for a multi-FPGA system makes substantial progress in solving the HPC communication bottleneck. This dissertation aims to provide an application-aware solution for communication infrastructure for FPGA-centric clusters. Specifically, our solution demonstrates application-awareness across multiple levels in the network stack, including low-level

link protocols, router microarchitectures, routing algorithms, and applications.

We start by investigating the low-level link protocol and the impact of its latency variance on performance. Our results demonstrate that, although some link jitter is always present, we can still assume near-synchronous communication on an FPGA-cluster. This provides the necessary condition for statically-scheduled routing. We then propose two novel router microarchitectures for two different kinds of workloads: a wormhole Virtual Channel (VC)-based router for workloads with dynamic communication, and a statically-scheduled Virtual Output Queueing (VOQ)-based router for workloads with static communication. For the first (VC-based) router, we propose a framework that generates application-aware router configurations. Our results show that, by adding application-awareness into router configuration, the network performance of FPGA clusters can be substantially improved. For the second (VOQ-based) router, we propose a novel offline collective routing algorithm. This shows a significant advantage over a state-of-the-art collective routing algorithm.

We apply our communication infrastructure to a critical strong-scaling HPC kernel, the 3D FFT. The experimental results demonstrate that the performance of our design is faster than that on CPUs and GPUs by at least one order of magnitude (achieving strong scaling for the target applications). Surprisingly, the FPGA cluster performance is similar to that of an ASIC-cluster. We also implement the 3D FFT on another multi-FPGA platform: the Microsoft Catapult II cloud. Its performance is also comparable or superior to CPU and GPU HPC clusters. The second application we investigate is Molecular Dynamics Simulation (MD). We model MD on both FPGA clouds and clusters. We find that combining processing and general communication in the same device leads to extremely promising performance and the prospect of MD simulations well into the us/day range with a commodity cloud.

Contents

List of Tables

List of Figures

List of Abbreviations

ASIC	.....	Application-Specific Integrated Circuits
ALM	.....	Adaptive Logic Module
API	.....	Application Programming Interface
CC	.....	Clock Cycle
CCAR	.....	Credit Count Adaptive Routing
COTS	.....	Commercial Off-The-Shelf
DOR	.....	Dimensional Ordering Routing
DPC	.....	Direct Programmable Communication
DSP	.....	Digital Signal Processor
FFT	.....	Fast Fourier Transform
FIFO	.....	First In First Out
FPGA	.....	Field-Programmable Gate Array
FSM	.....	Finite State Machine
HDL	.....	Hardware Description Language
IP	.....	Intellectual Property
LFSR	.....	Linear-feedback shift register
MD	.....	Molecular Dynamics
MGT	.....	Multi-Gigabit Transceiver
NoC	.....	Network on Chip
O1TURN	.....	Orthogonal One-turn Routing
OCR	.....	Offline Collective Routing Algorithm
OSI	.....	Open System Interconnection
PCS	.....	Physical Coding Sublayer
PHY	.....	physical layer of the OSI model
PMA	.....	Physical Medium Attachment
RLB	.....	Randomized Load Balance Routing
ROMM	.....	Randomized, Oblivious, Multi-phase, Minimal
RPM	.....	Recursive Partitioning Multicast Algorithm
RTL	.....	Register Transfer Level
VC	.....	Virtual Channel
VOQ	.....	Virtual Output Queue

Chapter 1

Introduction

Field programming gate arrays (FPGAs) are widely used Commercial Off-The-Shelf (COTS) Integrated Circuits (ICs) whose logic is configurable, rather than being fixed as in a Central Processing Unit (CPU) or Graphics Processing Unit (GPU). FPGAs can be adapted to the application rather than the application being encoded for the processor. In this way, applications can be run with high efficiency and low power. On a chip, many thousands of memory elements and compute components can be interconnected with an application-specific network; the expectation is that, for any application, each of the compute components will then be able to perform useful work most cycles. With other devices such as GPUs, getting such high utilizations is rare outside certain application domains such as dense matrix operations (see, e.g. (Lee et al., 2010)).

FPGAs have historically been used as glue logic. In the 1980s they became large and complex enough to find niche applications, especially in signal processing, where they could often replace Application-Specific Integrated Circuits (ASICs) and Digital Signal Processors (DSPs). Another decade another core application, network routers, is found. This domain led high-end FPGAs to have a capability still unique among commodity ICs: many dedicated high-performance communication ports. From around 2000 until 2015 around 70% of high-end FPGAs were used in routers, 20% in signal and image processing, and the rest in a variety of applications including ASIC verification, in-satellite computing, triggers for high-energy physics

experiments, finance computations (especially in high-frequency trading), and many others. Several attempts were made to use FPGAs as accelerators for High Performance Computing (HPC); these showed some promise, especially for bioinformatics, but the emergence of GPUs around 2008 slowed this progress.

In the last two years, there has been a dramatic shift to FPGAs as central compute components in large-scale systems. The primary motivation has been the need for higher performance communication-related computation. In data centers, there has been a tremendous increase in the need for encryption, compression, network stack offload, and software-defined networking. Almost simultaneously several significant events occurred. Altera was acquired by Intel, which then predicted that 30% of all Cloud nodes would be equipped with FPGAs; IBM made a similar alliance with Xilinx to enhance their IBM POWER-based systems; Microsoft started Project Catapult and had been deploying FPGAs in a large fraction of their production servers; Amazon has announced the availability of FPGAs as part of its compute services; and Baidu is using large numbers of FPGAs in its datacenters for machine learning.

There are many reasons for this dramatic change. FPGAs blend the benefits of ASICs and general-purpose processors. They can be used to implement circuits like ASICs but also perform general-purpose computing like CPUs. FPGAs have the potential for higher performance and lower power consumption than CPUs. They offer lower non-recurrent engineering (NRE) costs, reduced development time, easier debugging, and lower risk, compared with ASICs. FPGAs use a small fraction of the power of GPUs and, in the latest generation, have a comparable computational (floating point) capability. And FPGAs retain their unique capability as communication processors.

This shift to FPGAs in the datacenter is having a ripple effect in all FPGA usage. Development tools and support are improving; costs have come down dramatically so

that, across product lines, CPUs, GPUs, and FPGAs are now all comparably priced. Besides the major companies already described, at least two dozen others are now making NICs with user-programmable FPGAs, most prominently Mellanox.

HPC is also facing increasing difficulty due to communication performance, in large part because of the continuation of Moore's Law: per-chip computational capability has been increasing faster than the capability of moving data on/off chip and through the network. As the gap between the speeds of the network and computing devices keeps increasing, network latency and bandwidth have become the first-order bottleneck of many applications. Applications that have been communication bound are becoming more so, and even applications where communication was not previously a limiting factor are becoming communication bound. All of this leads to problems in scalability, the ability to productively map applications to larger numbers of compute nodes. This is especially a problem with strong scaling, where the problem size is fixed, but more compute resources are applied. For example, achieving strong scaling of Molecular Dynamics (MD) simulations is particularly challenging. Part of the calculation, that of the long-range force, requires global all-to-all communication. As the cluster size increases, the packets must traverse longer distances while at the same time the budget for communication time decreases (due to the application of more nodes to the same amount of processing).

To summarize, FPGAs are promising as devices that can dramatically improve HPC performance. They are low power, high performance, and have unique communication capability. And their configurable fabrics allow for tight coupling of computation and communication and for creating application-aware communication hardware. **Advancing the state-of-the-art of FPGA-centric HPC clusters, especially by improving all aspects of communication, is the primary goal of this thesis.**

Before the recent wide-spread integration of FPGAs into cloud nodes, researchers have been building multi-FPGA systems for many years (Mencer et al., 2009; Tsoi and Luk, 2010; Baxter et al., 2007; Patel et al., 2006; Sass et al., 2007; Moore et al., 2012; Kono et al., 2012; George et al., 2016). Some of the early work in FPGA clusters was to exploit the economic advantage of replacing a large FPGA with several smaller ones. For example, a designer could always buy the biggest existing FPGA to fit their design. But the gain in on-chip resources does not increase linearly with the monetary cost (Markettos et al., 2014). Also, resources like the number of external memory interfaces and I/O pins do not scale with FPGA size often making multi-FPGA systems better solutions. Most recent FPGA clusters more conventionally seek to scale compute capability far beyond the capacity of single devices.

A multi-FPGA system has inherent advantages because FPGAs are the only COTS component that has native communication support, high-compute capability, low-power, and an installed application base at the same time (George et al., 2016). FPGAs have multi-gigabit transceiver (MGT) ports that allow FPGAs communicate with each other directly without passing through any other devices (Altera, 2014b; Altera, 2015; Xilinx, 2009). The computation logic on the FPGA is tightly coupled with network ports like MGTs. An MGT link could have an end-to-end latency as short as 100 ns and bandwidth as high as 40 Gbps (Sheng et al., 2015).

Until very recently, inter-FPGA communication mechanisms could be placed into one of two categories. In the first, the communication support only satisfies the requirement of a single application. This approach loses the flexibility of being extendable to other applications. In the second, inter-FPGA communication mostly consists of MPI primitives implemented using soft processors. This approach is generic; however, the software-based packetization and depacketization introduce high latency overhead. Also, this approach does not make use of the tight coupling of commu-

nication and computation on FPGA. In summary, communication support on these previous FPGA clusters has not addressed generality and efficiency at the same time.

The newly deployed Microsoft Catapult systems (Putnam et al., 2014; Caulfield et al., 2016) address both generality and efficiency. It is deployed in general purpose nodes in datacenters. It has in-FPGA hardware support for general communication operations. That effort, however, differs from ours in several ways which emerge from the difference in basic goals. Catapult is deployed to large datacenters and must conform to stringent datacenter requirements. The FPGAs are "bumps in the wire" in the NIC, and there is likely to be a substantially fixed shell (set of system hardware) not accessible to the user. Since the FPGAs retain a mission of supporting general communication, even when the FPGAs are used for application processing, reconfigurability is limited in both frequency and characteristics. In our target systems, there substantially more flexibility, including the possibility of direct FPGA-FPGA connections; also there will be the possibility of full integration of communication and computation with a much lighter weight shell. This final point is that in our target systems, nearly the entire FPGA fabric can be mapped into application-aware configurations to support communication, computing in the network, and communication/application interfaces, which is a major point in this thesis.

Our thesis is that an application-aware communication infrastructure for a FPGA cluster makes substantial progress in solving the HPC communication bottleneck issue. In this dissertation, we are investigating how to build a complete application-aware communication infrastructure for FPGA-clusters. We use the Novo-G#, with 64 FPGAs, as a testbed. The purpose of Novo-G# is to be both a production cluster and community infrastructure, especially for research in issues related to direct, inter-FPGA, communication. Our infrastructure, therefore, addresses generality and efficiency simultaneously. Our work covers the issues across multiple levels of the

hardware/software stack, all the way from low-level link protocols, through router architecture, routing algorithm, to application mapping and partitioning. A major distinction with previous work is that each contribution of our dissertation is integrated with application-awareness.

Any comprehensive investigation into communication for FPGA clusters must deal with four levels of work.

- The first is how to connect FPGAs physically, which includes issues such as choice of link protocol, dealing with timing issues, and mechanisms to interface applications with the MGTs.

- The second is how to implement the infrastructure at the network layer, which includes both router design and routing algorithms. It also includes support for higher level functions such as collectives.

- The third is how to specify the user/system interface that allows users to integrate their applications with the communication infrastructure. With the inherent reconfigurability of the FPGAs, it is desirable that the communication infrastructure is adaptive to applications. The adaptiveness means the design choices are flexible to meet different users' requirements.

- The fourth is how to create, map, and run applications on this infrastructure; this is essential for proof-of-concept, for validation, and for test. This is necessary because of the relative paucity of production FPGA cluster applications and because of the difficulty in porting existing applications to new, different clusters.

We have made contributions for all four levels, which are listed below. This work is being done in the context of creating the Novo-G# testbed based on a cluster of 64

FPGA nodes. All the results within 4-node scale are validated on a 4-node subsystem. All the other results with large scale are simulated using cycle-accurate simulators.

1. **Physical and Datalink Inter-FPGA Connections.** We have explored several MGT SerDes protocols that trade off various parameters; at the highest level this involves latency versus support for essential link-level functions. We have also measured the variance of MGT link latency. Our major contribution at this level is that, to our best knowledge, ours is the first work that investigates the impact of the MGT link latency variance on the performance of a real application.

   Our results show that the MGT link latency variance degrades the performance in all cases. However, the degree of degradation is manageable. The impact of this result is that, even relying solely on commercial link IP, we can model the FPGA cluster as having near-synchronous flow. This allows us to use routing methods, especially those involving application-aware and static scheduling, that had been regarded as impossible for clusters with distributed nodes.

2. **Communication Network Layer.** The network layer design has two major aspects: router microarchitecture and routing algorithms. We have made contributions in both. We propose two router designs for two different workloads. For applications dominated by dynamic communication pattern, we provide a Virtual Channel (VC)-based router design and the flow control mechanism related to it. For applications which have only static communication pattern, we extend the existing Virtual Output Queue (VOQ) router design with hardware support for two collective functions (multicast and reduction).

   Our general VC-based wormhole router is based on the *de facto* pipelined Network on Chip (NoC) router design (Dally and Towles, 2004). We make significant modifications to it due to the different hardware constraints between

FPGA clusters and NoCs. It supports five different dynamic-scheduled (online) routing algorithms and three switch arbitration policies. It is easily extended with additional routing algorithms and arbitration policies. Also, the entire design is fully parameterized, which can satisfy users' requirement with simpler changes such as flit width and maximum packet size.

We also propose a VOQ router with hardware support for collective operations. There are two major distinctions in this router. The first is that it extends the canonical four-stage router pipeline to a seven-stage pipeline to support multicast and reduction. The second distinction is that this router uses routing tables instead of online dynamic routing computation logic. Among the advantages of table-based routing is that it supports application-aware static-scheduled routing. The idea behind static-scheduled routing is to use *a priori*application knowledge to address network congestion. Surprisingly, to our best knowledge, there are no existing FPGA clusters that have used this method. Supporting statically-scheduled application-aware routing requires not only a novel router design but also new routing algorithms. We propose a new offline collective routing (OCR) algorithm for collective operations, which reduces network congestion and balances load on the network links (Sheng et al., 2017; Sheng et al., 2016). We compare this algorithm with the state-of-the-art online dynamic routing. Our results show that static-scheduled table-based routing has an advantage over online adaptive routing in most cases both in both communication latency and hardware resource utilization.

3. **On-FPGA User/System Interface.** Lack of user-friendliness is a well-known problem for FPGAs. A multi-FPGA system makes things even worse. There have been some attempts to make progress on this topic, but little of that work has been systematic. Mostly it has relied either on the use of softcore

processors (Peng et al., 2014; Schmidt et al., 2012) or on latency-insensitive (LI) channels (Fleming et al., 2012). These approaches achieve generality, but at the cost of significant performance degradation. None of them takes into account application characteristics. Our approach differs in that it exposes the high-speed communication interfaces while allowing for flexibility and optimizations.

In our VC-based wormhole router, we support five different routing algorithms and three kinds of switch arbitration policies. Also, the number of virtual channels linked to a network port can range from 2 to 9. As a result, the configurable router supports more than 100 different configurations. We find that there is no universal optimal configuration for all workloads. In fact, even a single workload may have different optimization goals, such as latency, throughput, or area cost, and thus a different preferred configuration. To address this problem, we make several contributions. First, we provide a cycle-accurate simulator that exactly matches the behavior of the RTL model. Second, we define a standard for the interface among hardware modules. As long as new hardware components follow this standard, we can quickly extend the router and simulator with them. Third, with the help of this simulator, we implement a framework to help users evaluate different router configurations for a given application, including support of various optimization goals. This application-aware framework exhaustively searches all the possible router configurations and outputs the one with the best performance for the selected metrics. Because of their inherent reconfigurability, FPGAs can quickly switch among different configurations. Our results show that the application-aware configurations can achieve significant improvements compared with the optimized router configuration without applying application-awareness.

4. **Applications for Multi-FPGA Systems.** Creating model applications is

crucial to this project. These are necessary for overall proof-of-concept and to test work in the other areas. There are currently very few applications for FPGA clusters, and these are generally not portable or publicly available. We have therefore developed our own.

We have investigated which applications are particularly suitable for mapping onto FPGA clusters (Meng et al., 2016), and find that the 3D FFT is a good candidate. We fully implement the 3D FFT and use it as a case study to show how an application is mapped onto our cluster and communication infrastructure. Also, as the FFT routing pattern is deterministic, we create a complete tool flow to help map applications onto FPGA clusters with static routing support. Also, we implement the 3D FFT on another Multi-FPGA platform, the Microsoft Catapult II (Caulfield et al., 2016). The architecture of the Microsoft Catapult II cloud places the accelerator (FPGA) as a bump-in-the-wire on the way to the network and thus promises a dramatic reduction in latency as layers of hardware and software are avoided. We then use various experiments to build a model for performance over a range of parameters. We have shown sensitivity to node count, problem size, and also predicted the relative benefit of future network enhancements. We also compare the performance of 3D-FFT implementation on Novo-G# and Catapult. The results not only show advantages over all the other previous 3D FFTs on CPU, GPU, and FPGA but also demonstrate comparable performance with 3D FFT on an ASIC cluster (Young et al., 2009).

The second application we investigate is Molecular Dynamics (MD). MD is an iterative application of Newtonian mechanics to ensembles of atoms and molecules (particles). 3D FFT is an essential kernel to compute the electrostatic forces, and it is on the critical path. We examine phased application elasticity,

i.e., the use of a reduced set of nodes for the 3D FFT in MD. We find that, for the FFT phase within Molecular Dynamics, such contraction is beneficial with a 13%-14% performance improvement. Turning to MD, we show how this elasticity can be integrated into the existing data transformation to hide its communication overhead and increase the performance benefit to 16%-29%. Using these and other results, we model MD on FPGA-enhanced clouds and clusters. We find that combining processing and general communication in the same device leads to extremely promising performance and the prospect of MD simulations well into us/day range with a commodity cloud.

The rest of dissertation is organized as follows. We start with context and related work in Chapter 2. After that, the next several chapters follow roughly the outline of the four levels of work. Chapter 3 describes how we build an FPGA-accelerated hardware platform, including the network topology and the low-level link protocol. Chapter 4 discusses the proposed flow control mechanism and router microarchitectures, including our proposed VC-based wormhole router and VOQ collective acceleration router. Next, Chapter 5 describes routing algorithms and the framework that produces application-aware router configurations. There follow Chapters 6 and 7 about our application case study on FPGA cluster and cloud. The final Chapter 8 summarizes the entire dissertation and discusses possible future work.

Chapter 2

Background and Context

In this chapter, we provide background and context for the work of this dissertation. The overall areas of interprocessor communication and networks, HPC, and reconfigurable computing (RC), are all major branches of computer science and engineering and are covered in general by standard references. Here we begin by stating some open problems in HPC and how they are addressed by RC in general and current generation RC devices (FPGAs) in specific. We then examine the major previous work related to the four major contribution areas of the dissertation: connections, routing, interfaces, and applications.

2.1 Background

HPC remains a critical aspect of nearly all branches of science and engineering. However, three major factors are limiting HPC performance: computational efficiency, power density, and communication latency. All of these are addressed by increasing heterogeneity in computing systems, but the last one in particular by integrating communication into accelerators. This integration enables direct and programmable communication (DPC) among compute components. By *direct* we mean both direct accelerator-accelerator communication and direct communication between application and network interface. By *programmable* we mean that the communication design is flexible, in particular, that it can be optimized for both hardware and software, concerning the application.

Current high-end FPGAs are all equipped with some number (generally dozens or more) of high speed interfaces called Multigigabit Transceivers (MGT) (Altera, 2014b; Altera, 2015; Xilinx, 2009). In many production application the MGTs are the communication interfaces used by the network router/switch. In FPGA-centric clusters, MGTs are used to provide physical links among FPGAs that are high-bandwidth and low-latency. Besides the MGTs, further DPC support is provided by the FPGA's configurable computational capability including the ability to configure applications to transfer data directly to/from the MGTs. This tight coupling of computing and communication provides great potential for solving communication bottleneck issues using FPGAs. At the same time, FPGA is reprogrammable, which means the protocol, routing policy, and switch design of FPGA clusters can all easily be modified. We can therefore select the options with respect to applications such that the best performance (or the least cost) is obtained as specified by user requirements. Figure 2·1 shows a FPGA cluster equipped with MGT links.

Much of the work in this dissertation was developed by using, and to provide infrastructure for, the Novo-G# cluster (George et al., 2016). The background and system architecture of Novo-G# is described in section 3.2 and 3.3.

One of the end-goals of this thesis is to make the inter-FPGA communication of the Novo-G# more efficient and easier to use.

2.2 Previous Work Related to Contributions

High performance communication is a very broad topic. It is a subset of the general area of Networks characterized by the limitation of the communication scope to a well-defined cluster of nodes. These HPC clusters are usually assumed to have much reliable and secure communication than general networks allowing many more optimizations to be applied. Even so, HPC communication is a vast area of research. In

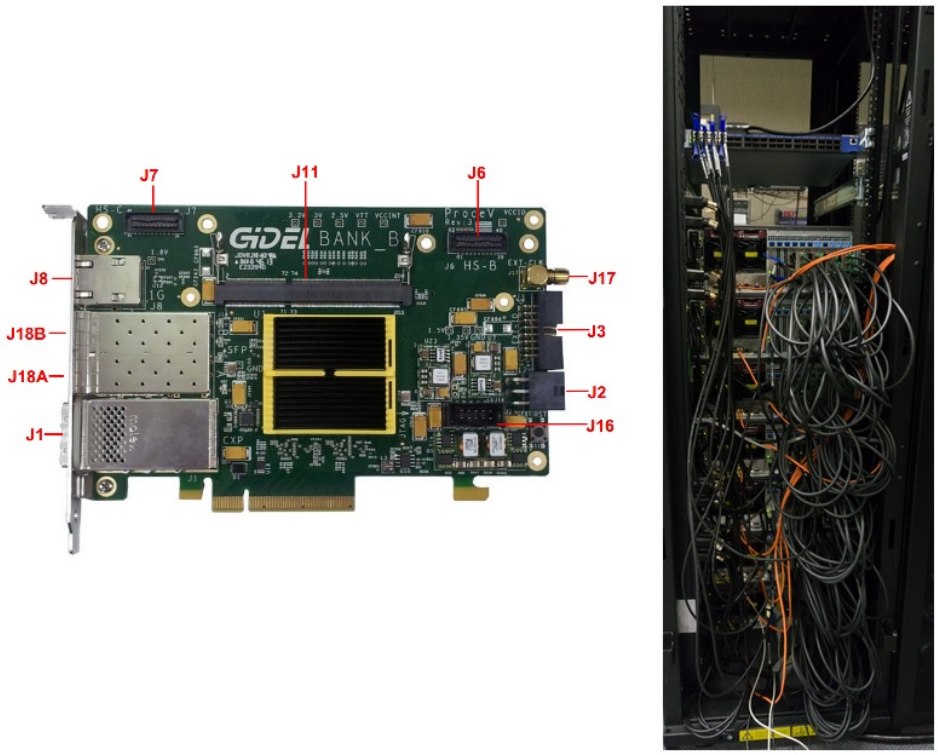


Figure 2·1: Left side: a Gidel ProceV D8 board with MGT ports (J18), right side: a FPGA cluster with 32 ProceV D8 boards

this section, we sample the work that is directly related to this dissertation. Each of the four subsections corresponds roughly to one of contribution areas mentioned in Chapter 1.

2.2.1 Previous FPGA Clusters and Interconnection Networks

Early FPGA Clusters and Interconnection Networks

Researchers have been using multiple FPGAs to implement applications for more than twenty years. In the 1990s, many researchers used multiple FPGAs to do logic emulation. This was largely due to limited resources on a single FPGA (Babb et al., 1993; Hauck, 1995; Hauck et al., 1998; Shaw and Milne, 1992; Lan, 1995; Khalid, 1999). At that time, the MGT had not yet emerged. FPGAs were interconnected via two kinds of approaches, either directly hard-wired through pins (Babb et al., 1993;

Shaw and Milne, 1992; Hauck, 1995; Hauck et al., 1998), or by using specialized Field Programmable Interconnect Chips (FPIC)(Lan, 1995; Khalid, 1999). These approaches were limited by the number and speed of the pins on FPGAs at that time. Researchers came up with different approaches to use bandwidth on pins efficiently. In (Babb et al., 1993), Babb et al. developed the virtual wire to multiplex pin resources. Hauck et al. explored methods to optimize the pin and internal FPGA logic usage based on mesh topology (Hauck, 1995; Hauck et al., 1998). FPIC was another approach to overcome pin limitations. In (Lan, 1995) and (Khalid, 1999), authors investigated methods to optimize the routing inside the FPIC by programming the FPIC into a crossbar structure.

BEE

The Berkeley Emulation Engine (BEE) system grew out of the famous RAMP project (Wawrzynek et al., 2007) and followed a similar approach. It interconnected 20 FPGAs through a mixture of FPIC and direct wiring (Chang et al., 2003). The BEE's main purpose was originally also logic emulation, specifically prototyping wireless communication algorithms. The BEE's successor design, BEE2, replaced the direct inter-FPGA wirings with MGT links; this was one of the first multi-FPGA design with MGT links (Chang et al., 2005). Together with better Digital Signal Processing (DSP) function support on FPGAs, BEE2 achieved great performance on DSP applications. It also achieved remarkable efficiency for multiprocessor system simulation in the RAMP projects (Wawrzynek et al., 2007; Krasnov et al., 2007).

Projects related to BEE include (Abdellah-Medjadji et al., 2008), which designed a multi-FPGA platform to emulate SoCs. The main limitation of these projects, including BEE, is that they did not further optimize inter-FPGA communication, perhaps because their applications did not have a communication bottleneck. Researchers from UCSD used a more recent version of the BEE system, BEE3 (Davis

et al., 2009), to measure network performance of a cluster with 8 BEE3 enclosures (Bunker and Swanson, 2013). Their primary contribution is measuring latency and bandwidth of unicast communication for different topologies.

TMD

The Toronto Molecular Dynamics machine (TMD) was a multi-FPGA system with MGT interconnections (Patel et al., 2006) developed at the University of Toronto. A major application was Molecular Dynamic simulations. Arun et al. instantiated soft processors on each FPGA and implemented a specialized MPI library called TMD-MPI to handle inter-FPGA communication (Peng et al., 2014). While soft cores have the potential for generality, they can cause high latency on critical paths.

Maxwell

Maxwell (Baxter et al., 2007) was a multi-FPGA system similar to TMD and developed at the University of Edinburgh. It contained 64 FPGA nodes organized in a 2D torus network with MGT interconnections. Similar to TMD, each FPGA had a PowerPC soft core and an MPI library to facilitate inter-FPGA communication. The main limitation again was that use of soft cores increases the latency on critical paths. Also, collective operations on Maxwell needed to go through a CPU-based Ethernet network. Another issue was that the MGT 2D-torus network could only handle basic nearest neighbor one-to-one communication.

RCC

The reconfigurable computing cluster (RCC) project is a multi-FPGA cluster developed at UNC Charlotte by the Sass Group (Sass et al., 2007). All the FPGAs are directly connected with MGT links. Besides that, they developed a configurable network abstraction layer called AIREN to provide friendly and efficient on-chip and

off-chip network interfaces (Schmidt et al., 2012; Schmidt et al., 2009). Sass, et al. also designed specialized hardware cores to offload MPI collective operation from software (Gao et al., 2009; Gao et al., 2010).

COPACOBANA and CUBE

The Cost-Optimized Parallel Code Breaker (COPACOBANA) is a high-performance, low-cost cluster consisting of 120 FPGAs (Guneysu et al., 2008) that is optimized for code breaking tasks. For interconnection among FPGAs the designers went back to a low-bandwidth 64-bit data bus. This design choice is appropriate because code breaking has little need for communication among parallel computing components.

The CUBE is a 512-FPGA system developed by Imperial College London and the Chinese University of Hong Kong (Mencer et al., 2009). They used 8 PCB boards, each with 64 FPGAs. In one PCB, inter-FPGA communication is also done by a 64-bit data bus, and all the 64 FPGAs are connected into a systolic array. Similar to COPACOBANA, the main use of this design is key search.

There are a few drawbacks in CUBE and COPACOBANA designs. First, the data bus interconnection is much less flexible than MGT interconnection and its low bandwidth cannot is not likely to satisfy communication requirements other than those similar to code cracking. Also, in CUBE, the method of integration of 64 FPGAs into a single PCB board makes it very vulnerable to system failure because a failure of one FPGA chip could break the entire board.

Axel

Researchers at Imperial College also did much work on mapping applications on heterogeneous clusters with a mixture of CPUs, FPGAs, and GPUs (Tsoi and Luk, 2010; Anson et al., 2010). Their main focus was partitioning and mapping applications on heterogeneous computing components to facilitate load balancing. Researchers of

Axel paid little effort into optimizing communication.

Zedwulf

Zedwulf is a 32-node multi-FPGA SoC. The inter-FPGA communication is implemented by connecting all the 32 FPGAs via an Ethernet switch (Moorthy and Kapre, 2015). Although it is a good practice at multi-FPGA SoC implementation, the Ethernet-based interconnection introduces much higher latency overhead than MGT interconnection.

Catapult

The most significant FPGA application project of (at least) the last decade is Microsoft's Catapult where they have integrated FPGAs into their data center nodes (Putnam et al., 2014; Caulfield et al., 2016). In the first version (Putnam et al., 2014), they connected 48 FPGAs in a rack into a 6×8 2D torus using MGT links. The MGT links are mainly used to relay the pipelined data flow for their Bing search engine. In the second version (Caulfield et al., 2016), they have removed the MGT secondary networks because of the difficulty of scaling and maintaining cabling to datacenter scale. The emphasis is more on the scalability of FPGAs in data centers, while we concentrate on HPC clusters of directly connect FPGAs (rather than through network routers), programmable interconnects, and application-aware communication.

Bluehive and other application-specific FPGA clusters

There have been many projects that have used multiple FPGAs to accelerate single applications, including neural network Simulation (Bluehive) (Moore et al., 2012), lattice Boltzmann computation (Kono et al., 2012), content-based image retrieval (Liang et al., 2013), and stencil computation (Sano et al., 2014). All tuned inter-FPGA communication for their particular applications. Our emphasis is on general

communication and methods of optimizing for many different applications.

Other Work on Inter-FPGA Communication

There is also previous works that investigated inter-FPGA communication without necessarily building a cluster. In (Markettos et al., 2014), the authors developed a new MGT link protocol called BlueLink, which demonstrated better area consumption than a commercial protocol. However, in this work they did not show the latency comparison against other commercial protocols besides Ethernet, which has high latency because of its complexity. In (Jun et al., 2015), the authors used a standardized way to the implement network and transport layers of inter-FPGA communication. However, the packet-switching they adopted introduced high latency per hop because of long headers. Also, they did not investigate reducing networking congestion. (Giordano and Aloisio, 2011; Giordano and Aloisio, 2012; Liu et al., 2014) investigated techniques to achieve near fixed-latency inter-FPGA high-speed serial links. It requires customized link protocol IP, however, which is beyond the scope of this dissertation.

2.2.2 Previous Work on FPGA System/User Interfaces

Since the invention of FPGAs, users have been concerned about programmability. Within the massive literature, the most closely related is that which addresses FPGA middleware to simplify the programming model of FPGAs while keeping the efficiency as high as possible.

Some of this work has targeted encapsulating FPGA kernels into OS processes, such as BORPH (So and Brodersen, 2008). SPREAD (Wang et al., 2013) and ReconOS (Agne et al., 2014) targeted threads so that application development on an FPGA-based system could rapidly cross the SW/HW boundaries. Other work provides a unified hierarchy for on-chip and off-chip memory of FPGA (Chung et al.,

2011; Weisz and Hoe, 2015). Yet other work provides better programmability, scalability, and customization for MPSoCs built by multiple soft cores on FPGA (Yiannacouras et al., 2006; Mahr et al., 2008; Unnikrishnan et al., 2009; Kritikos et al., 2012; Skalicky et al., 2015).

There are only a studies that have provided middleware for inter-FPGA communication. TMD-MPI (Saldana and Chow, 2006) is a software-based communication library that facilitates inter-FPGA communication for TMD system. They released an improved version in 2014 (Peng et al., 2014) by offloading MPI Bcast and Reduce into FPGA logic. Andrew, et al. made a similar effort in (Schmidt et al., 2012). There are two issues in these studies. First, their systems are based on soft processors. The packetizing and depacketizing on a soft processor causes huge overhead. Second, they both offload MPI collective directives onto FPGA logic. However, neither of them exposed the ability to modify collective routes to application users, which means application knowledge cannot be used to reduce network congestion in their systems.

LEAP took another approach to providing middleware for inter-FPGA communication (Fleming et al., 2012; Fleming et al., 2014). They blurred the difference between inter-FPGA and intra-FPGA communication using a concept called the Latency Insensitive (LI) Channel, which makes a multi-FPGA system look like a bigger single FPGA. However, because of the LI channel, communication latency is not their concern.

2.2.3 Previous Work of Router Architectures

The router is an essential component in almost all of the interconnection networks including HPC clusters, datacenters, and NoCs. A vast literature discusses router designs on all of these platforms. We give a high-level overview. (Feng, 1981) provides a comprehensive survey for early router designs. (Karol et al., 1987; Hluchyj and Karol, 1988) did a thorough study of the performance of simple input queueing and output

queueing routers. Almost at the same time, (Tamir and Frazier, 1988) proposed the Virtual Channel Queue (VOQ) router. (Dally, 1992; Dally and Aoki, 1993) then proposed what has become the canonical wormhole VC-based router design and the flow control related to it. Later Dally's VC-based router design became a standard for almost everyone. Researchers have since proposed additional techniques such as speculation (Peh and Dally, 2001; Kim et al., 2006), look-ahead (Mullins et al., 2004; Mullins et al., 2006), express VC (Kumar et al., 2007), and prediction (Matsutani et al., 2009) to shorten the latency and optimize the throughput of the conventional VC-based router. Almost all the state-of-the-art interconnection of big systems such as Intel Ominipath (Birrittella et al., 2015), IBM BlueGene (Chen et al., 2011) use VC-based routers.

Since we have designed two router architectures, the related work of router architectures are correspondingly divided into two subsections. We have already walked through the milestones in the VC-based router designs, so in the first subsection, we mostly concentrate on the previous VC-based routers implemented on FPGAs. Compared with general VC-based routers, the amount of literature for collective acceleration routers is vastly reduced. Therefore, in the second subsection, we do a rather comprehensive survey of the existing collective acceleration routers.

Previous Work of Wormhole VC-based Routers on FPGAs

The FPGA community has also given a lot of effort on porting the VC-based router design onto FPGAs. (Brebner and Levi, 2003; Marescaux et al., 2004; Kapre et al., 2006) discussed the design choices of implementing a packet switching network on FPGAs. They did not go into in-depth router design. CONNECT (Papamichael and Hoe, 2012) is the first work that studied the differences in router implementation between FPGAs and ASICs. CONNECT supported three kinds of router architectures: simple input buffering, VC-based, and VOQ. The Authors in CONNECT

believed that the conventional pipelined router design is not appropriate for FPGA NoCs because of longer wiring delay. The design philosophy of CONNECT rather is to achieve minimal resource utilization while maintaining a moderate operating frequency. (Huan and DeHon, 2012) proposed a design called split-merge NoC pipelined, which is very different from the conventional design. It solves the long wiring delay issue in (Papamichael and Hoe, 2012) but with higher area overhead. Heracles (Kinsy et al., 2013) build a framework generate an entire NoC on FPGAs. Hoplite (Kapre and Gray, 2015; Kapre and Gray, 2017) proposed a deflection router based on the conventional router. Its main design goal is extremely high efficiency to facilitate massive replication on a single chip.

To our best of knowledge, we are the first to implement wormhole VC-based routing on a network based on FPGA MGT links. All the previous multi-FPGA systems either do not have routers or do not support VC-based routers. In the most recent BEE-series work (Bunker and Swanson, 2013), its router design has no VC support. TMD adopted fully connective topology to avoid routers (Patel et al., 2006), which also limits the scaling of the network. Maxwell has no router support, which results in that its network only supporting nearest neighbor patterns (Baxter et al., 2007). The router on RCC is a simple crossbar switch with some buffering (Schmidt et al., 2012), which has no VC support. The inter-node communication in COPACOBANA is done on a 64-bit bus, which has no need for a router (Guneysu et al., 2008). In CUBE, the FPGAs are organized in a systolic array-style (Mencer et al., 2009). Each FPGA can only talk to nearest FPGAs. In Axel, the inter-FPGA communication is mainly through the front-end Ethernet network (Tsoi and Luk, 2010). The back-end network with MGT links has no router support. In Zedwulf, FPGAs communicate purely through commodity Ethernet (Moorthy and Kapre, 2015). Few details are publicly available about the Catapult I router (Putnam et al., 2014). However, its

flow control is virtual cut-through rather than wormhole. In Catapult II, the FPGAs are bump-in-the-wire in a commodity network (Caulfield et al., 2016) with the routing done by a commodity switch. In Bluehive, the router can only perform simple DOR routing (Moore et al., 2012). There is no specific VC support. Other multi-FPGA systems (Kono et al., 2012; Sano et al., 2014; Liang et al., 2013) work for only one application; their inter-FPGA communication only supports systolic-array-style data transfer. None supports general routing.

Later we describe versions of our router microarchitectures based on rings and crossbars. Other researchers have looked at different router microarchitectures (Kim et al., 2007). Intel (Intel, 2013) and IBM (Pham et al., 2006) also have used ring topologies because compared with crossbars, they has linear area complexity and easier flow control.

Previous Work of Collectives Routers

Several groups have studied, designed, and fabricated ASICs to support multicast and reduction on NoCs (Jerger et al., 2008; Rodrigo et al., 2008; Samman et al., 2008; Abad et al., 2009; Wang et al., 2009; Krishna et al., 2011; Krishna and Peh, 2014). However, all of them implemented their own dynamic adaptive routing logic on their chips. None of them considers static-scheduled routing. Compared with static-scheduled routing, an adaptive routing algorithm is unable to use the global traffic information to help them make routing decisions.

Among these studies, (Krishna and Peh, 2014) has the best performance, achieving single-cycle multicast. It requires special hardware support, which is impractical on FPGA clusters. Among the other work, (Wang et al., 2009) and (Krishna et al., 2011) have the best performance. The differences between these two are that (Wang et al., 2009) only supports multicast, while (Krishna et al., 2011) supports both multicast and reduction. In dynamic adaptive routing, different logic is needed to support both

multicast and reduction. In table-based routing, both multicast and reduction can share the same set of routing tables; this is another benefit of statically-scheduled routing. So we select (Wang et al., 2009) together with their algorithm (RPM) as our baseline.

2.2.4 Previous Work of Routing Algorithms

Routing algorithms is another major area of computer research. There have been multiple books (Dally and Towles, 2004; Duato et al., 2003; Leighton, 2014) and thousands of papers published on this topic. (Bjerregaard and Mahadevan, 2006; Agarwal et al., 2009) conduct two comprehensive surveys on the research topics about state-of-the-art routing algorithms.

Routing algorithms can be categorized into two classes, unicast and collective. Most literature emphasizes unicast routing algorithms. We sample five representative routing algorithms and implement them on our proposed VC-based router. The five routing algorithms are Dimensional Order Routing (DOR) (Sullivan and Bashkow, 1977), Randomized Oblivious Multi-phase Minimal Routing (ROMM)(Nesson and Johnsson, 1995), Orthogonal One-turn Routing (O1TURN)(Seo et al., 2005), Randomized Load Balance Routing (RLB)(Singh et al., 2002), Credit Count Adaptive Routing (CCAR)(Kim et al., 2005a). The details are given in Chapter 5.

Compared with unicast routing algorithms, there are fewer collective routing algorithms. They can also be divided into two parts: previous static-scheduled routing algorithms and previous dynamic collective routing algorithms. We organize them into two sections.

Previous Work of Statically-Scheduled Routing Algorithms

If communication patterns are static and known *a priori*, as is often the case, then judicious routing can reduce congestion, latency, and the hardware required. This

routing method is often referred to as offline or statically scheduled routing. The concept of offline routing is not new. As early as the 1980s and 1990s, there were studies where routing decisions were preloaded into context memory to schedule the routes in their systolic arrays (Kung, 1988; Borkar et al., 1990). After that when 2D-mesh topology became popular, the author in (Shoemaker et al., 1996) applied offline routing into the 2D-mesh routing as well.

(Kinsy et al., 2009) is the first work that proposed an offline routing solution on NoC, which mainly targeted at workloads with one-to-one communication. There are a few previous studies that have compared offline static routing with online dynamic routing on FPGAs (Laffely et al., 2001; Kapre et al., 2006; Kapre, 2016). There are several limitations in these studies. First, they all mentioned context memory for pre-computed routing decision causes a lot of overhead. However, none of them uses efficient methods to compress it. Second, all of them explored offline routing for one-to-one communication. None of them discussed offline routing for collective operations. Third, communication in their papers remained on either SoC or NoC. None of them studied multi-chip communication.

Previous Work of Dynamic Collective Routing Algorithms

Researchers have not been active on this topic for a while. The most popular time for this topic was between 2000 to 2014, when the NoC emerged. There are several representative studies: VCTM (Jerger et al., 2008), MRR (Abad et al., 2009), bLBDR (Rodrigo et al., 2008), RPM (Wang et al., 2009), and whirl (Krishna et al., 2011). RPM and whirl are the two algorithms with the best overall performance. Compared with whirl, RPM saves more chip resources. RPM has been updated. (Wang et al., 2011) modified it in order fit it into irregular networks. (Ebrahimi et al., 2014) extended it from 2D-mesh NoCs to 3D stacked NoCs.

2.2.5 Previous Work on Applications on FPGA Clusters

We are interested in examining the strong-scaling of applications because this provides the most extreme test of HPC communication. Any application with significant communication reaches a strong scaling limit eventually as the problem size remains fixed as it is distributed across an ever increasing number of nodes. Most previous work, however, has focused either on non-communicated applications, or examined weak scaling.

3D-FFT and Molecular Dynamics Simulations (MD) are the two applications we mainly focused on. 3D-FFT is especially difficult to scale strongly because it requires all-to-all communication no matter how it is partitioned and mapped. MD is well-known as a computation-intensive application. However, it could also easily become communication-bound because of two reasons. First, it has a 3D-FFT on its critical path. Second, to compute short-range force, each node needs to distribute positions of molecules to neighbor nodes and collect forces from neighbor nodes, which will cause a substantial number of multicast and reduction operations in strong-scaling scenarios.

Since we mainly study 3D FFT and MD in this dissertation, we examine the related work about the implementation of them on FPGA in the next two subsections.

Previous Work of 3D FFTs on FPGAs

We now review previous work implementing 3D FFTs on FPGAs. In (Varma et al., 2013), the authors focus on accelerating the 3D FFT by redesigning 1D FFT IP using Hard Embedded Blocks (HEB). However, their 1D FFT IP is still slower than the Altera's IP. The authors in (Nidhi et al., 2013) reduce the data transfer time of XY corner turn and YZ corner turn by using a runtime configuration method called Coarse Grain Reconfigurable Architecture (CGRA). However, their approach does not

scale well. When the problem size increases, the running time increases exponentially. In (Humphries et al., 2014), we provide an effective 3D FFT implementation on a single FPGA. Preliminary work by one of the authors (Humphries, 2013) investigates some switching issues in more depth but does not account for congestion. We have extended that in this dissertation.

Previous Work on MD on FPGAs

(Wolinski et al., 2003) first did a preliminary attempt of implementing MD on FPGAs. In this work, only the motion update is implemented. (Scrofano and Prasanna, 2006) investigated the hardware/software co-design of MD on FPGAs. (Patel et al., 2006) designed a Message Passing Interface (MPI) function library to facilitate the communication of MD on FPGAs. (Phillips et al., 2007) proposed a novel architecture called FLEX to support dynamic load balancing on an FPGA for MD.

BU CAAD lab has been focusing on advancing the state-of-the-art of MD on FPGAs consistently for more than a decade. (Gu et al., 2006a) implemented MD on a Xilinx Virtex-II board, which demonstrated a speedup up to 88×. (Gu et al., 2006b; Gu et al., 2006c) integrated FPGA accelerated non-bonded forces calculation into ProtoMol MD code(Matthey et al., 2004). In (Gu and Herbordt, 2007), multigrid computation, which is an essential part of long-range force calculation, was implemented on FPGAs with a speed of 5× to 7×. In (Gu et al., 2008), the force pipeline was extensively explored. (Chiu et al., 2008; Chiu and Herbordt, 2009) proposed new methods of filtering and mapping particles onto different pipelines. (Chiu and Herbordt, 2010; Chiu et al., 2011) systematically examined the design space of the short-range force pipelines on FPGAs. (Khan and Herbordt, 2011) presented an event-based decomposition to scale discrete molecular dynamics simulation. (Khan and Herbordt, 2012) studied the communication requirements for MD on FPGA clusters. (Herbordt, 2013) researched the approaches of the architecture/algorithm

codesign of MD processors. (Herbordt et al., 2007; Herbordt et al., 2008; Khan et al., 2013; Vancourt and Herbordt, 2009) provide general overviews of these methods.

Chapter 3

Target System Architecture and Implementation

In this chapter, we first discuss the design choices in building multi-FPGA systems. We then introduce the background and give an overview of our multi-FPGA target system. There follows our system architecture and design details. After that, we describe our investigation on the impact of commercial IP on the jitter of inter-FPGA communication. Finally, we discuss the impact of these results, especially on routing statically scheduled communication.

3.1 Design Choices of Multi-FPGA Systems

In this section, we list the most prominent design choices of existing multi-FPGA systems; these are described in Chapter 2.

1. **Direct versus indirect Network** A multi-FPGA network *direct* when the router resides on the same chip as the processing unit. Otherwise, it is a indirect network. Indirect networks send packets to commodity routers which perform the packet switching. Besides Catapult and Zedwulf, all the multi-FPGA systems listed in Chapter 2, including Novo-G, are direct networks. Indirect networks save the effort of designing routers, but also introduce significant latency overhead.

2. **Different Topologies** The choice of topologies is flexible for multi-FPGA systems because current FPGAs have many MGT ports. It is more desirable to have a high radix topology because of shorter diameter, but the drawback is additional board and cabling complexity. (Bunker and Swanson, 2013) did an excellent study on the impact of different topologies on performance. Novo-G adopts the 3D-torus because it has high radix, matches well with physical applications, and leads to reasonably priced FPGA boards.

3. **Cloud versus cluster** For multi-FPGA systems, cloud and cluster setups have their own advantages. An illustration showing the differences is displayed in Figure 3·1. In conventional HPC clusters, as shown in Figure 3·1(a), accelerators communicate through their hosts, which introduces a long latency penalty. FPGA clouds such as Catapult II (Caulfield et al., 2016) address this issue by placing the FPGAs on the NIC as a bump-in-the-wire, which sits between the commodity network and hosts. More details about FPGA clouds are in Chapter 7. FPGA clusters such as Novo-G take this a step further by interconnecting FPGAs directly through MGT links; this is the approach for most multi-FPGA systems mentioned in Chapter 2. While this leads to good cluster designs, the additional cabling may not be viable in cloud datacenters.

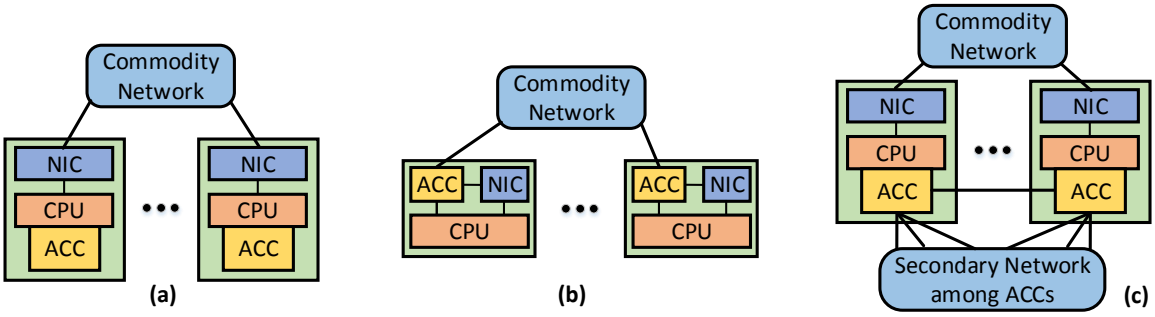


Figure 3·1: Three models for FPGA-based HPC systems. a) Standard HPC cluster. b) FPGA clouds. c) FPGA clusters.

3.2 Background of Novo-G

Novo-G (George et al., 2011) began in 2009 as an effort to create a research cluster using high-density FPGA boards to accelerate scientific applications. The original machine began with a head node and 24 Linux servers, each featuring a quad-FPGA board from Gidel (Gidel, 2016), for a total of 96 Altera Stratix III E260 FPGAs. Over subsequent years, the machine has been upgraded annually and now stands at 192 Stratix III E260 FPGAs in 24 servers, 192 Stratix IV E530s in 12 servers, 64 Stratix V GSMD8s in 16 servers, and the second set of 64 Stratix V GSMD8s in 16 servers under construction. Each server features dual Intel Xeon multicore processors. Server connectivity is provided by Gigabit Ethernet and DDR/QDR InfiniBand within the system, and a 10 Gb/s connection to the Florida LambdaRail.

At the NSF Center for High-Performance Reconfigurable Computing (CHREC), Novo-G has been used for a variety of application acceleration projects from the domains of bioinformatics (Lam et al., 2013), image processing (Craciun et al., 2013), and financial computing (Sridharan et al., 2012). The common factor among the above applications is that they are embarrassingly parallel and can, therefore, scale almost linearly with the available hardware resources. A greater challenge is accelerating communication-intensive applications like Molecular Dynamics. Traditionally such communication makes use of centralized networks such as Ethernet or InfiniBand and entails multiple interactions between the FPGA and the host. The increased latency and communication bottleneck in such applications emphasizes the need for a better solution.

The Novo-G# system (George et al., 2016), which has been under development for the last two years, features high-density Stratix V FPGAs connected directly into a 3D torus network, enabling low latency and high-bandwidth communication among the FPGAs. The hardware is supported by an easy-to-use, but efficient, protocol

stack that packetizes, routes, and buffers data, allowing communication-intensive, multi-FPGA applications to be developed rapidly. CHREC has also recently added support for OpenCL-based, multi-FPGA applications that can also utilize the inter-FPGA communication links.

3.3 Architecture of Novo-G#

The Novo-G# system is part of our effort to create an FPGA cluster that can handle communication-intensive applications. In keeping with that theme, the system features ProceV boards, which are PCIe-based accelerator boards from Gidel populated with Stratix V GSMD8 FPGAs from Altera. The GS-series devices are optimized for high performance, high bandwidth applications with support for up to 36 on-chip transceivers that can operate up to 12.5 Gbaud. Each FPGA is connected to two 8GB DDR3 SODIMM and two 36-Mbit SRAM memory banks and communicates with the host CPU via PCIe v3. Four FPGAs are housed in a 4U chassis with two Xeon E5-2620V2 (Ivy Bridge) processors per server. The servers themselves are interconnected via Gigabit Ethernet and QDR InfiniBand.

The boards from Gidel have a custom daughter board that allows external access to 24 high-speed transceivers. The transceivers are grouped into six bidirectional links, each link consisting of four parallel channels, enabling the construction of a 3D torus of arbitrary size. Physical connectivity between the boards is provided by a COTS CXP-3QSFP+ split cable that allows each FPGA to be connected in six different directions. Initial deployment of the Novo-G# system was completed in the second half of 2014 with 32 Stratix V boards housed in eight chassis and supporting up to a 2^3 torus, and an upgrade to 64 nodes ($4 \times 4 \times 4$ torus) was completed in August 2015. Figure 3·2 shows how a 2^3 torus is formed in Novo-G#.

There are several obvious reasons why we chose the 3D-torus as our topology. First

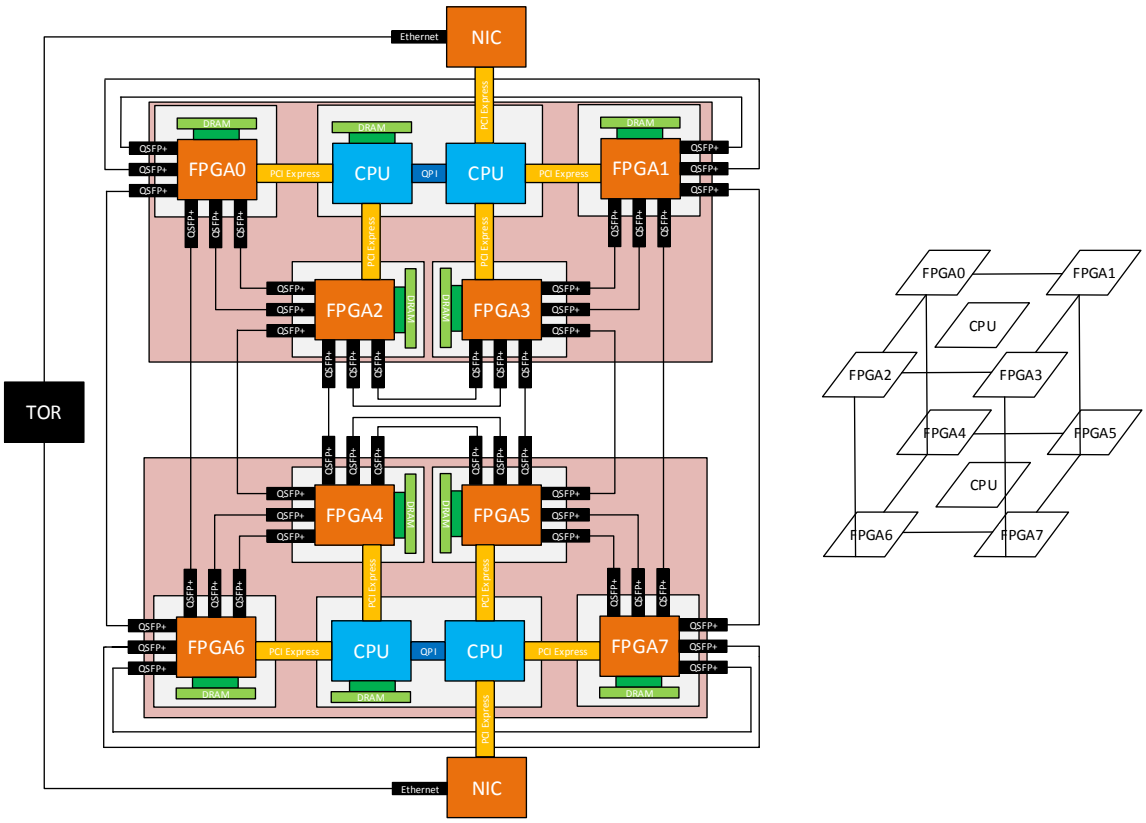


Figure 3·2: An example showing a 2^3 torus Novo-G# system

is that many scientific application operate in 3D space — 3D image processing, N-body, and MD simulation — and so map naturally onto the 3D torus. Second is that the 3D torus has a comparatively short network diameter, high bisection bandwidth, and high connectivity.

3.4 Investigation of MGT Link Connections

3.4.1 Introduction of MGT link Protocol

The Altera Statix V has a large number of protocol-specific transceiver PHY IPs (Altera, 2015). The protocol-specific transceiver PHYs configure the PMA and PCS to implement a particular protocol. Among them, we pick Interlaken PHY (StremDSP, 2015) and SerialLite III (Altera, 2013). We select them because they are light-weight

and have both low-latency and high-throughput, which are the main needs of HPC applications.

Interlaken PHY and SerailLite III are both based on the Interlaken protocol. Interlaken is an open interconnect protocol that was developed by Cisco Systems and Corina Systems in 2006. Its full version targets chip-to-chip packet transmission under high-bandwidth requirement (Cortina and Cisco, 2008). However, its full version is not suitable for low-latency communication. The first PHY IP we selected is the Interlaken PHY IP core, which supports a lightweight version of Interlaken PHY protocol. The Interlaken PHY IP core is a scalable, high-bandwidth and low-overhead FPGA interconnection supported by both Altera and Xilinx FPGA device families with an identical user interface.

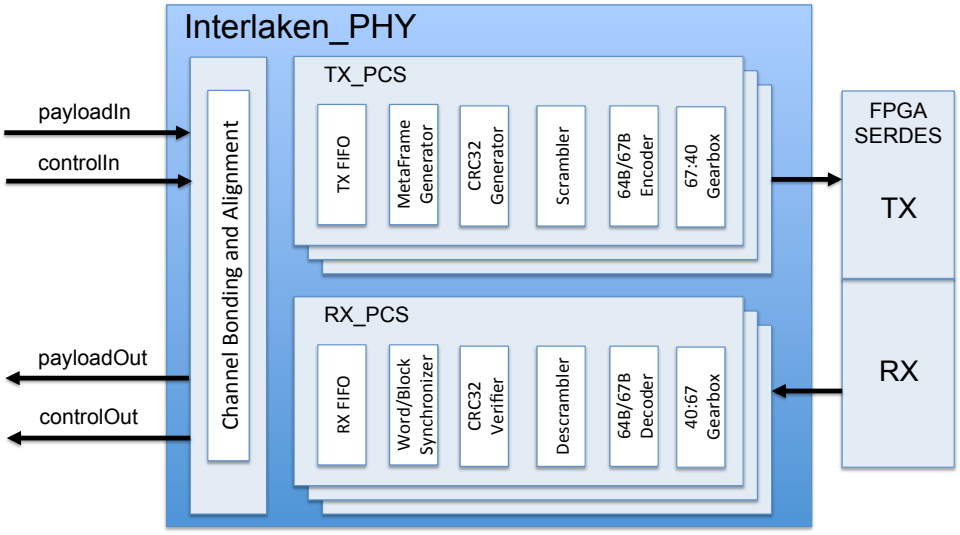


Figure 3·3: The structure of the Interlaken PHY IP core

The structure of the Interlaken PHY is displayed in the Figure 3·3. The transceiver has a SerDes part, which is called Physical Medium Attachment(PMA). It links to the serial links. The data rate on this link can go as high as 14.1 Gbps per lane. Another part is called the physical Coding sublayer(PCS), which processes the data in front of the SerDes logic. This part is the kernel of the transceiver. The PCS includes

the TX/RX FIFO, MetaFrame generator, CRC32 generator, scrambler, 64B/67B Encoder, and Gearbox. The gearbox in the PCS adjusts the PMA data width to a wider PCS data width when the PCS is not two or four times the PMA width. For a serial interface to distinguish work boundaries, an encoding method is required. In Interlaken PHY, 64B/67B encoding is developed based on the IEEE 802.3 64B/66B encoding (Cortina and Cisco, 2008). The scrambler aims to eliminate the error multiplication and guarantee bit transition density. On each lane, the CRC32 is provided as a diagnostic tool on each lane to trace errors. The data in Interlaken PHY are transferred in a frame-by-frame manner, which is called MetaFrame. One MetaFrame is formed by MetaFrame generator including one synchronization word, one scrambler word, one diagnostic word, one or more skip words, and the data payload. And the TX/RX FIFOs are going to buffer the data traffic. On the other side, the Interlaken PHY is connected to the user logic through the Avalon-ST interface.

The second PHY IP we selected is the SerialLite III Megacore. It is based on a hardened Interlaken PHY IP core, and it provides the synchronizer among multiple Interlaken PHY date lanes. This provides excellent scalability in the number of lanes (Altera, 2013). However, compared with Interlaken, its synchronizer introduces an extra delay on the data path and requires additional hardware resources. Table 3.1 gives a comparison of characteristics between Interlaken and SerialLite III. The latency comes from our measurements when running them on the board. The resource utilization comes from the report generated from Quartus II.

3.4.2 Problem of Link Latency Variances and Clock Jitters

The transceivers in the Novo-G# are based on the Interlaken-PHY IP core of the Altera Stratix-V FPGA family (Altera, 2015). We use the Interlaken-PHY IP core to implement the direct FPGA-FPGA communication. The Interlaken PHY IP core is a lightweight, scalable, high-bandwidth, and low-overhead FPGA IP family supporting

Table 3.1: The comparison of characteristics between Interlaken and SerialLite III

Attribute	Interlaken	SerialLite III
Data rate	10 Gbps	10 Gbps
End-to-end latency	176 ns	307 ns
Number of Lanes	4	4
ALMs	240	1080
ALMs used for memory	0	0
Combinational ALUTs	394	1380
Dedicated Logic Register	427	2525
I/O Registers	0	0
Block Memory Bits	0	0
DSP Blocks	0	0

both Altera and Xilinx FPGA devices.

When implementing applications on a single FPGA, the design is often easy to synchronize. If we want to expand it to a large-scale cluster, this issue is more problematic. There is no global clock for all the nodes in one cluster. Therefore, two identical computation kernels on two different nodes do not finish simultaneously. For communication, the latencies on different physical links differ as well: transceiver clock rates vary and cables have different lengths. The way to deal with these variances, or jitter, is through adding complexity to the communication mechanism such as increased buffer sizes and throttling through flow control. These add overhead to both resource utilization and link latency. What we investigate here is the nature of this cost and how it affects application performance.

In general, the source of latency variations in SerDes devices resides in both serial and parallel parts (Giordano and Aloisio, 2011). On the serial side, there is a frequency multiplier from the input clock of the transmitter to the output clock of the serial link. On the receiver side, a low-frequency clock is recovered from the high-frequency signal on the serial link with a frequency divider. Potential phase change

happens during the frequency multiplication and division. Also, the phase change is non-deterministic, which implies one source of the latency variation of the data transfer on the link. Also, the cable lengths and the serial clock frequencies vary from link to link. On the parallel side, the elastic FIFOs introduce another source of latency variation. This variation is because latencies in the FIFOs are determined by the distance between the read and write points assuming the reading and writing rates are the same. The positions of these pointers, however, is affected by many factors; these can be approximated as random if no dedicated mechanism is introduced.

3.4.3 Latency and Jitter Measurements

In an FPGA cluster there two primary forms of jitter: clock variation among different nodes and variations of latencies among different links. We examine these in turn and produce a simple model to be used as the basis for performing accurate simulations.

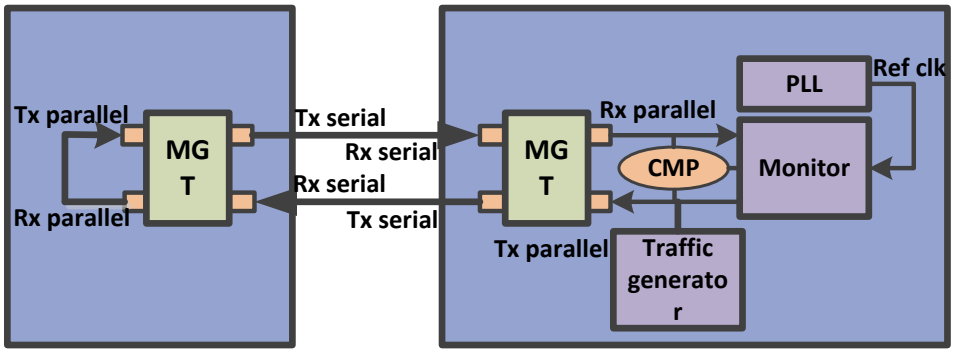


Figure 3·4: Ping-Pong test to measure latency variation.

3.4.4 Variation of Communication Latencies

The latency variations reside at four levels:

1. in one lane over time,

2. among different lanes of the same link,

3. among different links of the same FPGA chip, and

4. among different transceivers of the different FPGAs.

We performed experiments to obtain results for all four.

We use the classic ping-pong test illustrated in Figure 3·4. We validated by examining the waveforms directly. The FPGA on the right has a traffic generator which sends data to the MGT interface in a parallel format. At the same time, the monitor creates and records a random value for the outbound data stream of the transmitter. As data is transferred to the TX parallel port module, a counter in the monitor is initialized. The data is then sent through the serial link to the FPGA on the left side, after which the data is routed from that FPGA's RX port to its TX port and transmitted back again. The FPGA on the right side eventually receives the return data, and a comparator checks it with the recorded sent data. Once that happens, the counter in the monitor block stops. This mechanism is very accurate, working on the frequency from one PLL.

The setup of the experiment is shown in 3·5. We deployed two FPGAs to determine the variation among the boards. For each board, two MGTs are configured with Interlaken PHY IP to explore the difference among the MGTs on the same board. Each MGT is configured with four lanes to discover the variations among the lanes of the same link table. The physical characteristics for the Interlaken PHY IP are shown in Table 3.2.

To explore level-one latency variation (single lane) we sample its latency 75 times and plot the results (see Figure 3·6). We find that distribution appears to be random with distribution TBD. The mean and variance are shown in Table 3.3. The level-two variations are found by comparing the different lanes in the same transceiver. The level-three variations exist in the differences between the two MGTs of the same FPGA. The level-four variations can be identified by comparing the differences be-

tween FPGA0 and FPGA1. When we look at the variation over time for a single lane, the maximum latency is about 40% larger than the minimum (on average). In some extreme cases (Lane3 of the MGT1 on FPGA0), the maximum latency is close to double of the minimum latency.

When we compare the difference among multiple lanes and boards, however, the variation is relatively trivial compared to the fluctuation in the time dimension.

Table 3.2: The physical Characteristics of the Interlaken IP

Attribute	# or rate
metaframe length	2048
Data rate	10Gbps
Number of Lanes	4
ALMs	240
AlMs used for memory	0
Combinational ALUTs	394
Dedicated Logic Register	427
I/O Registers	0
Block Memory Bits	0
DSP Blocks	0

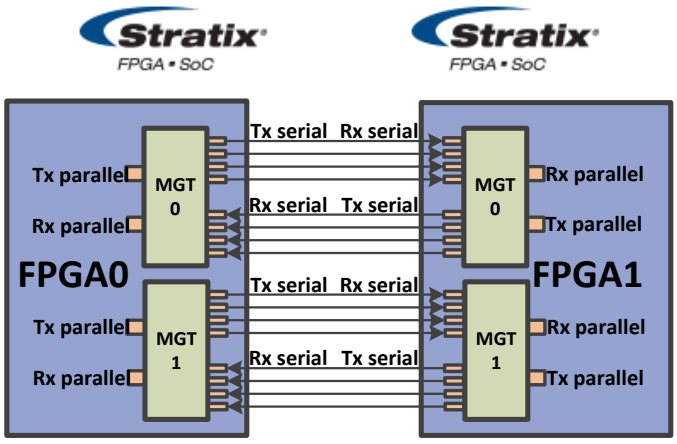


Figure 3·5: The setup for the ping-pong test

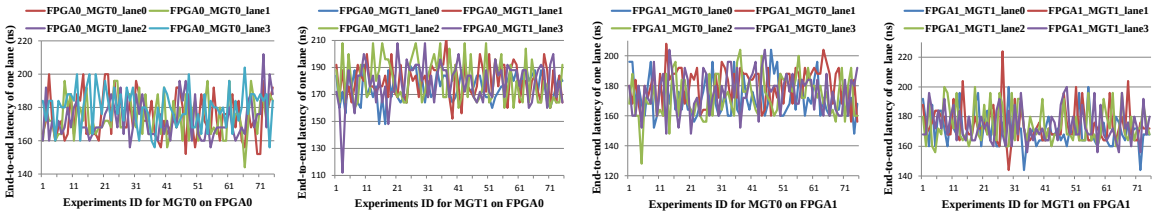


Figure 3·6: The variations of end-to-end latency over time for four lanes

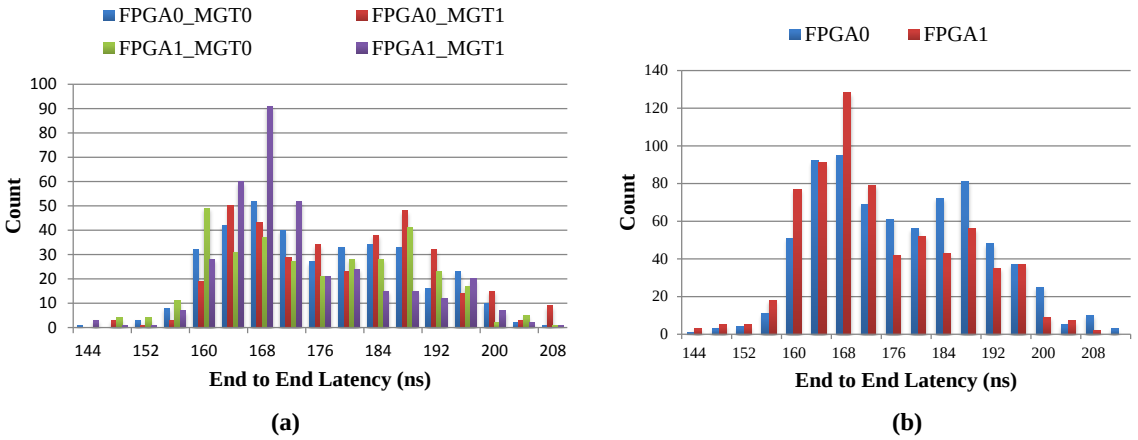


Figure 3·7: (a)The latency distribution comparison among four MGTs (b)The latency distribution comparison between two Altera Stratix V boards

Figure 3·7 gives us a different perspective with the (a) panel giving latency distributions for different MGTs and the (b) panel different FPGAs. While not a perfect fit, we use a Gaussian with a mean of 175.7ns and Standard Deviation of 12.34ns to express its distribution.

3.4.5 Measurement of Clock Jitter

The clocks on the different boards do not have perfectly matched clock frequencies. To measure the jitter, we use the output clock of a PLL at 250MHz as a reference to gauge the frequency of clk0 signals of the four ProceV boards.

To make the experiment concrete, on the four boards we instantiate a single 16-point 1D FFT IP. We run this module 1024 times on each and measure the execution

Table 3.3: The mean and variance of link latencies. All times in ns.

FPGA0	MGT0				MGT1			
Lane	0	1	2	3	0	1	2	3
Avg	175.8	175.5	172.9	180.1	175.7	175.7	179.9	179.0
Min	152	144	156	156	148	152	160	112
Max	208	204	212	204	192	212	212	212
STD	13.4	12.5	11.2	11.3	10.6	12.5	15.5	15.5
FPGA1	MGT0				MGT1			
Lane	0	1	2	3	0	1	2	3
Avg	171.8	179.8	172.5	175.4	170.6	173.9	173.9	173.5
Min	148	156	128	148	144	144	156	156
Max	204	208	204	204	200	224	200	200
STD	12.5	12.3	14.2	13.6	10.8	12.8	12.2	11.1

time with respect to the reference clock. The measured effective frequencies are shown in Table 3.4. The difference between the longest and shortest cycle is 6 ps. WHile this result appears to be trivial, it is actually significant for practical applications. For example, for a 256^3 FFT the number of clock cycles spent on the computation will be around 1000. Then the jitter between two neighboring nodes could be as large as 6 ns, which is in the same order of magnitude as the variation in link latency.

Table 3.4: The clock jitter over four ProceV boards

	FPGA0	FPGA1	FPGA2	FPGA3
freq(MHz)	103.174	103.213	103.239	103.200
cycle(ns)	9.692	9.689	9.686	9.690
jitter (ps)	6	3	0	4

3.4.6 Case Study: 3D FFT

In the previous section, by conducting several simple experiments, we obtained an estimate of link latency, the variation of link latency, and clock jitter. In this section, we apply this information to the simulation of the 3D FFT. The detailed method

and implementation of 3D FFT are be presented in chapter 6. In this subsection, we demonstrate the results and conclusions directly.

In our previous work (Sheng et al., 2014), our simulations assume fixed link latency and do not take clock jitter into account. In this paper, we replace fixed link latency with the various latencies that obey the Gaussian distribution and also add the clock jitter into the simulation. Table 3.5 demonstrates the new FFT simulation results and the comparison with the results that use fixed latency. Cluster size is 4^3, FPGA operating frequency is 150MHz, and the data type is the 32-bit floating point. The right three columns have latencies in microseconds. "Fixed" denotes average latency with no variance, "Variable" denotes that variance has been modeled, and "Ref" denotes previous results where we assumed (very conservatively) 100MHz FPGA frequency and 500ns fixed link latency (Sheng et al., 2014).

Table 3.5: The 3D FFT latency case study.

fft size	fixed	variable	reference
16^3	1.63	1.77	3.98
32^3	2.24	2.35	5.46
64^3	6.59	6.93	16.76
128^3	38.51	40.17	101.11

We note that despite the apparently substantial variance, the two new results are not that different. The reason behind is that the variation of the latency on the physical links and the clock jitter are small when compared with the latency caused by congestion. But still, the variation of physical links and clock jitter degrade the performance for all cases because they increase the cost of synchronization.

3.4.7 Discussion

We have investigated variance in communication latency at various levels and clock jitter among FPGAs. We are not aware of other studies on these topics. We find that

these quantities are significant and in the case of link variance, surprisingly large. We also find, however, that the communication infrastructure based on the Altera MGTs and the Interlaken PHY link IP is sufficient to make these variances and jitter manageable.

The fact that there is substantial jitter has at least two effects: there must be flow control between communication and application modules (already in place in our case study) and the link protocol must have appropriate buffer sizes and flow control mechanisms. This second effect means that the communication IP necessarily cannot be too light-weight, which affects chip resources and latency. As we see from Table 3.2 the chip resources needed are negligible. Latency, however, through additional buffering and flow control, may be increased by nearly a factor of two over a lighter weight alternative.

On the positive side, the current setup gives seamless communication; only simple flow control and no additional buffering are needed on the application side, which indicates the viability of particular modes of higher-level packet processing which assume a synchronous flow model. These include offline routing and computation within the network (see, e.g., (Herbordt and Swarztrauber, 2003)).

Chapter 4

Network Architecture, Part 1: Router Design and Flow Control

This chapter describes the design choices in our router microarchitecture and its related flow control mechanism. We have created two router microarchitectures for two different workloads. The first router, which is described in section 4.2, mainly targets the workloads with dynamic communication. It is based on the conventional wormhole VC-based router (Dally and Towles, 2004); we assume credit-based flow control. To meet the hardware constraints of the multi-FPGA system, we apply several modifications in our router compared with the standard design.

Section 4.3 the router microarchitecture for workloads with static communication. It also has a pipelined structure, but it is VOQ-based. Its main feature is the hardware support for the collective operations. Its routing method is table-based, which facilitates application-aware statically-scheduled routing.

4.1 Background and Assumptions

As we introduced in the Chapter 2, we are the first to describe the implementation of a wormhole VC-based router on a multi-FPGA network built with MGT links. All the previous wormhole VC-based FPGA routers are for NoCs. Also, all the routers in the prior multi-FPGA systems are simple input buffering routers. Beyond the NoC level, there are huge number of complex VC-based ASIC router designs (see (Chen et al., 2011; Birrittella et al., 2015)). We narrow down the scope of VC-based routers

that are most likely to be cost-effective on FPGA clusters. We do this by observing several properties of FPGA clusters and extrapolating their implications to constrain the router design space.

The first assumption is that the user logic and the router are located on the same FPGA chip. The reason why we make this assumption is that the co-location of user logic and router is the key point to achieve the tight coupling of communication and computation. The second assumption is that there is only one router per node. Extra routers introduce both area and latency overhead. The parallelism benefits brought by extra routers can be maintained by proper application logic design. The third assumption is that the logic elements consumed by the router less than a quarter of the entire chip. More area means more VCs and better routing algorithms can be implemented in the router. However, we prefer more room is reserved for user's logic. We determine 25% as our budget because we find 25% is sufficient for our router design to achieve good performance. The fourth assumption is that the maximum operating frequency of the router is above 140MHz. The frequency provided by the MGTs is about 70 MHz, so the frequency of the router should be multiple times of that. If we use 70 MHz as the operating frequency of our router, we have to select the 256-bit as the flit size, which results in over-budge router area. Therefore, 140 MHz is the best choice for the operating frequency.

Since the previous FPGA VC-based routers have all been for NoCs, we continue our constraint of the design space by comparing the hardware characteristics of FPGA clusters with FPGA NoCs. We give these and the impact of the differences in Table 4.1.

In FPGA NoCs, the topology is most often a 2D-mesh or torus; the topology of clusters is much more flexible and generally higher dimension. For example, the Novo-G# is a 3D-torus. This higher dimensionality causes the number router of the

FPGA cluster to have more ports than that of FPGA NoC. A further impact of the difference in topology is that the more ports result in more complicated switch design.

The inter-node latency of FPGA clusters is 20-100 clock cycles while the that of FPGA NoC is negligible. Because of the negligible latency in FPGA NoCs, (Papamichael and Hoe, 2012) argues that pipelined router design is not suitable for router design of FPGA NoCs. On the FPGA cluster side, the long inter-node latency lets one-or-two extra stages in the router pipeline become insignificant.

The phit size in FPGA cluster is determined by the FPGA fabric and the MGT link IP, which is 256-bit. Typically, the flit size is bigger than phit size in NoC. If we were to stick with the NoC policy, then the router would consumes too much area because of the wide flit. Our decision is to use flit size smaller than phit size and increase the operating frequency.

As we stated above, the number of routers per chip for FPGAs in clusters likey to be one. In NoCs, we want to fit as many routers as possible. The bigger area budget on the FPGA cluster allows more pipeline stages, more VCs, and more room for the switch. Since in FPGA clusters, there are no global clocks, and the MGT links do not provide specific backpressure signals, the credit-based flow control is the most convenient way to synchronize the entire system and avoid packet drops.

4.2 Wormhole VC-based Router on FPGA Cluster

Since William Dally proposed Wormhole VC-based router in the early 1990s (Dally, 1992; Dally and Aoki, 1993), it has been well-studied for decades. In the following subsection, we are going to give a brief walk-through of the classical wormhole VC-based router microarchitecture.

Table 4.1: The comparison of hardware constraints between Novo-G# and NoC

	FPGA cluster	FPGA NoC	Impact
Topology	3D-torus	2D-mesh or torus	More ports
# of ports	6	4	More complicated switch
Inter-node latency	20-100 CCs	1-3 CCs	Pipeline stage # is unimportant
Phit size	256-bit	<32-bit	Flit size < Phit size
# of routers/chip	1	>10	More room for switch and VCs
global clocking	No	yes	Credit-based flow control

4.2.1 Classic Wormhole VC-based Router Microarchitecture

Figure 4·1 shows the block diagram of the conventional pipelined VC-based router. Each flit of a packet from the network arrives the router at an input unit. If it is a head flit of the packet, the routing computation unit performs one routing algorithm to determine the output port (or ports) where this packet should leave. Every input unit contains a set of virtual channels. A VC holds the arriving flits until there are available output ports to forward them. A VC also includes a finite state machine (FSM) that keeps the state of each packet and manages the signals come from the switch allocator.

After routing computation is performed on a packet, a VC is allocated to it by the VC allocator. Later on, the switch allocation is done by the switch allocator when multiple packets compete for a single output port simultaneously. The packet with the highest priority is granted for switch traversal. After switch traversal, the packet leaves the router from an output port.

Figure 4·2 shows a Gantt chart illustrating the pipelining of a classic virtual channel router. The four pipeline stages are routing computation (RC), virtual-

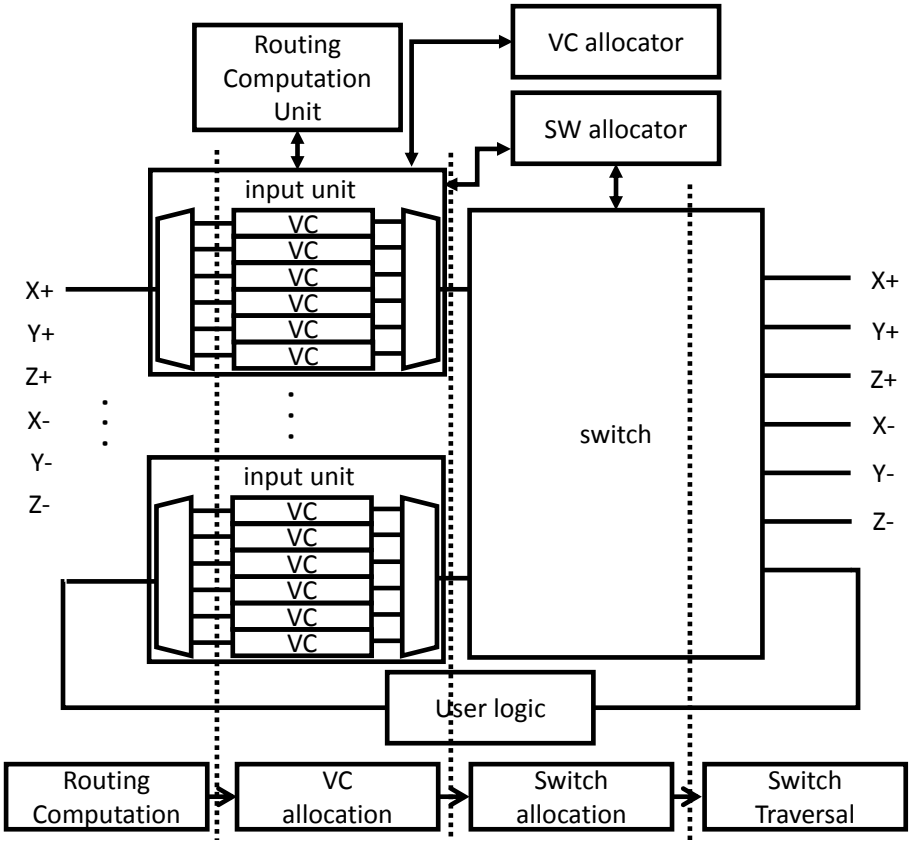


Figure 4·1: The classic wormhole VC-based router microarchitecture

channel allocation (VA), switch allocation (SA), and switch traversal (ST). Typically, each flit of one packet only spends one clock cycle at every stage. The RC and VA stages perform computation for the head flit only (once per packet). Body flits traverse through the two stages without computation. The SA and ST stages operate on every flit of the packet.Figure 4·2 shows a case where the packet advances through the pipeline stages without any stalls.

The VC-based flow control has two kinds of benefits. First, it overcomes the blocking problems of traditional wormhole flow control by allowing other packets to use the channel bandwidth that would otherwise be left idle when a packet blocks. One example illustrating comparing it with traditional non-VC routing is displayed in Figure 4·3. The upper half shows a router example without VC. The head flit of

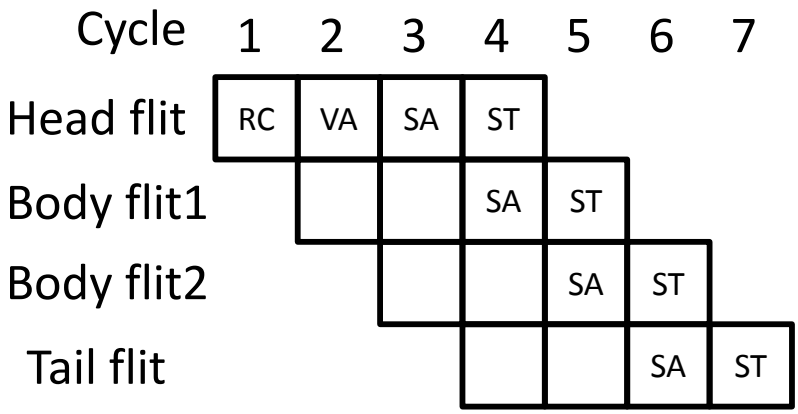


Figure 4·2: The classic Wormhole VC-based router pipeline

packet B is blocked at Node 2 because the downside port is not available. At this point, packet A also would like to go from node 1 to node 3 while passing node 2. The two channel p and q that the packet A would like to occupy is idle. However, it is still blocked because packet B is blocked. In the down half of Figure 4·3, by splitting channel P into two VCs, the blocking issue of packet A is solved. The second benefit of VC-based flow control is it can solve the deadlock problem by breaking channel dependency cycles. The details are described in next subsection.

4.2.2 Proposed VC-based Router on Novo-G#

Router Microarchitecture

The classical wormhole VC-based router microarchitecture is mostly studied on NoC. However, the hardware constraints of an FPGA cluster like Novo-G# is very different from those on NoC. Their differences are listed in Table 4.1. Because of these disparities, our proposed router microarchitecture has several significant differences from the classic one. Our design is illustrated in Figure 4·4.

The first significant difference is that there are two sets of asynchronous FIFOs at the front and back ends. The Stratix V FPGA fabric constrains the phit size to be 256-bit and the parallel clock provided by MGT IP to be around 71 MHz. Usually,

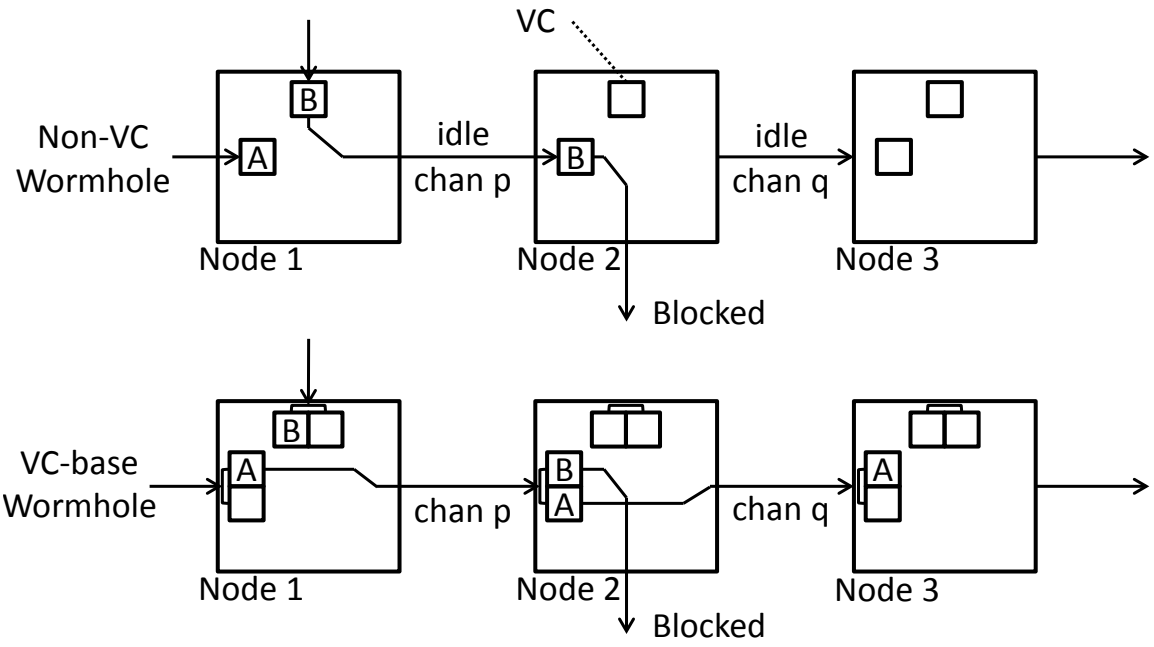


Figure 4·3: An example showing how VC-based flow control solves the blocking issue of typical wormhole flow control

flit size should be greater than phit size. However, if we select flit size larger than 256-bit, the router will consume an enormous amount of area on the FPGA. Also the 71 MHz is too slow for current FPGA devices. We select 128-bits as our flit size. This is natural since it leads to a better, but still conservative, clock frequency of 142 MHz. The two sets of asynchronous FIFOs at the front and back ends are used for the data to cross the two clock domains.

The second noticeable difference is that there is no particular port for local injection and ejection. In most NoC router designs, there is one extra port for local traffic (besides the ports for all the bypass traffic). In our router, the packet is ejected directly after the routing computation is done. Also, the packet is directly injected into the output network link when bypass traffic is not occupying it. Figure 4·4 shows the result: six injection and six ejection ports. There are three obvious benefits from this design. First, this yields the highest possible offered injection throughput and

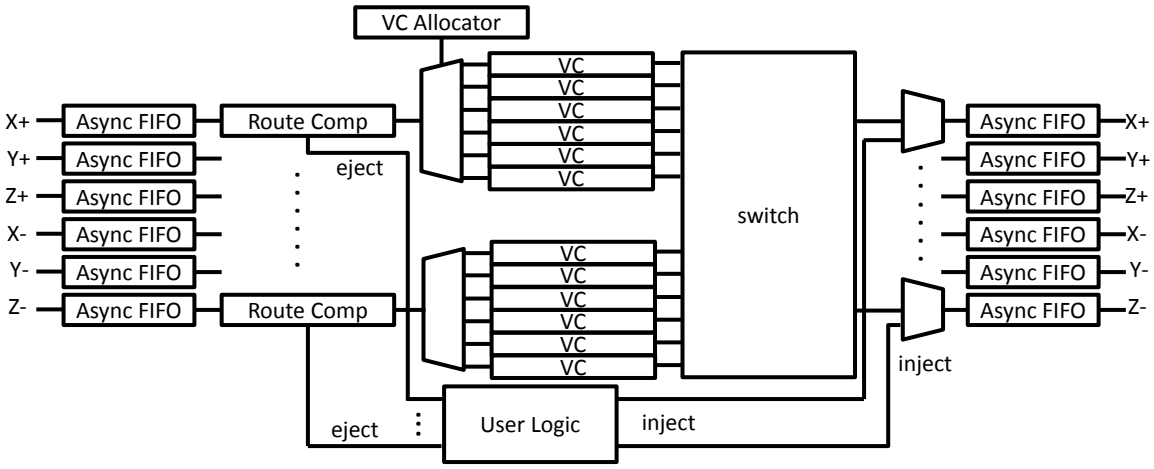


Figure 4·4: The proposed router microarchitecture on Novo-G#

can achieve the full outgoing bandwidth of each node. Second, the nearest neighbor pattern can achieve ideal performance. "Ideal" means congestion-free and 100% utilization of the link bandwidth. Third, the complexity of switch is reduced because there are fewer ports. There are also drawbacks. One is that, if the user's pattern does not need full outgoing bandwidth, extra logic is required to help users to determine which port to use for data injection. Another drawback is that the injection port selection has to satisfy the requirement of the deadlock avoidance, which we discuss later.

The routing algorithms in the routing computation stage are discussed in the next chapter. The VC allocation policy we used here is round-robin. The VC is implemented by a FIFO IP and a state machine.

The third difference is the connection between the VC and switch. There are three possible implementations (Dally, 1992), which are illustrated in Figure 4·5. The number of inputs and outputs that need to be switched increases when the number of virtual channels multiplexed on each physical channel is increased. If all the virtual channels are separately handled, as shown in Figure 4·5(c), the switch complexity is increased dramatically, because it is quadratically related to the port number.

However, the area budget for the router on NoC is quite tight. The NoC always desires to fit as many nodes as possible. Therefore, all the virtual channels multiplexed on the same physical channel are multiplexed again before they are connected to switch, as illustrated in Figure 4·5(a). In this manner, the switching complexity is not increased. Most NoC VC-based routers follow this method to compact their router size. But the compactness is not achieved by free. First, the input VC needs to be arbitrated not only for an output port but also for access to the switch. Second, the VC that is granted for switch traversal may block other VCs that are multiplexed from the same physical channel. One example is shown in Figure 4·6(a). After the routing computation, the output port of packet B is Y, which is idle. However, it cannot advance because the VC of packet A is the one that is granted for switch traversal, although it is blocked at output port X. In contrast, in Figure 4·6(b), this issue is solved by connecting all VCs to switch directly. Figure 4·5(b) is an intermediate solution. The switch provides separate inputs for all VCs and multiplexed outputs. It simplifies the arbitration of VCs. Only one input VC competes for switch output ports. However, the switch is still larger than the solution in Figure 4·5(a). But again, in FPGA clusters we just need to fit one router per chip and so in this case is affordable. So we adopt it as our router microarchitecture.

As we stated previously, our switch design provides separate inputs for all VCs and multiplexed outputs. Let us assume there are N input ports ($N = 6$ in our case), and each of them is multiplexed by M VCs. Then our switch is an $M \times N$ to N switch. Conventionally the NoC router implements the switch using a crossbar structure as shown Figure 4·7(a). If we also adopt this structure, it causes two problems. First, it requires complex arbitration logic. Second, it cannot easily scale. As stated above, the operating frequency must be higher than 142 MHz. When the number of VCs multiplexed on each physical channel is greater than 4, it is almost impossible to meet

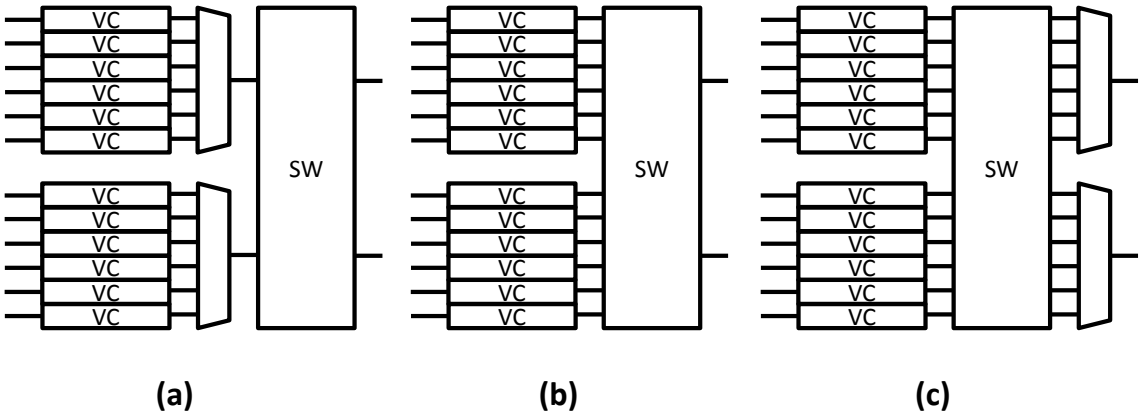


Figure 4·5: Three ways of implementing the connection between the VC and switch. (a)

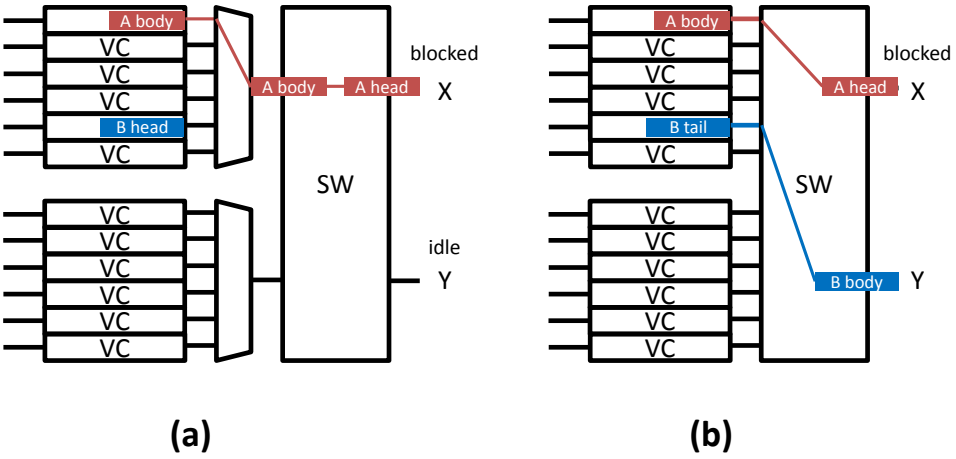


Figure 4·6: An example showing how multiplexing of multiple VCs on the same physical channels causes extra blocking

this requirement.

To address this issue, we propose a reduction-tree-based switch. An $M \times N$ reduction-tree-based switch is formed by having N M-to-1 reduction trees. Each reduction tree is formed with multiple levels of 2-to-1 and 3-to-1 reducers. One example of the proposed switch design is illustrated in Figure 4·7(b). This example shows a 6×3 switch. Each input port is directed to one of the three reduction trees based on their output port number (that is computed in the RC stage). Each reduction tree in this example has two levels of reducers. The first level is two 3-to-1

reducers, and the second level is one 2-to-1 reducer. In each N-to-1 reducer, there is one small FIFO with a depth of two for each of the N inputs. All N inputs compete for a single output. Once the head flit of a packet is granted, all the following flits are granted until the entire packet has passed this level of reducer. The input with the highest priority is granted for the output. Each flit spends one clock cycle at each level of reducer.

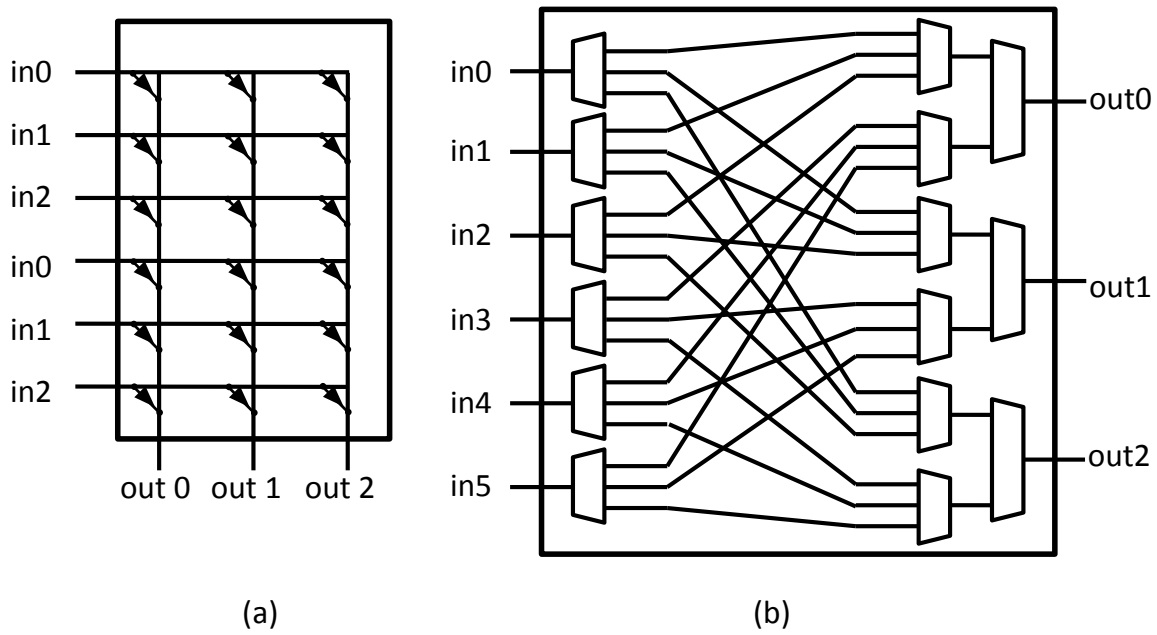


Figure 4·7: (a)The conventional crossbar switch, (b)the proposed reduction-tree-based switch

The reason we only use 3-to-1 and 2-to-1 reducers to form the reduction tree is that reducers equal to or larger than 4-to-1 cannot satisfy the Fmax requirement. In Table 4.2, we list the configuration of the reduction tree for the different numbers of VCs multiplexed on each physical channel with the optimal area consumption. The first column is the number of VCs multiplexed on one physical channel. The second column is the switch size (the number of input ports $\times$ the number of output ports). The third column shows the types of reducers on each level. For example, 3×2 means, from input ports to output ports, the first level is built with 3-to-1

reducers; the second level is made with 2-to-1 reducers. The number of reducers on each level can be easily computed. The fourth column is the how many ALMs are used after synthesizing this router on an Altera Stratix V chip. The last column is the percentage of ALMs used on the entire chip. We stop at 9 VCs per physical channel because we believe 25 % of entire chip resource is the upper limit for a router. Our results later show that 25% is sufficient for most workloads to achieve maximal performance. Since each flit spends one clock cycle at each level of the reduction tree, from Table 4.2, we can tell that the delay of our proposed switch is 2-5 cycles, depending on how many VCs are multiplexed on a physical channel.

Table 4.2: The configuration of the reduction tree for different number of VCs multiplexed on each physical channel with the optimal area consumption.

VC # muxed on a phy ch	switch size	reduction tree configure	ALMs	area %
1	6×6	3×2	12980	5%
2	12×6	$3 \times 2 \times 2$	18980	7%
3	18×6	$3 \times 3 \times 2$	24860	9%
4	24×6	$3 \times 2 \times 2 \times 2$	30980	12%
5	30×6	$3 \times 3 \times 2 \times 2$	42470	16%
6	36×6	$3 \times 3 \times 2 \times 2$	42470	16%
7	42×6	$3 \times 2 \times 2 \times 2 \times 2$	54980	21%
8	48×6	$3 \times 2 \times 2 \times 2 \times 2$	54980	21%
9	54×6	$3 \times 3 \times 3 \times 2$	60500	23%

Deadlock Avoidance

Deadlock happens when a sequence of packets cannot make progress because they are all waiting for other packets to release resources, and the resource dependency relationship forms a loop (Dally and Towles, 2004). Deadlock is disastrous to a network. Not only do the packets that are deadlocked never reach their destinations, but also all the resources occupied by these packets are virtually removed from the

network.

The deadlock could happen in two ways in our 3D-torus network. We provide deadlock avoidance techniques for both of them to make sure that the 3D-torus network in Novo-G# is deadlock-free.

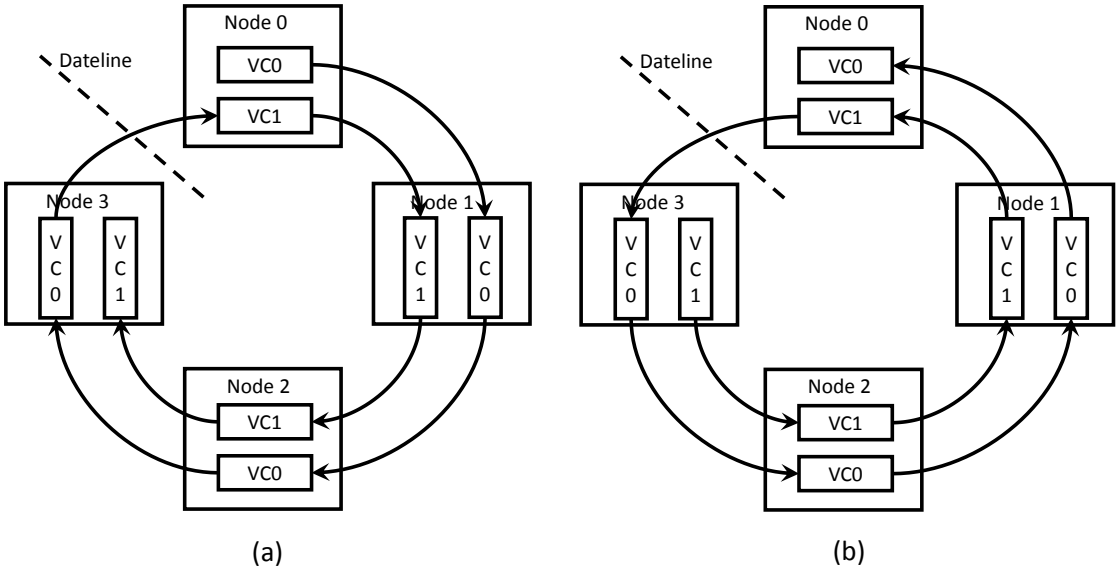


Figure 4·8: Virtual channels divided into dateline classes to break the ring loop in torus, (a) packets on the clockwise direction, (b) packets on the counterclockwise direction

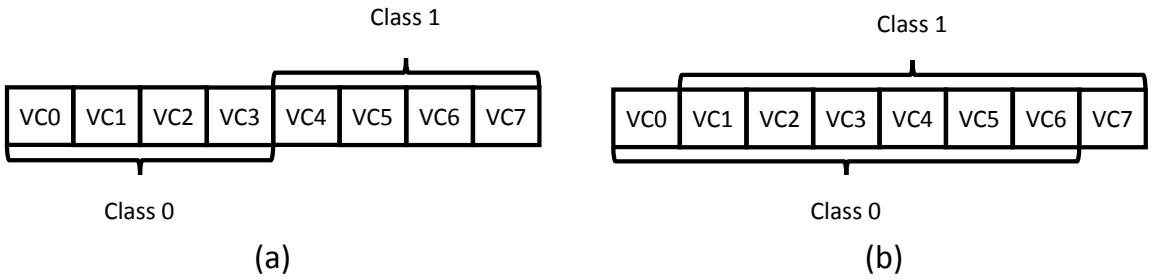


Figure 4·9: Two methods to partition 8 buffers into 2 classes.

In torus networks, the wraparound link forms a ring at each dimension. The authors in (Dally and Seitz, 1986; Dally and Seitz, 1987) propose that we can break

the resource dependency loop by applying VCs and the dateline technique. The details are illustrated in Figure 4·8. All the VCs on each node are divided into two classes. We set a dateline between Node 0 and 1. When the packets are injected into the clockwise ring, they start in VC class 0. They are only allowed to use the resources belong to the VC class 0 before crossing the dateline. After crossing the dateline, they are only permitted to use the resources belong to VC class 1. The routing algorithms adopted by us ensures that the packets will not cross the dateline by twice. Therefore, by splitting VCs into two dateline classes, we can make sure deadlock will not happen on the 1D rings of the 3D-torus network. Figure 4·8(b) shows the case when the packets are injected into counterclockwise direction. The packets that start from the VC class 1 are only allowed to use the resources belong to the VC class 1 before crossing the dateline. After crossing the dateline, they are only authorized to use the resources belong to the VC class 0.

When implementing the dateline technique, we set a VC class bit in the packet header, which is a small overhead. Depending on whether the injection direction is positive (X+, Y+, or Z+) or negative (X-, Y-, or Z-), the VC class bit is initialized to be 0 or 1, respectively. The VC class bit is toggled only when the packet crosses the dateline or changes dimension. When crossing the dateline, the VC class is toggled. When changing dimension, the VC class bit is set to be 0 or 1, depending on whether the new direction is positive or negative.

An obvious problem of dividing the VC channels into two classes is that it doubles the buffer space required for VCs, while keeping the performance mostly unchanged. However, most packets do not cross the dateline, which makes the utilization of the two classes quite imbalanced. The load imbalance lets wastes the idle VCs. Our method to address this problem is to overlap buffer space for the two classes (Dally and Towles, 2004) (see Figure 4·9). In Figure 4·9(a), the first four buffers are assigned

to VC class 0 and the other four buffers are assigned to VC class 1. The imbalanced distribution of the two VC classes results in oversubscribing the first four buffers while (generally) starving last four buffer. In Figure 4·9(b), six out of eight buffers are shared between two VC classes. Only the first and last buffers are exclusively used by VC class 0 and 1, respectively. Although most of the buffers are shared by both dateline classes, the correctness of the deadlock avoidance is not altered. The fundamental idea behind maintaining deadlock freedom, despite having a cyclic channel dependence graph, is to provide an escape path for every packet in a potential dependency cycle. As long as the escape path is deadlock-free, packets can move more freely throughout the network, possibly creating cyclic channel dependencies. However, the existence of the escape route ensures that if packets ever get into trouble, there still exists a deadlock-free path to their destination. Although the two VC classes share most of the buffers, the two exclusive buffers ensure there is at least one deadlock-free path, which will lead packets to their destinations.

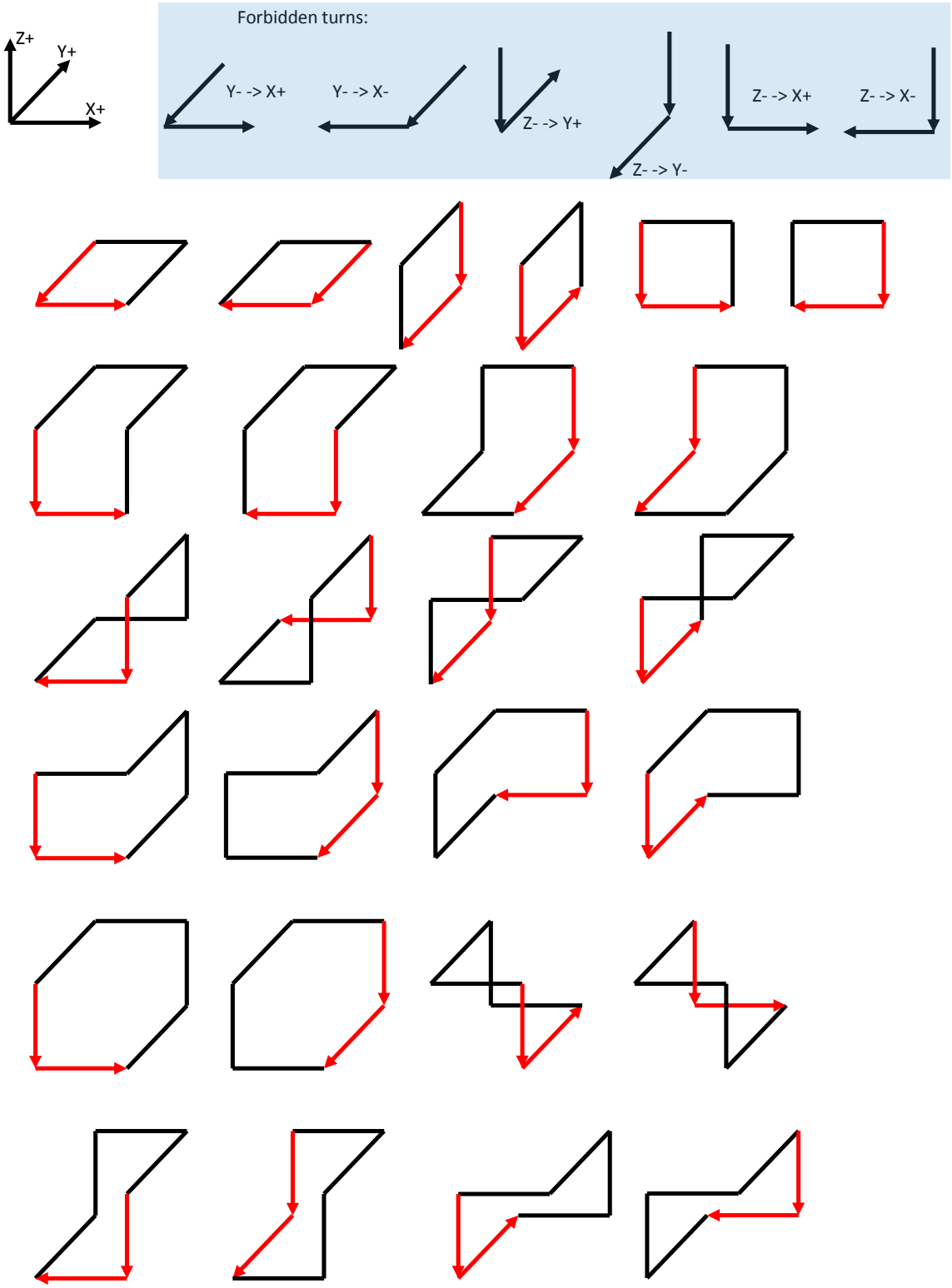


Figure 4·10: Six forbidden turns break all of dependency cycles in 3D-torus

Another category of deadlock resides in the dependency loop across multiple dimensions. To solve this problem, we forbid six turning directions and so break all the possible dependency loops. Figure 4·10 lists all the possible channel dependency loops in the 3D-torus and shows how the six forbidden turns break all of these cycles. The first two forbidden turns (Y- to X+ and Y- to X-) avoid the deadlock in XY plane. The last four forbidden turns (Z- to X+, Z- to X-, Z- to Y+, and Z- to Y-) ensure deadlock-free in YZ and XZ planes. The last four forbidden turns also define our turn forbidding policy to be Z- last. After a packet goes into Z- direction, it cannot turn anymore. The Z- last ensure that all the loops covering multiple 2-D planes are broken because all the loops covering multiple 2-D planes must have an edge on Z- direction.

Packet Formatting

The network based on our router design supports arbitrary size packets. To achieve this, we support five types of flits: head flit, body flit, tail flit, single flit, and credit flit. Usually a packet starts with a head flit, which is followed by several body flits, and ended by a tail flit. If the packet has only a single flit, its flit type is single flit. The credit flit is used to transfer credit information from downstream nodes to upstream nodes.

All kinds of flits have a flit-type field and a payload field. The payload field in the credit flit contains the credit information of a downstream node; the payload field in other flit types contains the payload data. A single or head flit also includes the destination field, VC class field, and priority field. The priority field is used when multiple packets are competing for the same output port. Its content might be packet age, distance from the destination, or both. How the priority fields are used to arbitrate packets is discussed in next chapter.

Credit-based Flow Control

When the buffer space on the downstream node is used up, the downstream node must generate a backpressure signal to the upstream node to notify it to stop sending data. We use credit-based flow control to manage the inter-node backpressure. The implementation of credit-based flow control on Novo-G# is different from that in conventional NoC.

First, the MGT link IP does not provide any functionality for backpressure. Also, the credit information should be multiplexed into the normal data flow. In the previous subsection, we introduced a particular credit flit type for transferring credit from downstream nodes to upstream nodes. Another undecided parameter is the frequency of sending backpressure credit flits. In NoCs, the credit flit is usually piggybacked on each data flit (Dally and Towles, 2004). Because the short inter-node latency on NoC, this solution can ensure the upstream node is always aware of the most recent credit information on neighboring nodes. The overhead is significant. Half of the link bandwidth is consumed by sending credits. Since the inter-node latency in Novo-G# is 20-100 cycles, it is impossible for the upstream nodes to be aware of the real-time credit information on neighboring nodes. And we do not want to let the credit consume too much valuable bandwidth resources. Therefore, in our design, we sent one credit flit from downstream node to upstream node every 100 cycles. There is a credit threshold used by upstream nodes to decide to send packets or not. The threshold equals the link latency plus the 100. When the value of credits is lower than this threshold, the upstream node stops sending. All the in-flight flits are guaranteed buffer space.

4.3 Proposed Statically-scheduled Collective Acceleration Router on Novo-G#

We have designed and implemented an entire solution for statically-scheduled (offline) routing on an FPGA cluster. This solution includes a new router design that supports both online and offline routing (simultaneously) and a new offline routing algorithm for collectives. We assume that collective communication patterns have been extracted from the application (as is done, e.g., in (Grossman et al., 2015)). We then use our offline routing algorithm to build an optimized tree topology for each collective operation. These are fed into scripts to generate the routing, multicast, and reduction tables. Finally, the routing data are downloaded into the appropriate tables within the routers. In the rest of this section, we describe the how we implement the table-based routing and the router microarchitecture.

4.3.1 Table-based Routing

Table-based routing is a perfect fit in FPGA clusters because routing tables need to be accessed very frequently and might be spread on multiple network ports. FPGA has rich on-chip distributed block RAMs with fast access time, which is very suitable to implement routing tables.

Table-based routing can be implemented in two ways: source routing and node-table routing (Kinsy et al., 2013). Both are widely used in network routers (Kinsy et al., 2013; Murali et al., 2007). Since source routing requires packets to carry table indexes, which consumes extra bandwidth, we instead use node-table routing. There the resident routing table preserves a table entry for each incoming packet (see Figure 4·11).

In this example, node A dispatches a multicast packet that carries three fields: packet type, table index, and payload. The router routes the packet to either a

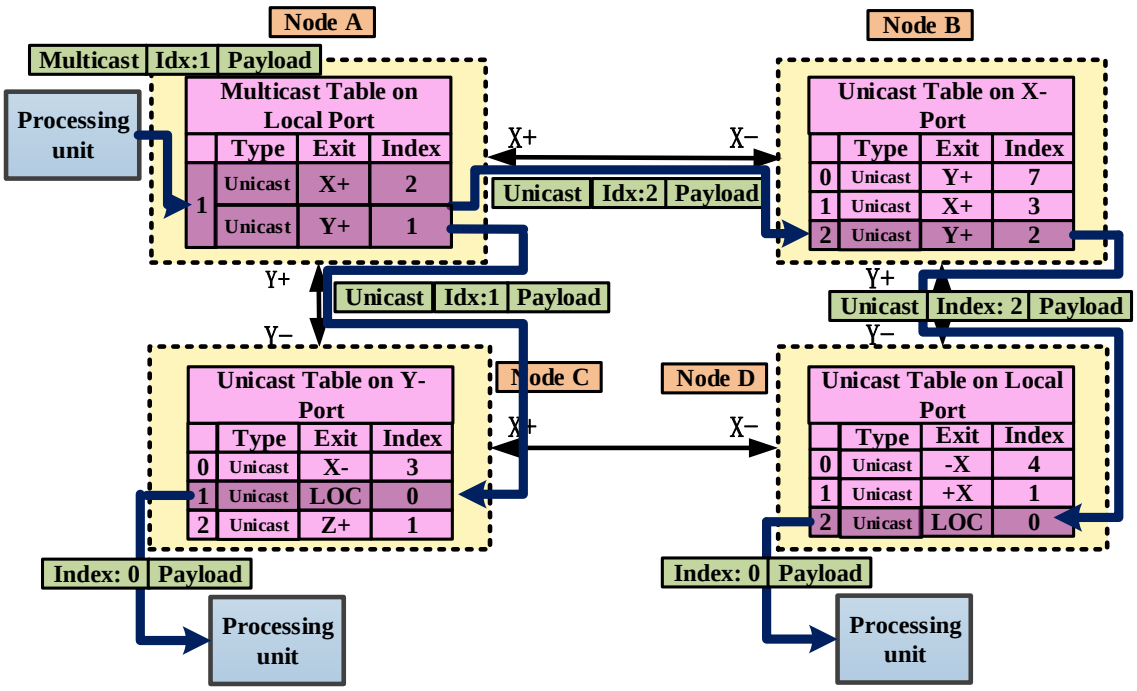


Figure 4·11: An example of node-table routing

unicast, multicast, or reduction table based on the packet type. In the corresponding table, multicast in this example, the router looks up the table entry based on the index field in the packet. The multicast table entry has slots for all of the six possible fan outs.

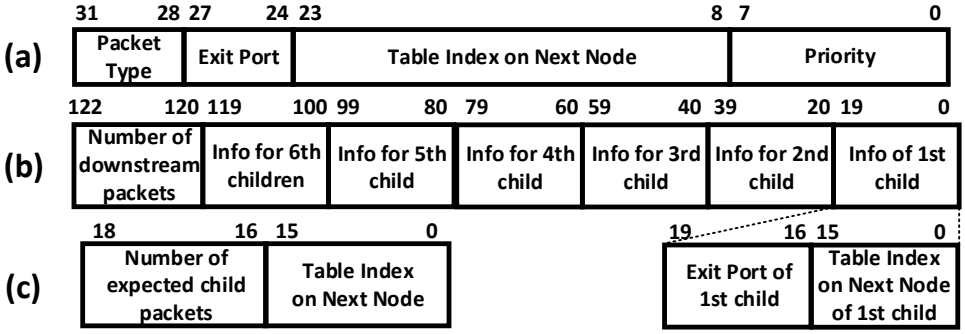


Figure 4·12: (a) Unicast table entry format (b) multicast table entry format (c) reduction table entry format

Figure 4·12 shows the data formats for the various types of table entries. In this

example, the multicast packet has two fan outs. For the first fan out, the multicast table entry shows that it is a unicast packet, that it should be routed to the X+ port, and that the table index for next node is 2. The router then generates a unicast-type packet and routes it to the X+ port, whence it enters the X- port on node B. On node B, since it is a unicast packet, the router will look up the entry in the unicast rather than the multicast table. The table entry shows that this packet should be routed to the Y+ port of node B. In the same manner, the router on Node B sends the packet to the Y- port of Node D, at which point it is ejected. Similarly, for the second branch, the packet is routed to the Y- port of Node C, where it is ejected.

4.3.2 Router Microarchitecture

We explored two kinds of router microarchitectures in this section: ring and Virtual Output Queue (VOQ) router.

Ring Router

Rings have linear area cost and constant fan-in/fan-out and so have been adopted in many small-scale, multicore architectures including the IBM Cell (Pham et al., 2006) and Intel Xeon Phi (Intel, 2013). Here we use a seven-node ring topology based on a light-weight ring-based router microarchitecture (Kim et al., 2007). We augment the design by incorporating support for offline routing.

Since the design is a 3D-torus, there are six links for each node. As shown in Figure 4·13, on the chip there are seven routers, six to handle the internode communication and one to deal with the traffic to and from the local processor. The seven routers are connected with a bidirectional ring. The bidirectional ring is composed of two separate bidirectional buses. One bus is the data bus managing the intranode communication. The other is the control bus to handle the flow control including the generation of back pressure (to the previous router when the FIFO in the next router is full). In

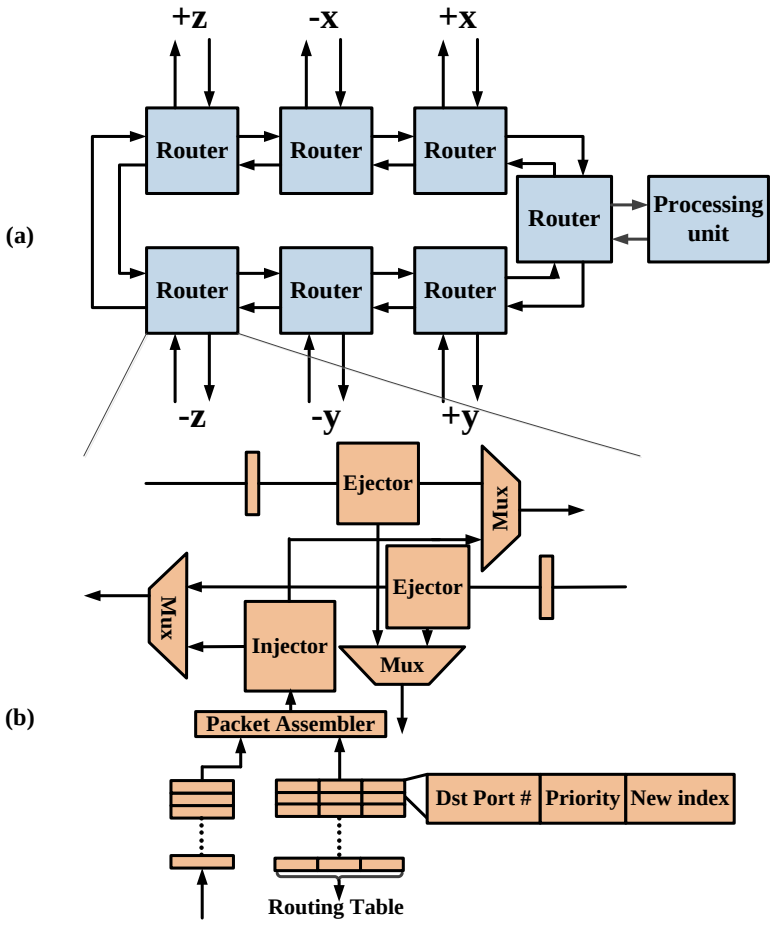


Figure 4·13: (a) 7-node ring topology and (b) router microarchitecture for the ring router

Figure 4·13, note that the two ports for the same dimension (e.g., +x and -x) are placed next to each other. This setting is because most of the time the packet stays on the same dimension.

Inside the router, there are three input FIFOs: one for injecting the packet either from other nodes or the local processing unit, and two for input traffic from the two different directions of the ring. These latter two FIFOs each contain a single register. As we said before, we use table-based routing in our design. Packets outside the ring include a short header which indexes the table in the next node. When injected into a new node, the router looks up the entry corresponding to the index on the packet

header in the routing table. Besides the exit port number and index, the entry also contains a priority field; this helps arbitrate contending packets. The priority policy is currently farthest first. The router is flexible enough to support nearly arbitrary and fine-grained policies.

After attaching a new tag to the incoming packet, the packet is routed on the shortest path, clockwise or counterclockwise, to the exit port. If the packet has a higher priority than a packet already in the ring, and the input FIFO in the next router is not full, then the current router will deliver the packet to next router (and so on), until the packet reaches the exit router. At the exit router, if the packet has higher priority than the packet on the other direction which is going to exit at the same router, it will leave this node first and be sent to the other adjacent node through the MGT. If there is no congestion, the time spent on each router will be one clock cycle. In general, each packet will spend 2-4 cycles on each node.

VOQ Router

The VOQ router is similar to the VC-based router. The main difference is how virtual channels (queues) are organized. In the VC-based router, the packets from every input port are allowed to be allocated by multiple VCs multiplexed on this port. In VOQ router, all the virtual output queues multiplexed on every input port has one-to-one mapping relationship with the all the output ports. So the packets from every input port can only be allocated by one VOQ associated with the output port that this packet will be routed. Because of this one-to-one mapping relationship, the VOQ router has a much simpler switch than the VC-based router.

Our VOQ router architecture has a similar staged pipelined structure of the conventional four-stage pipelined Virtual-Channel (VC) router (Dally and Towles, 2004). By adding support for multicast and reduction, the four-stage pipeline is extended to a seven-stage pipeline. Its architecture is illustrated in Figure 4·14.

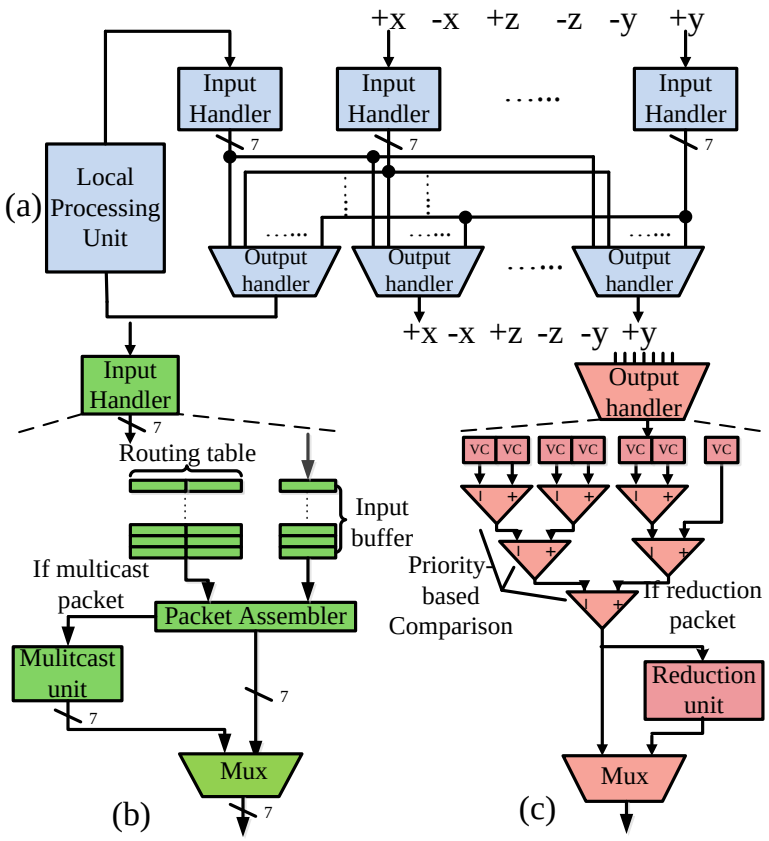


Figure 4·14: VOQ router microarchitecture: (a) The VOQ router is connected by seven input handlers and seven output handlers. (b) The input handler has four stages: input buffer consumption, routing table lookup, multicast table lookup, and virtual output queue allocation. (c) The output handler has three stages: switch allocation, reduction table lookup, and reduction table write-back.

Figure 4·15 shows the original and modified pipelines. The original pipeline has four stages: routing computation (RC), virtual channel (queue) allocation (VA), switch allocation (SA), switch traversal (ST). The first difference is that we divide the RC stage into three stages: input buffer consumption (IC), routing table lookup (RL), and multicast table lookup (ML). When a packet is injected into switch input buffer, during the IC stage, we spend one cycle to fetch the flit from the buffer. We examine its header, to obtain its routing table index; this is used during RL to find the routing table entry. At that moment, the router knows whether the packet is

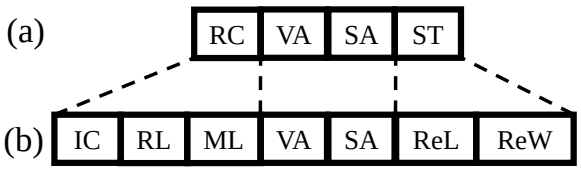


Figure 4·15: (a) The classic four stage router pipeline and (b) our proposed seven stage pipeline supporting mulitcast and reduction

multicast or not. If so, then during ML we look up its multicast table entry based on the index from the routing table entry. The VA stage allocates VOQs to all the multicast children that are generated during ML. If VA fails, the router creates back pressure to stall the pipeline. In the SA stage, there might be multiple packets (up to 7) contending for the same output port. The packet with the highest priority, which is determined during the RL stage, wins the arbitration. Our current priority scheme is farthest-first.

Another difference between this VOQ router and the VC-based router is that we divide ST stage into two stages: reduction table lookup (ReL) and reduction table write-back (ReW). If the packet is not a reduction packet, it still traverses the last two stages (a bypass is certainly an option). If the packet is a reduction packet, it is routed to the reduction unit. We allocate one entry in the reduction table for each reduction operation. During ReL, the reduction packet checks for its corresponding entry and whether the expected number of downstream packets have arrived. If not, then the reduction operation is executed and the reduction table entry updated. If all the expected downstream packets have arrived, the reduction unit dispatches a new packet and injects it into the upstream link.

Comparison Between Ring and VOQ Router

Regarding latency, the ring router can perform similarly to the VOQ only when one-to-one communication is the primary communication operation. When collective

Table 4.3: The resource utilization of two proposed router architecture (ring and VOQ) on an Altera Stratix V 5SGSMD8

Router Architecture	# of Logic	# of Registers
Ring	2485	3568
VOQ	3312	2672

operations, such as multicast and reduction, are the major workload, then the traffic saturates the bandwidth of the ring quickly (because of packet duplication). So in the next chapter we only use both ring and VOQ router in 3D FFT implementation because there is no collective operation in 3D FFT. For collective workloads, we only use the VOQ router.

We make a comparison of their area comparison on FPGA. The results are demonstrated on table 4.3. For fairness, the VOQ router in this table does not have collective hardware support. The results show that two routers have almost the same area consumption, which are both trivial compared with the total logic resources on the chip.

Chapter 5

Network Design, Part 2: Routing Algorithms and Switch Arbitration Policies

The routing algorithms and switch arbitration policies of the two router microarchitectures are different. For the VC-based router, proposed in section 4.2, we mainly implement several kinds of existing general routing algorithms and switch arbitration policies. These algorithms and policies are targeted for unicast workloads. We do, however, make some modifications to adapt them to our network hardware. Details and implementations are shown in sections 5.1.1 and 5.1.2.

We find there are no universal (optimal) routing algorithms and switch arbitration policies. That is, for different workloads, the algorithms and policies that can provide the best performance are different. Because of the large number of alternatives (including those for future study), we created a cycle-accurate simulator to evaluate different combinations of these algorithms and policies; this is described in section 5.1.3. Based on this simulator, we propose a framework that can help users find the best router configuration for their applications (described in section 5.1.3). We define an interface standard for new router components, such as for routing computation logic or switch arbitration. Users can easily include their new router components into our framework as long as they conform to the interface standard.

For the statically-scheduled collective acceleration VOQ router in section 4.3, we propose and implement a novel offline collective routing algorithm. The details and

evaluations of it are shown in section 5.2.

5.1 Implemented Routing Algorithms and Switch Arbitration Policies for Unicast workloads

For our proposed VC-based router, which is described in the previous chapter, we implemented five routing algorithms and three arbitration policies for a total of 15 combinations. There are still an enormous amount of other algorithms and policies that we have not implemented. All of these algorithms and policies form a big configuration space for our proposed VC-based router. For different workloads, it is unknown which configuration is optimal. We propose a framework that is based on a cycle-accurate simulator to search for an optimal router configuration given a particular workload and evaluation metric.

In this section, we first introduce the five routing algorithms and three arbitration policies we implemented. There follows the framework to search for an optimal router configuration, including the implementation details of the cycle-accurate simulator. Finally, we evaluate the framework by comparing the performance of the optimal router configuration with the configuration that has the best average performance over all the workloads tested.

5.1.1 Implemented Routing Algorithms in Proposed VC-based Router

We implement five routing algorithms in our proposed VC-based router for unicast workloads. Four of them are oblivious algorithms. The remaining one is an adaptive algorithm. All the oblivious algorithms can be implemented either dynamically, using on-chip FPGA logics, or statically using routing tables. The adaptive algorithm can only be implemented dynamically using on-chip FPGA logic. In this subsection, we describe how this all works, how it ensures freedom from deadlock, and how it is implemented in Novo-G#.

1. **Dimensional Ordering Routing (DOR)** The first routing algorithm is Dimensional Ordering Routing (DOR) (Sullivan and Bashkow, 1977; Glass and Ni, 1992). In a 3D-torus, there are six possible DOR routing algorithms (XYZ, XZY, YXZ, YZX, ZXY, and ZYX). In general, they have no difference in performance. We arbitrarily select XYZ routing as our implementation. In XYZ routing, all the packets must start from X dimension and are not allowed to enter Y dimension links until their current position has the same X coordinates as their destination. Then they are not authorized to enter Z dimension links until their current position has the same X and Y coordinates as their destinations. XYZ routing is an oblivious routing algorithm. Its rules can easily be implemented in hardware with little cost. Many large-scale parallel systems use it because of its simplicity.

2. **Randomized, Oblivious, Multi-phase, Minimal Routing (ROMM)** As indicated by its name, ROMM is a class of randomized, oblivious, Multi-phase, Minimal routing algorithms (Nesson and Johnsson, 1995). In a 2-D mesh, ROMM routing randomly selects p intermediate nodes in the minimum sub mesh constrained by source and destination. And the routes between the two nodes are based on the DOR routing. The number of phases is determined by the number of intermediate nodes. We make some modifications to original ROMM to adapt it to 3D-torus network in Novo-G#.

   The first modification is that the intermediate nodes are selected in the minimal subcube (instead of submesh) constrained by the source and destination. The second modification is that we have no constraints on the number of phases. Whenever the packet arrives at an intermediate node, it has at most three possible routing directions because of the minimal-path constraint. The actual routing decision is made among all the possible directions uniformly at random.

The third modification is that in conventional ROMM routing, p VCs are used to avoid deadlock in a p-phase ROMM routing. Whenever the phase order is increased by one, the VC order should increase by one as well. In our ROMM routing implementation, we abandon this setting because we do not have a specific number of phases. Our deadlock avoidance is already defined in section 4.2.2. All the allowed routing directions have to conform the forbidden-turn rules shown in Figure 4·10. ROMM routing is also an oblivious routing algorithm. When implemented in hardware, it needs a 2-bit pseudo random number generator at each routing computation unit, which can easily be implemented using a 3-stage LFSR.

3. **Orthogonal One-turn Routing (O1TURN)** O1TURN is also a minimal, oblivious routing algorithm. Authors in (Seo et al., 2005) claim O1TURN can achieve near-optimal worst-case throughput. O1TURN is similar to DOR routing. The difference is that DOR routing always uses one DOR path all the time, while the O1TURN randomly select one of all the possible DOR routes. For example, in a 3D-torus, there are up to six possible DOR routes: XYZ, XZY, YXZ, YZX, ZXY, and ZYX. The O1TURN algorithm randomly selects one from these six routes for each packet.

   When implementing O1TURN on our proposed router, we need to implement a 3-bit pseudo-random number generator at each routing computation unit. Besides this, the packet head flit needs to contain an extra 3-bit field to represent which route this packet is following. To ensure freedom from deadlock, if any routes among those six violate the forbidden-turn rules (in Figure 4·10), it should be removed from the list of O1TURN routes.

4. **Randomized Load Balance Routing (RLB)** RLB is a non-minimal, oblivious routing algorithm that offers a balance between locality and load balance in

torus networks (Singh et al., 2002). The RLB routing still follows DOR routing pattern, first routes on X dimension, then Y dimension, finally Z dimension. However, on each dimension, it does not follow the minimal routing. On each dimension, the torus ring has two possible directions. Most of the time, only one of the two routes of the two directions is minimal. RLB algorithm allows routing packets in both directions. Let us assume the number of nodes on one torus ring is N, and the distance between destination and source is P. If $P < N$, the probability that the packet is routed to the shorter path is $(N - P)/N$, while the probability that it is routed to the longer path is P/N. It is evident that the packet is still more likely to take the minimal path on each dimension.

The RLB routing meets the deadlock-free requirement automatically since it takes XYZ routing. Since it is a non-minimal routing, it might have livelock issue. Surprisingly, it is inherently livelock free. Once a route has been selected for a packet, it monotonically advances along the route, reducing the number of hops to the destination at each step. Since there is no incremental misrouting, all packets reach their destinations after following a predetermined, bounded number of hops (Singh et al., 2002). Implementing RLB routing is slightly more complicated than XYZ routing. Compared with XYZ routing, it requires an extra pseudo random number generator at each routing computation unit. The width of the generator depends on the size of the network.

All the above four algorithms are oblivious routing algorithms. They can all be implemented dynamically with small on-chip FPGA logic resources. They can also be implemented statically with routing tables.

5. **Credit Count Adaptive Routing Algorithm (CCAR)** There is a vast literature on different adaptive algorithms. Given the information of free and allowed output ports, an output direction could be chosen randomly (Feng and

Shin, 1997) or based on the remaining hop count in each dimension (Badr and Podar, 1989). An algorithm called NOTURN tries to avoid turns by following a dimension until it is either exhausted or blocked (Glass and Ni, 1994). These three algorithms are unaware of congestion information of neighboring nodes. Therefore, they are inherently unable to deal with the unbalanced issue.

There are other algorithms gather the local/global congestion information before making routing decisions. (Dally and Aoki, 1993) counts the numbers of idle VCs on neighboring nodes and route packet to the direction with the largest number of VCs. If we want to implement this algorithm in our network, we need extra logic to monitor the idle VC numbers. And additional bandwidth is consumed to transfer this information to neighboring nodes. (Singh et al., 2003) counts the available slots at the output buffers to determine routing decisions. In our router design, we do not have output buffer. The Credit Count Adaptive Routing Algorithm (CCAR) (Kim et al., 2005b) gathers the credit information on neighboring nodes. The credit gathering mechanism already exists in our credit-based flow control. All the routing computation logic we need to implement is to find the direction with the most credits among all the legal directions. The legal directions are the directions meets the forbidden turning rules in Figure 4·10.

5.1.2 Implemented Switch Arbitration Policies in Proposed VC-based Router

We design a switch based on multi-level reducers. Whenever two head flits from two packets collide at the same reducer, a switch arbitration policy is needed to decide which packet wins the arbitration. We implement three switch arbitration policies in our proposed VC-based router. In this subsection, we describe how it works, and how it is implemented in Novo-G# for each policy.

1. **Farthest First (FF)** Farthest first means the packet farthest from its destination has the highest priority. It is fairly easy to implement this policy. The head flit of each packet already contains destination. The distance to the destination could either be computed at each hop or carried along as one field in the head flit. In our implementation, we create a particular field in every head flit keeping the distance to the destination. Whenever the packet passes one routing computation unit, this field is deducted by one. The width of this field depends on the network size.

2. **Oldest First (OF)** Oldest first means the packet with the oldest age has the highest priority. It is even easier to implement than FF. The head flit of each packet should contain an age field. From our experiments, we find that 16-bit age is sufficient for most workloads.

3. **Mixed** The mixed policy is a combination of FF and OF. The arbitration starts with farthest first. If one packet's age is higher than a threshold, it has the higher priority than all the packets with ages lower than the threshold, no matter how close it is from its destination. If multiple packets' ages are greater than the threshold, the one with the oldest age wins the arbitration. When implementing this policy, we create both age and distance field in the head flit of each packet.

5.1.3 Proposed Application-ware Framework to Generate Optimal Router Configuration

As described in the previous two subsections, we implemented five routing algorithms and three switch arbitration policies. These algorithms and policies form a search space with fifteen different router configurations. For a given workload, the users could have difficulty determining which configuration is optimal for their goals. Given

a workload and its evaluation metric, we propose a framework that can evaluate all the possible configurations and produce the best one. This framework has good extensibility with new algorithms and policies.

Proposed Extensible Cycle-accurate simulator

One key component of this framework is the cycle-accurate simulator. The cycle-accurate simulator is implemented in C++. Every hardware module in the RTL model has a corresponding class in the simulator. These classes are organized in the same hierarchical structure as the RTL model. To give the cycle-accurate simulator good scalability with new algorithms and policies, we define an interface standard for all the hardware modules, which is illustrated in Figure 5·1. This figure shows that each module has two interfaces: upstream and downstream. In the upstream interface, the *data in* is a pointer pointing to the *data out* of the upstream module. The *data in latch* is the dereference of that pointer. *in avail* is the backpressure boolean value to the upstream module. When *in avail* becomes false, the upstream module should hold the current *data out* until *in avail* becomes true. In the downstream interface, *data out* is the data passed to the downstream module. *out avail* is a pointer pointing to *in avail* from the downstream module. The *out avail latch* is the deference value of that pointer. Similarly, if *out avail* is false, then *data out* should hold the current value until *out avail* becomes true.

With the interface defined in Figure 5·1, we can conduct cycle-accurate simulations simply with a producer-consumer model. Every module is both a producer and consumer. Every clock cycle is divided into a consuming step and a producing step. Correspondingly, every module has two important member procedures: consume and produce. The pseudocode is illustrated in Algorithms 1 and 2. The pseudo code for advancing a clock cycle is shown in Algorithm 3.

Before starting a simulation, all the modules call an initialization function, which

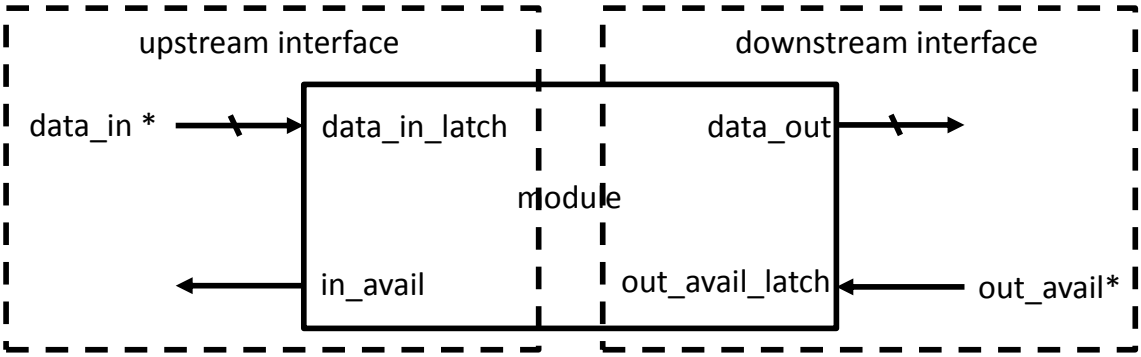


Figure 5·1: The standard interface for modules in the cycle-accurate simulator

Algorithm 1 Consume procedure of a module

```
1: procedure CONSUME(void)
2:     DataInLatch = *DataIn
3:     OutAvailLatch = *OutAvail
4:     for each submodules i in this module do
5:         submodule[i].consume()
6:     end for
7:     Return
8: end procedure
```

connects all the modules in the same hierarchical structure as the HDL code. The initialization mainly acquires a pointer copy of *DataOut* of the upstream module(s) into the *DataIn* pointers of the current modules, and a pointer copy of *InAvail* of the downstream module(s) into the *OutAvail* pointers of the current modules. It is possible to have multiple upstream and downstream modules.

The initialization function is shown in the Algorithm 4. When a module calls the initialization procedure, all submodules under it call the initialization procedure recursively. As shown in Algorithm 3, each clock cycle advance is divided into two steps. In the first, all the submodules under the top module call the consume procedures recursively. When calling the consume procedure as shown in Algorithm 1, one module latches all the *DataIn* and the *OutAvail*. Then this module invokes the consume procedures of all the submodules under it recursively. In the second step,

Algorithm 2 Produce procedure of a module

```
1: procedure PRODUCE(void)
2:     for each submodules i in this module do
3:         submodule[i].produce()
4:     end for
5:     LocalCompute(1)
6:     updateInAvail
7:     updateDataOut
8:     Return
9: end procedure
```

all the submodules under the top module call the produce procedures recursively. When calling the produce procedure as shown in Algorithm 2, one module invokes the produce procedures of all the submodules under it recursively. Then the computation logic of this module advances by one cycle. Finally, *InAvail* and *DataOut* are updated based on the computation results.

Algorithm 3 Procedure of advancing N clock cycles

```
1: procedure ADVANCE(N)
2:     CycleCounter = 0
3:     while CycleCounter < N do
4:         for each submodules i in top-level module do
5:             submodule[i].consume()
6:         end for
7:         for each submodules i in top-level module do
8:             submodule[i].produce()
9:         end for
10:        CycleCounter = CycleCounter + 1
11:    end while
12:    Return
13: end procedure
```

The consumer-producer model not only guarantees the correctness of the data dependency relationships of the RTL model, but also gives the cycle-accurate simulator good extensibility and maintainability. In the future, the VC-based router might be upgraded with better algorithms, policies, and perhaps even new architectures. As

Algorithm 4 Initialization procedure of of a module

```
1: procedure INIT(void)
2:     Ptr * DataIn = &(UpstreamModule.DataOut)
3:     Ptr * OutAvail = &(DownstreamModule.InAvail)
4:     for each submodules i in this module do
5:         submodule[i].init()
6:     end for
7:     Return
8: end procedure
```

long as the new module is implemented using the interface shown in Figure 5·1, the cycle-accurate simulator can be extended with little additional effort. One extra benefit of this consumer-producer model is that the simulation can easily be parallelized into a multi-threaded version. The parallel version of Algorithm 3 is displayed in Algorithm 5.

Algorithm 5 Procedure of advancing N clock cycles (parallel)

```
1: procedure AdvanceParallel(N)
2:     CycleCounter = 0
3:     while CycleCounter < N do
4:         #pragma parallel for
5:         for each submodules i in top-level module do
6:             submodule[i].consume()
7:         end for
8:         all threads join here
9:         #pragma parallel for
10:        for each submodules i in top-level module do
11:            submodule[i].produce()
12:        end for
13:        CycleCounter = CycleCounter + 1
14:    end while
15:    Return
16: end procedure
```

With this cycle-accurate simulator, we can exhaustively simulate all the possible router configurations and find the one with the best performance.

5.1.4 Evaluation

In this subsection, we did two sets of evaluations. The first compares the performance of the best configuration with the average performance of all the different configurations. The second generates a histogram displaying the times of all of the optimal configurations. In this subsection, we first introduce the experimental setup. And then we present and analyze the results of the experiments.

Experimental Setup

We conduct experiments on seven different workloads, which are listed in Table 5.1. All of these workloads appear frequently in applications. The NN and 3H-NN patterns are often used in stencil computations. CUBE-NN is used in the MD short-range force calculation. BC is important in many DSP applications. TRAN is often used in image processing. All-to-all appears in the FFT. We test two different representative network sizes: $4 \times 4 \times 4$ and $8 \times 8 \times 8$.

We evaluate six standard performance metrics, which are listed in Table 5.2. The first five metrics are straightforward. The is explained as follows. When running the experiments, the number of VCs on each input port is fixed to 9, which is the upper limit of the VC-based router, because we would like the area consumption of the router is lower than a quarter of the entire FPGA chip. The last metric in Table 5.2 measures the maximum number of non-idle VCs during the entire run. If this number is lower than 9, this means we can reduce the number of VCs without impairing the performance. In our design, the number of VCs affects not only the area consumption of the input units but also the area of the switch. The last column of Table 4.2 shows the relationship between the number of VCs and the area consumption of the router. We can find the area consumption is about linear to the number of VCs, which means the by evaluating this metric, we could significantly save the area consumption of the

router without any degradation in its performance.

Table 5.1: Seven different workloads to evaluate the VC-based router performance. For tornado pattern, the XSIZE is the number of nodes on the X dimension.

Workload Name	abbr.	Pattern
Nearest Neighbor	NN	(x,y,z)→(x+1,y,z),(x-1,y,z)...(x,y,z-1)
3-hop Diagonal Nearest Neighbor	3H-NN	(x,y,z)→(x+1,y+1,z+1),(x+1,y+1,z-1),(x+1,y-1,z+1)...(x-1,y-1,z-1)
Cube Nearest Neighbor	CUBE-NN	(x,y,z)→nodes⊂[x-1,x+1][y-1,y+1][z-1,z+1]
Bit Complement	BC	(x,y,z)→(bitcomplement(x,y,z))
Transpose	TRAN	(x,y,z)→(z,x,y)
Tornado	TORN	(x,y,z)→(x+XSIZE/2-1,y,z)
All-to-all	ATA	(x,y,z)→all the other nodes

Table 5.2: Six different performance metrics

Metric Name	Description
Total Latency	The difference between the time-stamp of the last flit is ejected and the first flit is injected
Average Latency	The average latency of all the packets
Worst-case Latency	The latency of the worst-case packet
Average Sending Throughput	The average injected number of flits per node per cycle
Average Receiving Throughput	The average ejecting number of flits per node per cycle
Maximum number of non-idle VCs	The maximum number of VCs is not idle during the entire run

Experimental Results

The first set of experiments is compares the performance of the optimal configuration with the average performance of all configurations. The results for network 4^3 are displayed in Figures 5·2, 5·3, 5·4, 5·5, and 5·6.

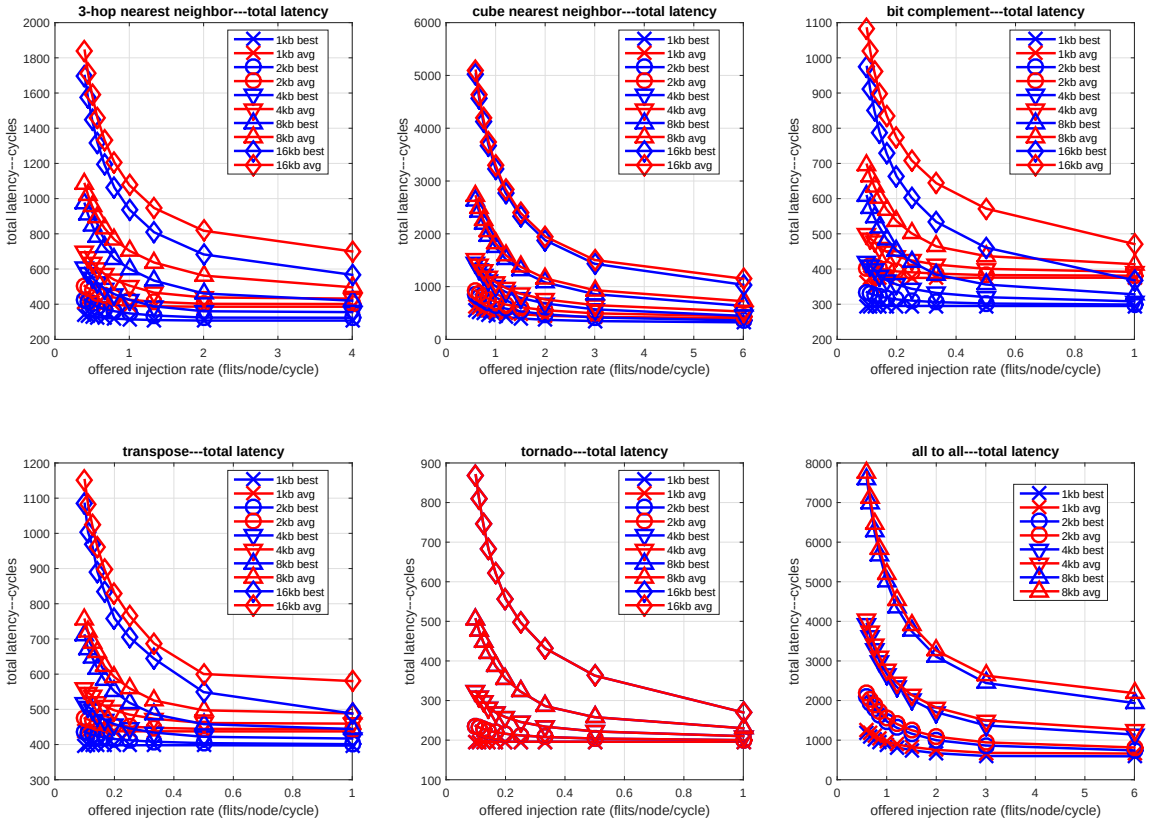


Figure 5·2: The comparison between the batch latency of the optimal configuration with the average batch latencies of all configurations for different workloads on a 4^3 torus. From left to right in first row: 3H-NN, CUBE-NN, bit complement; from left to right in second row: transpose, tornado, all-to-all

We do not plot any results for the NN pattern it can trivially achieve 100% bandwidth utilization on our router. So no matter which configuration we used, the results will be the same. In the other six patterns, the results of the tornado are also not interesting. The average and optimal performance are almost the same because the pattern is again too simple. There are no turns so different routing algorithms and policies do not alter the performance. So we did not plot the tornado pattern results for 8^3. The 8^3 results are shown in Figure 5·7, Figure 5·8, Figure 5·9, Figure5·10, and Figure 5·11. We notice that in the results of 8^3, there is no all-to-all pattern, which is because the all-to-all pattern on 8^3 still takes too much time to

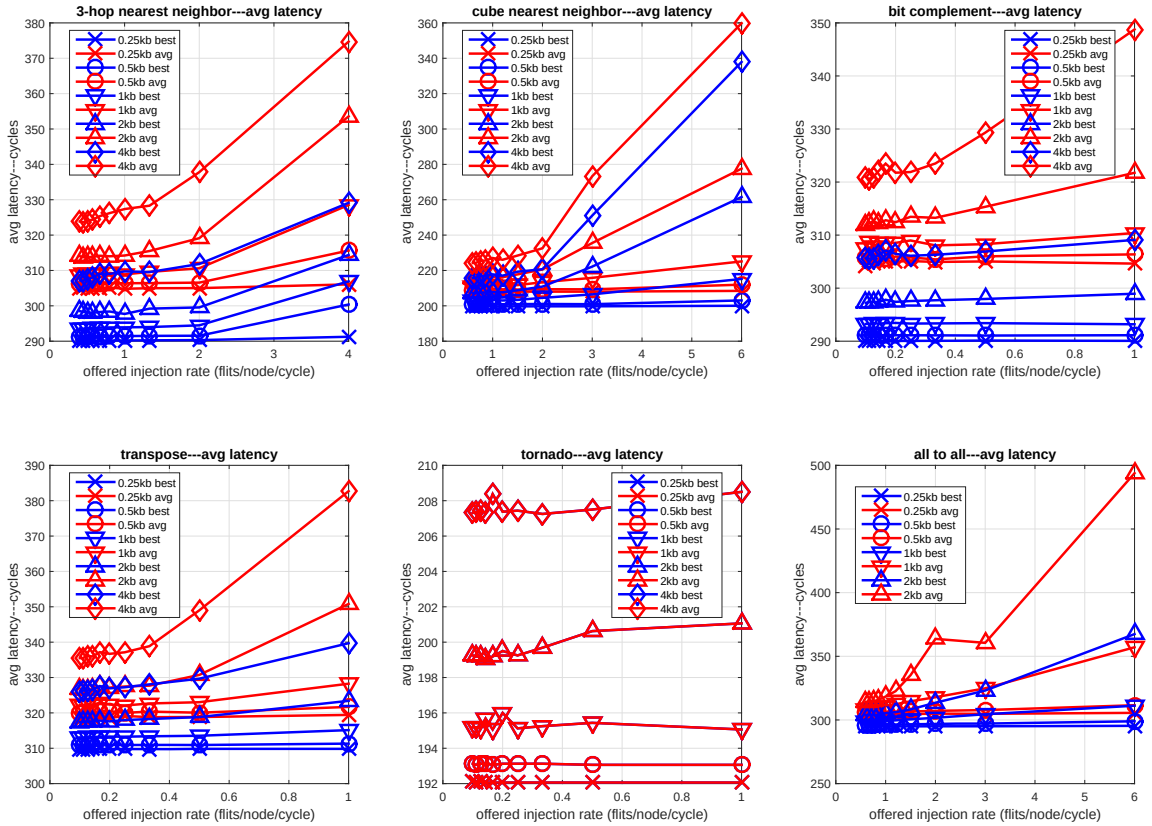


Figure 5·3: The comparison between the average latency of the optimal configuration with the average performance of all configurations for different workloads on a 4^3 torus. From left to right in first row: 3H-NN, CUBE-NN, bit complement; from left to right in second row: transpose, tornado, all-to-all

simulate (more than 500 destinations to send for each source). In all these figures, the horizontal axis is the offered injection rate. The offered injection rate means the ideal injection rate. It might be higher than the actual injection rate because the injection port might have backpressured from the bypass traffic. The red lines represent the average performance for all configurations. The blue lines represent the performance of the optimal configuration. The lines with different marks represent the various packet sizes.

By looking at Figures 5·2 and 5·7, we find that the batch latency is drastically reduced by increasing the injection rate, which is because by increasing the injection

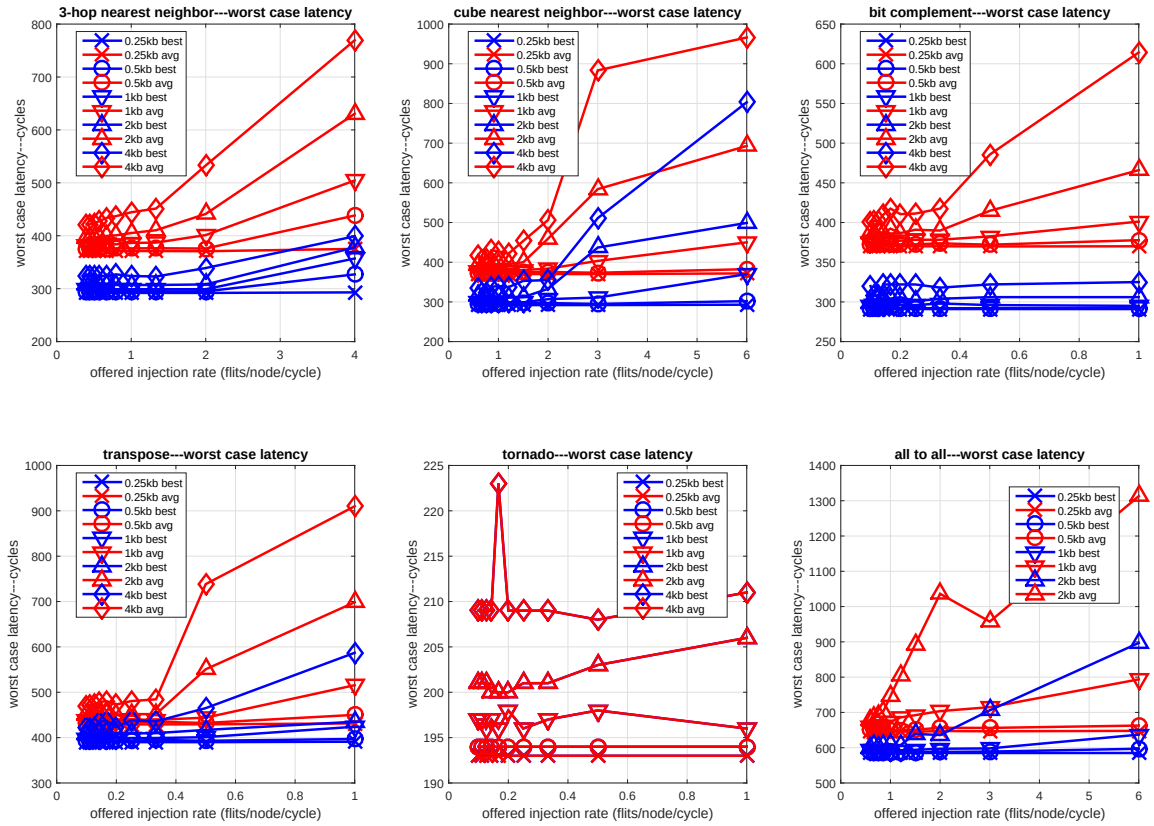


Figure 5·4: The comparison between the worst-case latency of the optimal configuration with the average performance of all configurations for different workloads on a 4^3 torus. From left to right in first row: 3H-NN, CUBE-NN, bit complement; from left to right in second row: transpose, tornado, all-to-all

ratio, the time to inject data into the network is drastically reduced. The injection time dominat the batch latency. In these two figures, we also find that the improvements for the optimal configuration is more obvious for 3-hop diagonal nearest neighbor, bit complement, and transpose. In the tornado, there are almost no improvements, which is because the tornado is too simple as described above. In CUBE-NN and all-to-all, the improvements are also not significant, which is because they are both heavy workloads. The entire network is too congested so that there is little room for better algorithms and policies to have a benefit.

By observing Figures 5·3, 5·4, 5·8, and 5·9, we can find that average and worst-

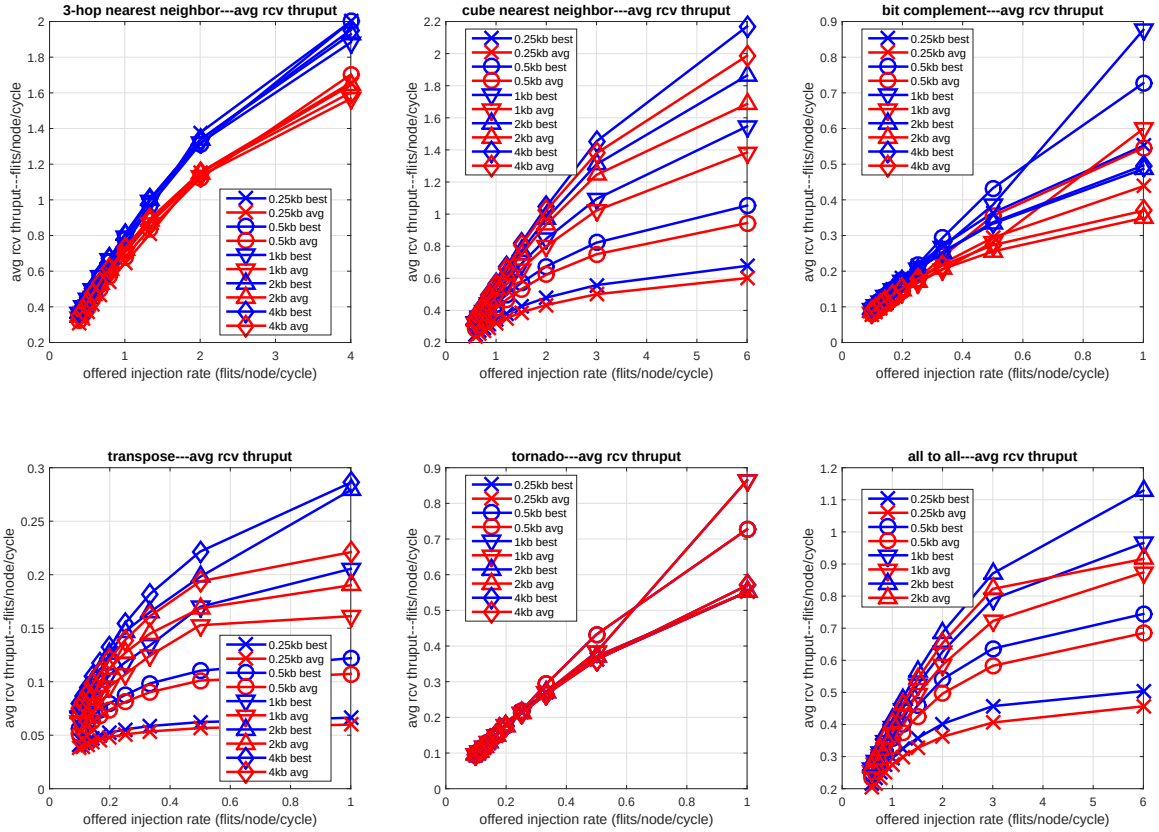


Figure 5·5: The comparison between the average receiving throughput of the optimal configuration with the average performance of all configurations for different workloads on a 4^3 torus. From left to right in first row: 3H-NN, CUBE-NN, bit complement; from left to right in second row: transpose, tornado, all-to-all

case latency increases slowly until the injection rate reaches a saturation point, which is a well-known and expected behavior. All the latency increases with the packet size, which is expected as well. Similarly as with the batch latency, we can also find that the improvement of the optimal configuration is more noticeable in the 3-hop diagonal nearest neighbor, bit complement, and transpose. This is because these three patterns are neither too lightly nor too heavily loaded. The better algorithms and policies have sufficient room to find improvements. We also find that the improvements on worst-case latency is much higher than the improvement on average latency: the worst-case latencies are generally have much greater variance than the average latencies.

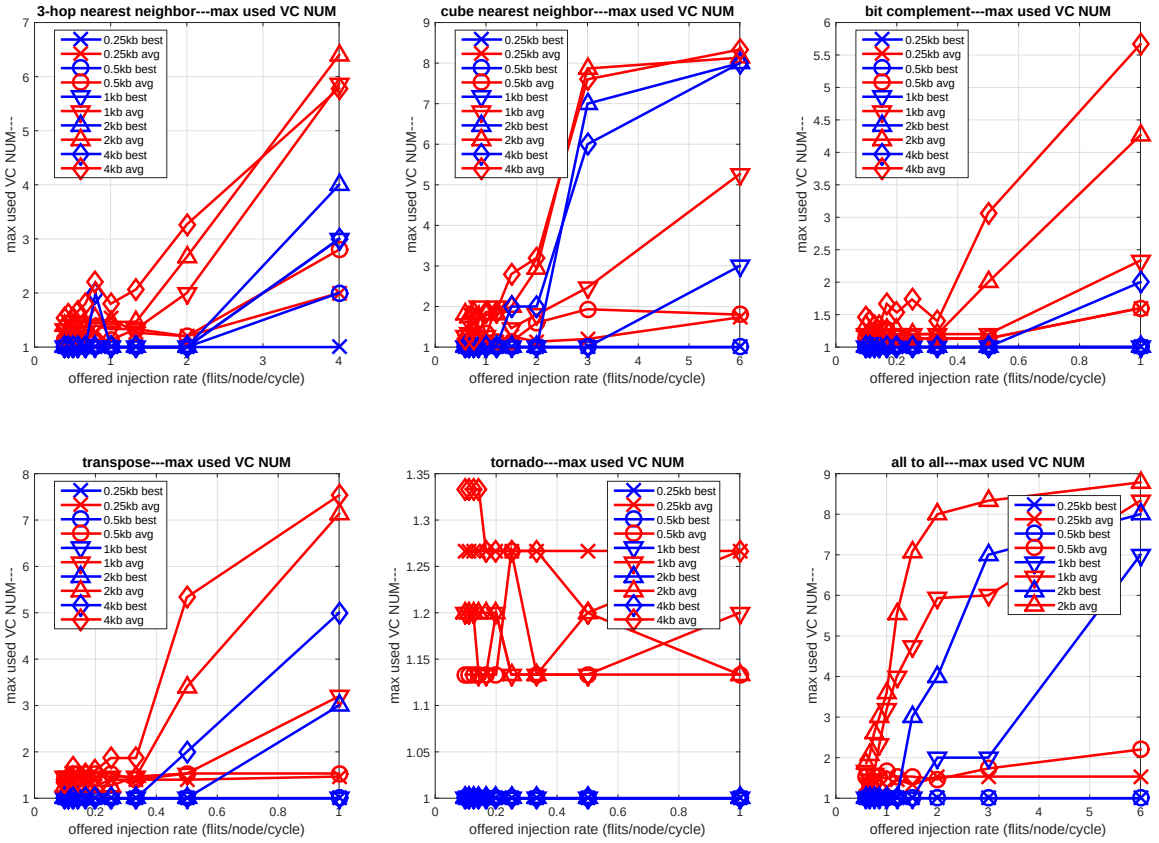


Figure 5·6: The comparison between the maximum number of non-idle VCs of the optimal configuration with the average performance of all configurations for different workloads on a 4^3 torus. From left to right in first row: 3H-NN, CUBE-NN, bit complement; from left to right in second row: transpose, tornado, all-to-all

In Figures 5·3 and 5·8 we find that the increasing slope of the throughput reduces when the offered injection rate increases. In some cases, the curves even become a plateau, which is because when the injection rate reaches a certain point, it saturates the bandwidth of the entire network. Again the tornado pattern is an exception. Because of its simplicity, the throughput increases almost linearly with the injection rate. One interesting phenomenon is that in most cases, the throughput increases with the packet size. This is because, in our network, the bypass traffic always has higher throughput than the injection traffic, which means the user's logic only allows

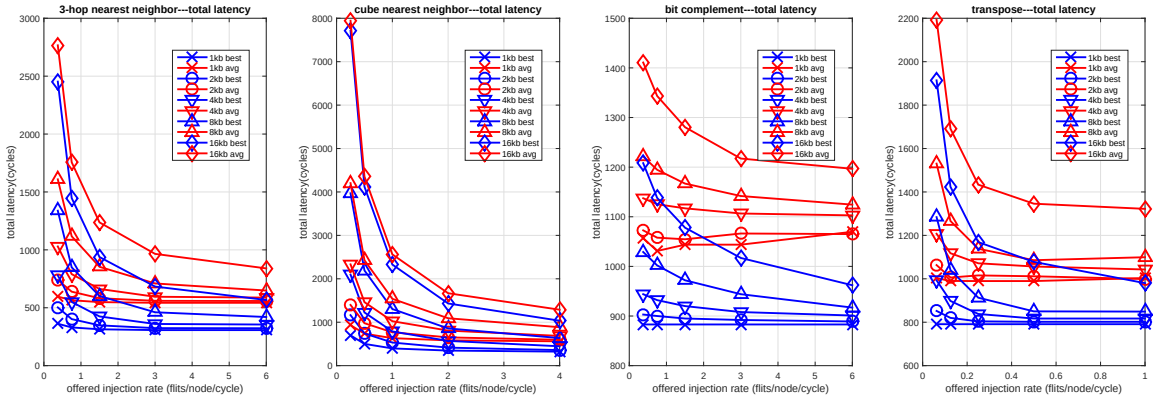


Figure 5·7: The comparison between the batch latency of the optimal configuration with the average batch latencies of all configurations for different workloads on a 8^3 torus. From left to right: 3H-NN, CUBE-NN, bit complement, transpose

injection of the packet when the output link has no bypass the traffic. Whenever the user logic starts to inject, it is not allowed to be preempted by bypass traffic until the tail flit is injected. Therefore, the large packet size provides a higher probability for injection logic to occupy output link bandwidth.

From Figures 5·6 and 5·11, we find that the maximum number of non-idle VCs increases with an increase in the injection rate, which matches our expectation. More VCs are occupied when the network gets more congested. From these two figures, we also observe that for most cases, 9 VCs are more than enough. From Table 4.2, we observe that the router size is almost linear in the number of VCs. The framework not only finds the configuration with the best performance, but can also compact the router size by using the smallest sufficient number of VCs.

It is evident that, in most cases, the performance of the application-aware configuration has significant improvement over the average performance of all configurations. Tables 5.3 and 5.4 quantify the improvements for 4^3 and 8^3 networks, respectively. The performance improvements are calculated by $|opt - avg|/avg$. The improvements of the batch latency, average latency, worst-case latency, and throughput are all di-

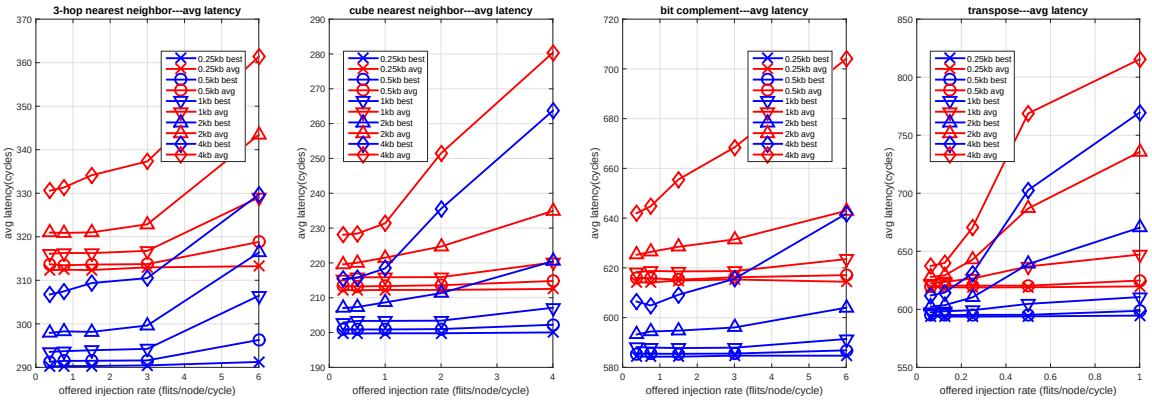


Figure 5·8: The comparison between the average latency of the optimal configuration with the average performance of all configurations for different workloads on a 8^3 torus. From left to right: 3H-NN, CUBE-NN, bit complement, transpose

rectly calculated by the geometric mean of all of the data points shown in Figure 5·2 to 5·5 and 5·7 to 5·10. The improvements in area consumption are calculated by converting the data points shown in Figures 5·6 and 5·11 into area consumption based on the last column of Table 4.2. The results show that in the 4^3 torus, the optimal configurations found by our proposed framework improve batch latency by 9%, average latency by 3%, worst-case latency by 15%, throughput by 6%, and area consumption of the router by 15%. The results also show that for the 8^3 torus, the found optimal configurations by our framework improve batch latency by 23%, average latency by 6%, worst-case latency by 34%, throughput by 17%, and area consumption of the router by 30%.

The five routing algorithms and three switch arbitration policies form fifteen different configurations. We count the times each configuration is optimal and plot them in Figures 5·12, 5·13, 5·14, 5·15, and 5·16. From Figures 5·14 and 5·15, we find that ROMM is obviously better than other routing algorithms for transpose and all-to-all workloads. From Figure 5·16, we find that DOR is apparently the best routing algorithm for bit-complement workloads. Figure 5·12 shows that for CUBE-NN work-

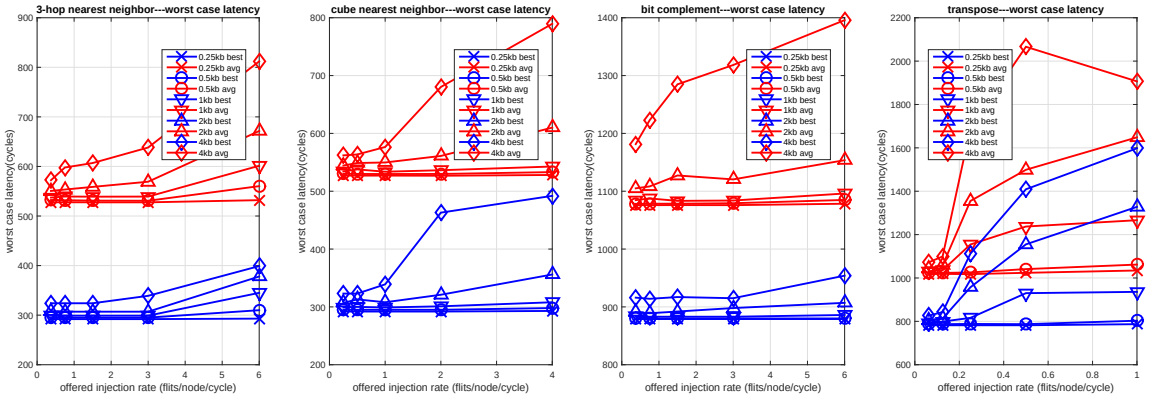


Figure 5·9: The comparison between the worst-case latency of the optimal configuration with the average performance of all configurations for different workloads on a 8^3 torus. From left to right: 3H-NN, CUBE-NN, bit complement, transpose

loads, with respect to the average latency, the adaptive routing algorithm with oldest-first policy outperforms other configurations substantially. For CUBE-NN workloads, the DOR routing with farthest-first achieves the best area consumption most of the time. The same situation appears in the 3H-NN workloads. For other metrics and 3H-NN workloads, there are no obvious consistent best routing algorithms.

We conduct another experiment to find the configurations that have the best geometric mean performance for all workloads. We refer to these configurations *globally optimal*. We also find the configurations with the best geometric mean performance for each individual metric. We refer to these configurations *application-aware optimal*. We compare the global and application-aware optimal configuration in Table 5.5. The results in this table are calculated by $|app - global|/global$. We observe that for different metrics, the optimal configurations differ. The observation from Table 5.5 is that the application-aware optimal configurations are better than the global-optimal configurations for all the metrics, which means that application-awareness is essential to achieving better communication performance on FPGA clusters. A golden global router configuration is not enough. The application-aware router configurations gen-

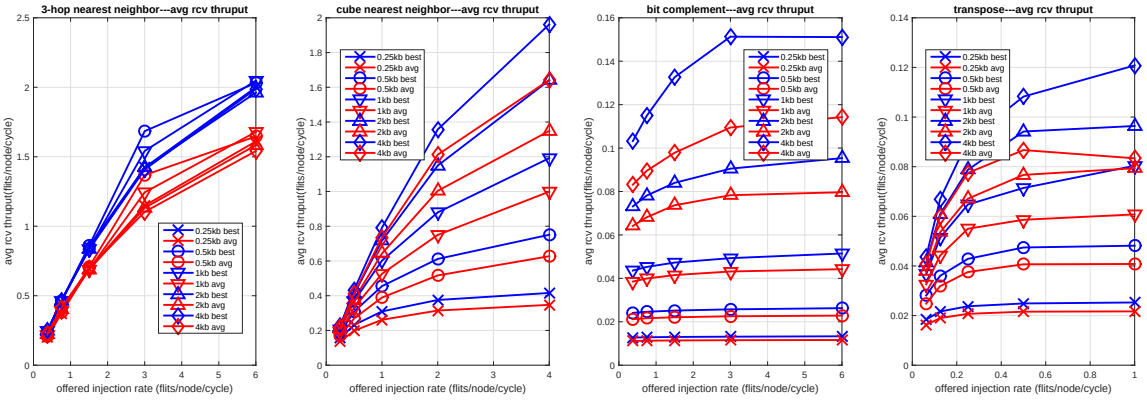


Figure 5·10: The comparison between the average receiving throughput of the optimal configuration with the average performance of all configurations for different workloads on a 8^3 torus. From left to right: 3H-NN, CUBE-NN, bit complement, transpose

erated by our framework can substantially improve the communication performance on FPGA clusters.

5.1.5 Discussion

In Chapter 4, we propose a wormhole VC-based router on Novo-G#. In this section, we first describe the routing algorithms and switch arbitration policies implemented on the router. We describe a cycle-accurate simulator we use to simulate the router and the entire Novo-G# network based on the router. With the good extensibility and maintainability of this simulator, we can propose a framework to search for the optimal application-aware router configuration. Our results demonstrate that in 4^3 torus, the optimal configurations found by our proposed framework improve batch latency by 9%, average latency by 3%, worst-case latency by 15%, throughput by 6%, and the area consumption of the router by 15%. The results also show that in 8^3 torus, the optimal configurations found by our proposed framework improve batch latency by 23%, average latency by 6%, worst-case latency by 34%, throughput by 17%, and the area consumption of the router by 30%. Our results also show that by

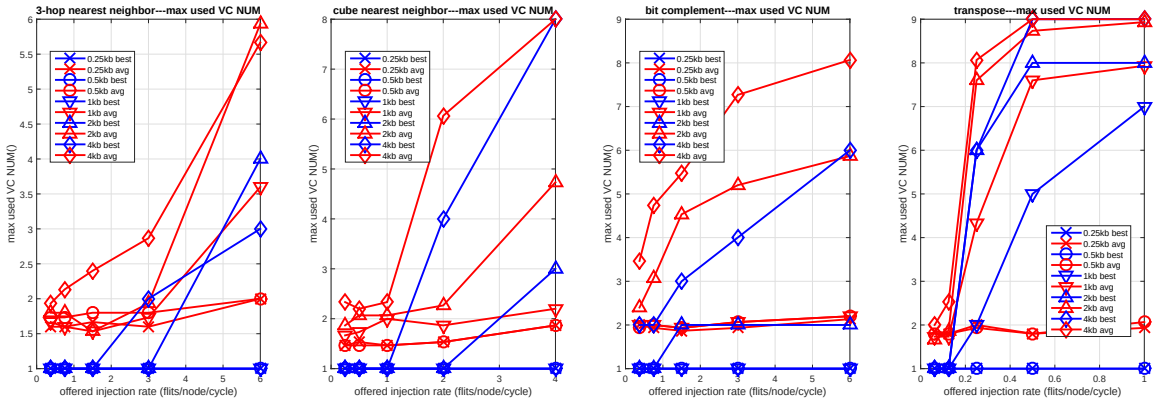


Figure 5·11: The comparison between the maximum number of non-idle VCs of the optimal configuration with the average performance of all configurations for different workloads on a 8^3 torus. From left to right: 3H-NN, CUBE-NN, bit complement, transpose

applying application-awareness provided by our framework, we reduce batch latency by 6%, average latency by 2%, worst-case latency by 21%, router area size by 18%, and improve throughput by 10%.

The current results are just a demo showing the potential of this framework. The framework can easily be extended with new routing algorithms, switch arbitration policies, and even new router microarchitectures.

Currently, the framework uses exhaustive search to find the optimal configuration because of there is only 15 configurations. With larger design spaces, a more efficient search algorithm is needed. Also, the simulator needs to be implemented into a multi-threaded version. In this work, we only propose pseudocode for it, as shown in Algorithm 5.

5.2 Proposed Offline Collective Routing Algorithm in Proposed VOQ Router

We develop a novel offline collective routing algorithm. Together with proposed statically-scheduled collective acceleration router, we form an entirely new communi-

Table 5.3: The improvements of performance of optimal router configuration compared with the average performance of all configurations on a 4^3 torus

Pattern	Total Latency	Avg Latency	Worst Latency	Throughput	Area
3H-NN	16%	5%	25%	12%	13%
CUBE-NN	9%	4%	22%	5%	17%
BitComple	18%	5%	23%	2%	10%
Transpose	8%	3%	12%	11%	15%
Tornado	0%	0%	0%	0%	0%
All to all	5%	4%	11%	5%	26%
Average	9%	3%	15%	6%	13%

Table 5.4: The improvements of performance of optimal router configuration compared with the average performance of all configurations on a 8^3 torus

Pattern	Total Latency	Avg Latency	Worst Latency	Throughput	Area
3H-NN	34%	8%	46%	18%	28%
CUBE-NN	24%	6%	43%	14%	25%
BitComple	16%	6%	25%	19%	39%
Transpose	20%	5%	25%	17%	27%
Average	23%	6%	34%	17%	30%

cation infrastructure to support offline collective routing. Our contributions include following:

- We propose and implement an entirely new communication infrastructure to support offline routing, which includes a new router design that supports both online and offline routing (simultaneously) and a new offline routing algorithm for collectives.

- We find that, compared with a state-of-the-art online router architecture, our Proposed VOQ router architecture saves FPGA logic resources while only requiring slightly more on-chip memory.

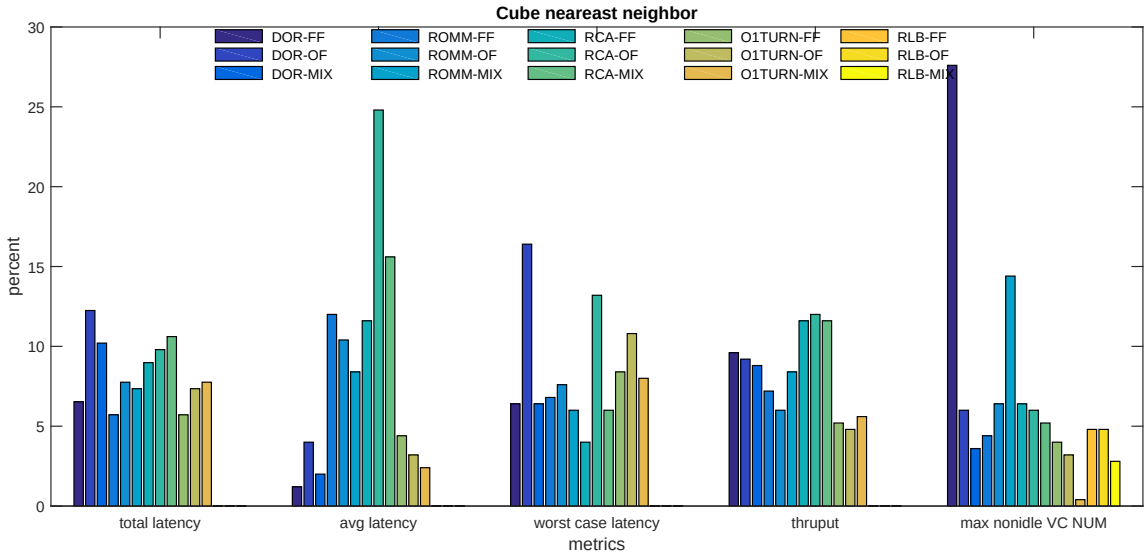


Figure 5·12: The histogram that plots the distribution of the fifteen router configurations being optimal for cube nearest neighbor workload

Table 5.5: The comparison between global optimal configuration and application-aware optimal configuration. The percentage is calculated by $geomean(|app - global|/global)$.

	Total Latency	Avg Latency	Worst Latency	Throughput	Area
Avg Benefit	6%	2%	21%	10%	18%

- We discover that a priori knowledge of communication can also be used to reduce buffer sizes and enable higher bandwidth utilization.

- Our experimental results demonstrate that, when compared with a state-of-the-art online routing algorithm, our new offline routing algorithm reduces the latency of multicast by 15% and of reduction by 4%.

In the rest of this section, we first propose and describe a novel offline collective routing (OCR) algorithm. Then, from multiple aspects, we evaluate OCR by comparing it with a representative online collective algorithm called RPM. Finally, we present results.

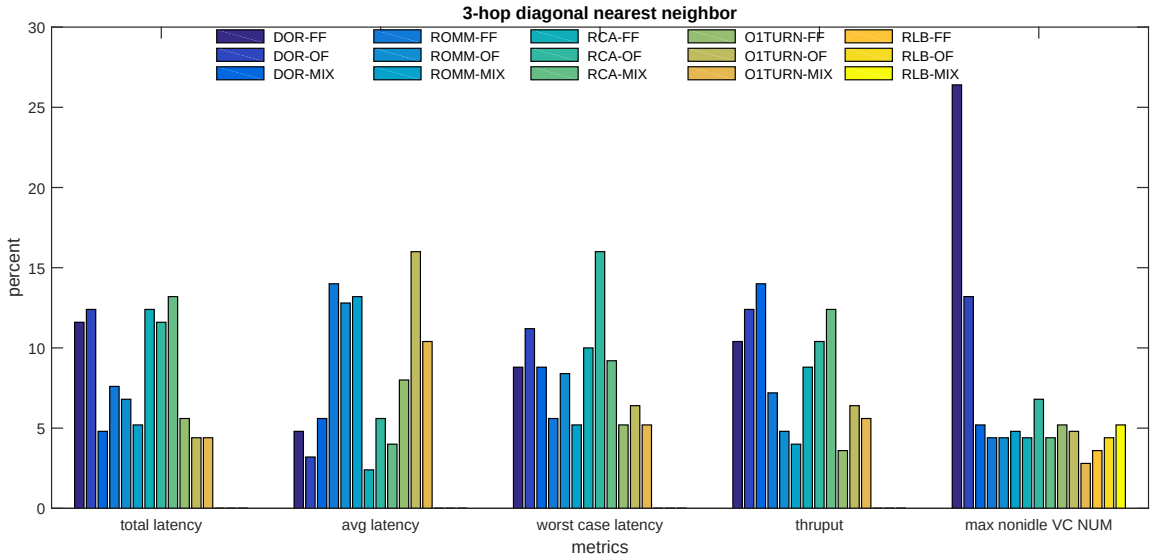


Figure 5·13: The histogram that plots the distribution of the fifteen router configurations being optimal for 3-hop diagonal nearest neighbor workload

5.2.1 Algorithm Details

Collective operations can be implemented either with unicast or multicast (see Figure 5·17). Figures 5·17 (a) and (c) show multicast and reduction using unicasts. All the packets are unicast but share the same source (or destination). Figure 5·17 (b) and (d) show multicast and reduction based on a tree topology; the communication burden is obviously drastically reduced.

Tree-based collective routing algorithms have been much studied recently (Jerger et al., 2008; Abad et al., 2009; Wang et al., 2009; Krishna et al., 2011). Recursive Partition Multicast (RPM) in (Wang et al., 2009) appears to be the leading such algorithm. Developed for Networks-on-Chip, RPM only works for 2D meshes. Here, we extend RPM to deal with both multicast and reduction on a 3D torus; we also enhance the original algorithm for 2D.

RPM provides a solution to generate the multicast pattern recursively on a 2D-mesh network. Their goal is to build a multicast tree that maximally reuses network

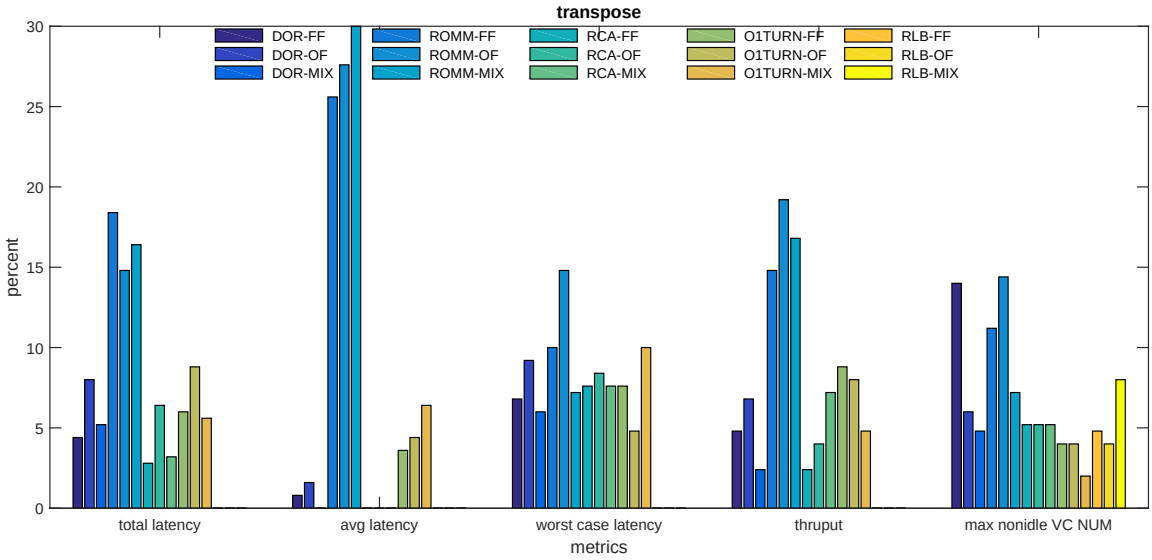


Figure 5·14: The histogram that plots the distribution of the fifteen router configurations being optimal for transpose workload

links. We found, however, that there are two places in the algorithm that can be improved (see Figure 5·18).

In (a) and (b), $dst0$ is northeast of src, and $dst1$ is southeast. The best multicast routing decision is first to send the packet east and then let this node distribute packets to $dst0$ and $dst1$ (as in (b)). RPM, however, sets North always to have the highest priority. The packet first goes north and then east to $dst0$; this does not reuse the east link. In Figure (c) and (d), the two routing decisions have the same reusability of links. However, the north and south links are less congested than the west and east links, indicating that the routing decision made in (d) is preferred to the one in (c). RPM makes these suboptimal decisions because it does not account for link congestion and because of the policy giving priority to the north link.

Our new offline collective algorithm addresses these two drawbacks in RPM, and also extends it to *tori* and 3D. We call our algorithm offline collective routing (OCR). Pseudo code for OCR is shown in Algorithm 6. The first step is to determine the dimensionality of the topology. If 1D, then multicast routing is immediate. The

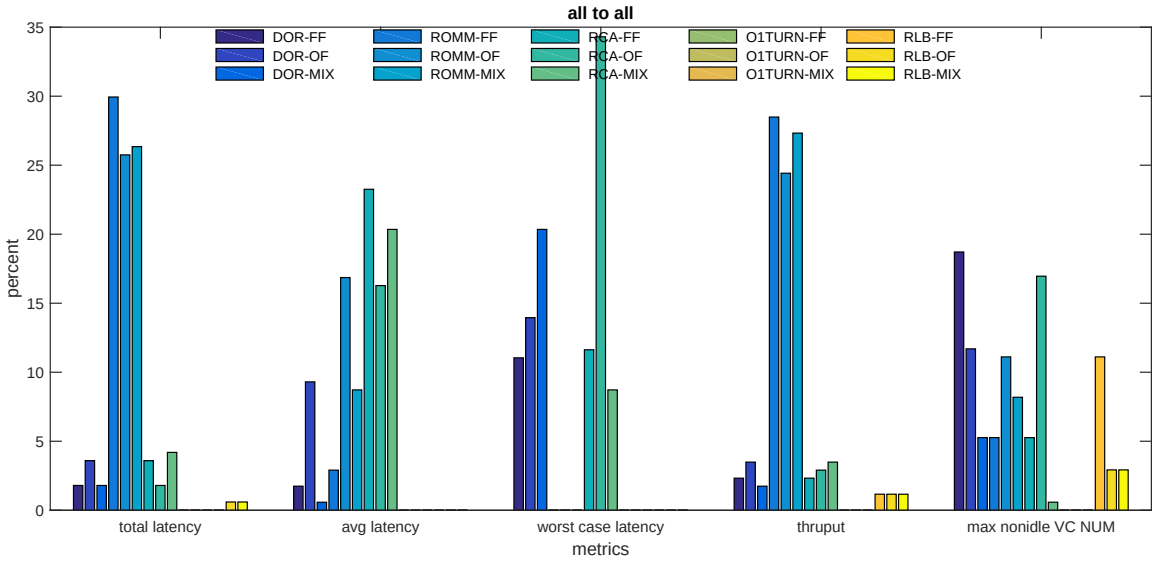


Figure 5·15: The histogram that plots the distribution of the fifteen router configurations being optimal for all-to-all workload

algorithm for the 2D is described below. If 3D, then the next step is to find an optimal partition. Three options are corresponding to the number of dimensions. If the space is a 3D torus, we can always partition it into three parts because every node can be viewed as the center of the network.

As illustrated in Figure 5·19, we partition the space along three dimensions. We then count the number of outbound links exiting the source to all the destinations for each kind of partition. In this example, the partition along the YZ plane requires only one outbound link, while the partition along the XZ and XY planes need two and three links respectively. It is apparent that the partition along YZ plane is the best partition method in this example. If there is more than one partition that has the minimal number of outbound links, we then select the one that results in a smaller variance in the loads on the six outbound links. If the partitions are still tied, we use a global round robin pointer. After we find the best partition, we partition the entire space into three parts: up space, middle plane and down space and distribute the destinations into the three subspaces depending on their coordinates. For the up

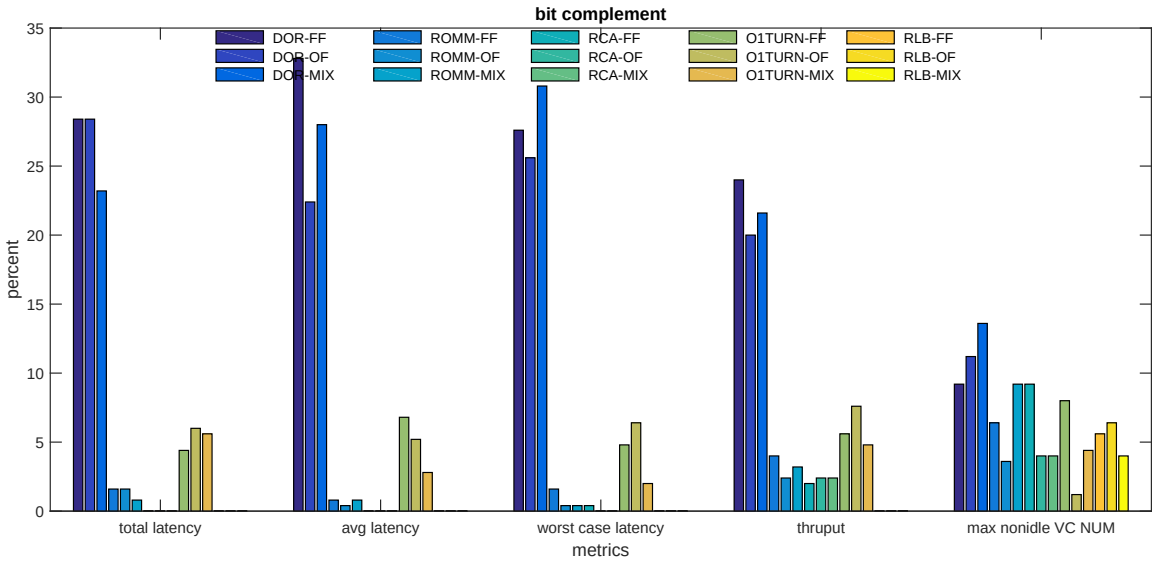


Figure 5·16: The histogram that plots the distribution of the fifteen router configurations being optimal for bit complement workload

and down space, we call the 3D OCR algorithm recursively. For the middle plane, we call the modified 2D OCR algorithm.

The 2D OCR algorithm is similar to RPM. As shown in Figure 5·20, we also partition the 2D space into eight regions depending on the source location. If the space is 2D, we can always find the eight regions since the source node could always be the center of the torus. We call regions 0,2,4,6 corner regions and regions 1,3,5,7 side regions. In the 2D plane, one source has at most four fan-outs, which means we can have at most four partitions among the eight regions. One corner region must merge with either one of its two adjacent side regions. The first step is to count the number of nodes in all eight regions. The next step is to determine whether to enable the links in the north, south, west and east directions and to determine which side region each corner region should merge with.

Algorithm 7 shows that the algorithm for the north link, south link, and region 0. The merge direction of each corner region depends first of all on whether there are nodes in region 1, 2, 6 and 8, and second load on the north and east links. In

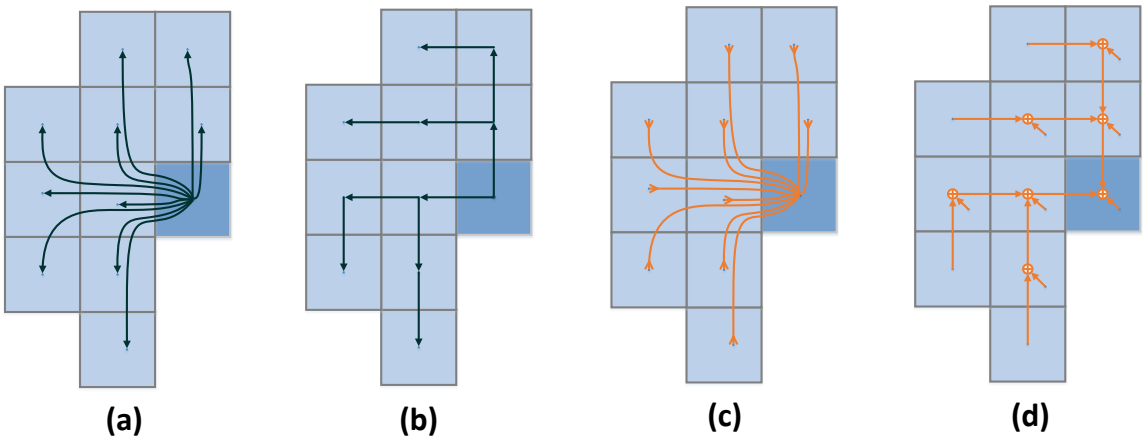


Figure 5·17: (a) unicast-based multicast (b) tree-based multicast (c) unicast-based reduction (d) tree-based reduction

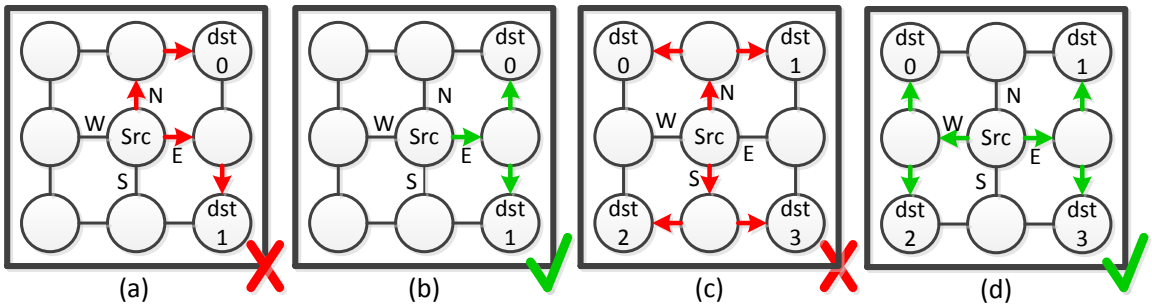


Figure 5·18: (a) and (c) show routing decisions made by RPM, (b) and (d) show those likely to result in improved performance. In (c) and (d), the north and south links are more congested than the west and east links.

the next step, the plane and the destination list are partitioned into up to four parts. The 2D OCR algorithm is called recursively for the four parts until they become 1D spaces.

5.2.2 Algorithm Evaluation

In this section, we compare our OCR-based offline solution with an online routing solution based on RPM.

Algorithm 6 The pseudo Code of Offline Collective Routing algorithm in a 3D torus space ($OCR3D$)

procedure $OCR3D(Src,\ DstList,\ Space)$
  if $dimension(Space) == 1$ **then** ▷ if space is 1D
    $multicast(Src, DstList, Space)$ ▷ multicast directly
  else if $dimension(Space) == 2$ **then** ▷ if space is 2D
    $OCR_2D(Src, DstList, Space)$ ▷ call OCR_2D
  else if (**then**$dimension(Space) == 3$) ▷ if space is 3D
    $BestPartition = EvalPartition(Src, DstList, Space)$ ▷ select best among partition along xy, yz, and xz plane
    $[SrcA, SrcB] = PartitionSrc(Src, BestPartition)$
    $[DstListA, DstListPlane, DstListB] = PartDst(DstList, BestPartition)$
    $[SpaceA, SpacePlane, SpaceB] = PartitionSpace(Space, BestPartition)$
    $OCR3D(SrcA, DstListA, SpaceA)$
    $OCR2D(Src, DstListPlane, SpacePlane)$
    $OCR3D(SrcB, DstListB, SpaceB)$
  end if
  Return
end procedure

Experimental Setup

Our target system is theNovo-G# as described in Chapter 3. Each board contains an Altera Stratix V 5GSMD8 chip and supports six links that use the FPGA's Multigigabit Transceivers (MGT). Each MGT link has a bandwidth of 40Gbps and latency of 175 ns. All the boards run at 156.25 MHz, the same as the MGTs. We have two router designs, including one that supports just the online RPM routing logic and another with the OCR-based design described previously. We are currently targeting 4^3 and 8^3 clusters. Designs are coded in Verilog and synthesized with Quartus II 14.1.

Currently, all paths in the routers are 256 bits to match the parallel I/O of the MGTs (256:1 ratio of frequencies). Modest tuning will allow us to double the operating frequency of the routers, and so halve the path widths and therefore the resources used. Smaller designs are also possible but result in the MGTs being used sub-optimally.

Algorithm 7 Part of the Pseudo Code of $OCR2D$ algorithm for $region0$, north link and east link

```
procedure OCR2D(Src, DstList, Plane)
    if SearchNode(DstList, region1) then                    ▷ if dst is in region0
        enable(LinkNorth)
    end if
    if SearchNode(DstList, region7) then                    ▷ if dst is in region7
        enable(LinkEast)
    end if
    if SearchNode(DstList, region0) then                    ▷ if dst is in region0
        if IsEnable(LinkNorth)&&IsDisable(LinkEast) then
            merge(region0, region1)
        else if IsDisable(LinkNorth)&&IsEnable(LinkEast) then
            merge(region0, region7)
        else if IsEnable(LinkNorth)&&IsEnable(LinkEast) then
            if LinkNorth.load > LinkEast.load then
                merge(region0, region7)
            else
                merge(region0, region1)
            end if
        else                                    ▷ both north and east link are disabled
            if region2.has(DstList)&&!region6.has(DstList) then
                enable(LinkNorth)
                merge(region0, region1)
            else if !region2.has(DstList)&&region6.has(DstList) then
                enable(LinkEast)
                merge(region0, region7)
            else
                if LinkNorth.load > LinkEast.load then
                    merge(region0, region7)
                else
                    merge(region0, region1)
                end if
            end if
        end if
    end if
end procedure
```

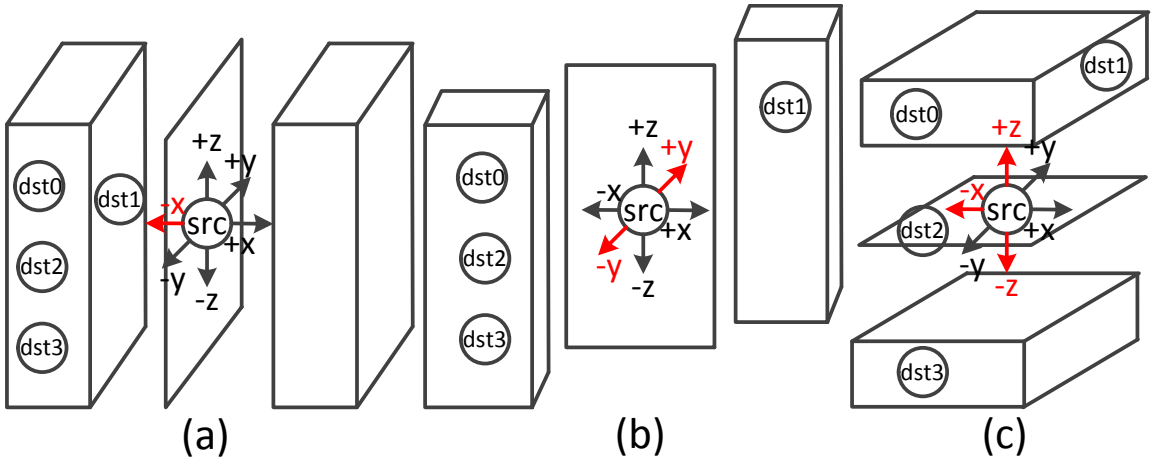


Figure 5·19: The partition evaluation of OCR algorithm. (a) partition along YZ plane, (b) partition along XZ plane, (c) partition along XY plane

We use one-sided communication over MGT links rather than handshake communication. One reason is that the in-flight latency is already much longer than the latency in the router (175ns versus about 45ns). To avoid packet drop, we must ensure that the buffers are sufficiently big. For online routing, the input buffer must always be bigger than the worst-case requirement. For offline routing, however, we can select the buffer size that satisfies the requirements for a specific application. The buffer size is also reduced because the routing algorithm balances the link load.

Designs have been tested and validated on a four node subsystem; performance results below are from ModelSim simulations. We have run experiments using three synthetic communication pattern: random, bit-rotation, and nearest neighbor. More details are given below.

Hardware Cost

Basic resource utilization is shown in Table 5.6. We find that the offline router can save 5% of chip area by eliminating the routing computation logic. We measure the table sizes required by OCR algorithm for three typical collective patterns (see Table

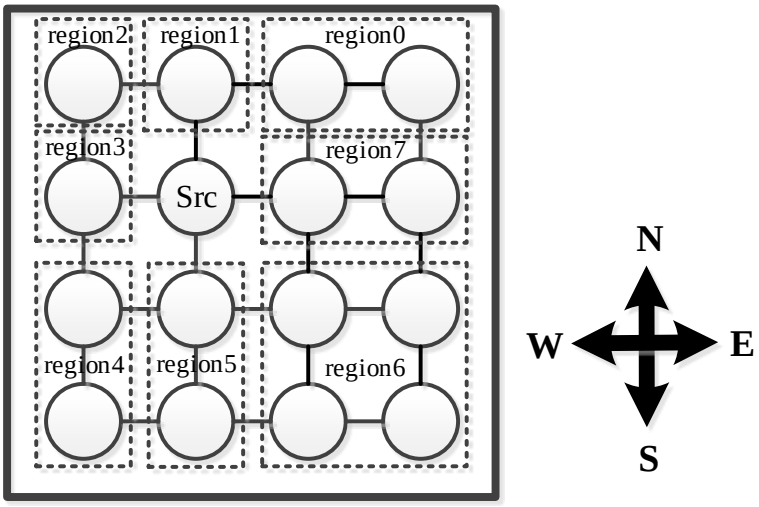


Figure 5·20: 8 regions on a 2D plane, Region 0, 2, 4 and 6 are called corner regions. Region 1, 3, 5 and 7 are called side regions.

5.7. For all three cases, the routing tables consume at most 1.17% of the total on-chip memory. We observe that reduction requires much larger tables than multicast, which is because that we need separate table entries for the different packets in the reduction to buffer temporary results. One reason why routing table sizes for multicast operations are small is that since the communication pattern is fixed. All the packets with the same paths could share a single routing table entry.

Table 5.8 compares the empirically determined worst-case buffer sizes of online and offline routing for the three patterns. The results of Table III show that offline routing can be expected to save around 20% to 30% input buffer size. Two other factors increase the advantage of the offline design. First, in a production design, the online buffers would need to be somewhat larger to deal with worst case scenarios. And second, while the offline buffer can be sized per application, the online account for the worst case across all applications. An alternative for the online design is to use backpressure, but this substantially increases latency.

Another advantage of offline routing is the packet size. In online broadcast, the packet header has to contain entire destination list, while the offline packet header

only needs to carry a table index in the header. For a network has N nodes, the online routing header needs to have N bits, while the offline routing header needs only $logN$ bits.

Table 5.6: The logic elements utilization of RPM router and OCR router on 4×4×4 torus network

	RPM online router	OCR offline router
ALMs	56177	40895
utilization percent	21%	16%

Table 5.7: The memory consumption of routing tables (including multicast tables and reduction tables) of OCR algorithm on 4×4×4 torus network

Pattern	operation	table size(bits)	percentage
All-to-all	multicast	6968	0.013%
All-to-all	reduction	61.7K	1.17%
Bit Rotation	multicast	1122	0.002%
Bit Rotation	reduction	10K	0.19%
Nearest Neighbor	multicast	2928	0.005%
Nearest Neighbor	reduction	25.5K	0.485%

Latencies

We measure latency with two types of loads, batch and continuous. For batch, each node transmits a fixed number of collective packets; latency is the time from when the first packet is sent until the last packet is received. For continuous, each node generates collective with a particular injection rate; latency is the average packet latency.

Figure 5·21 shows the results for batched experiments (speedup of offline versus online). We apply three typical benchmarks (all-to-all, nearest neighbor, and bit rotation) for two network size: $4 \times 4 \times 4$ and $8 \times 8 \times 8$. For nearest neighbor in

Table 5.8: The requirements of worst-case buffer size (depth) of online routing and offline routing for these three synthetic patterns, injection rate here is 1 packet per node per cycle

	operations	online routing	offline routing
All-to-all	multicast	1532	1132
All-to-all	reduction	367	288
Bit Rotation	multicast	91	57
Bit Rotation	reduction	1	1
Nearest neighbor	multicast	277	157
Nearest neighbor	reduction	14	9

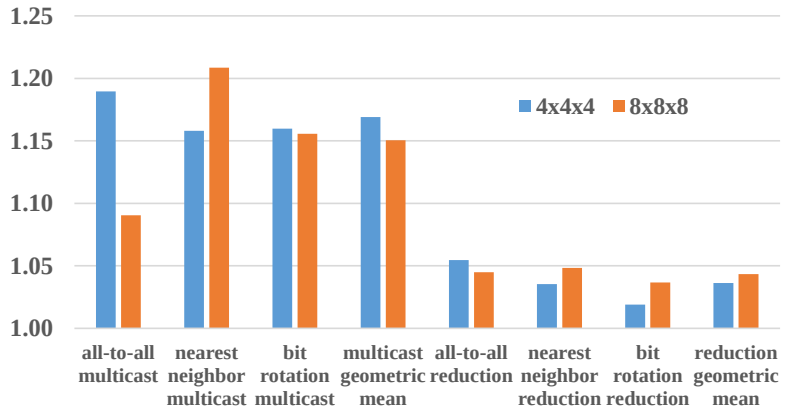


Figure 5·21: The batched experiments with three typical benchmarks (all-to-all, nearest neighbor, and bit rotation) for two kinds of network size: 4x4x4 and 8x8x8.

the $4 \times 4 \times 4$ network, each source node communicates with nearest 26 neighbors ($3^3 - 1$); in the $8 \times 8 \times 8$ network, each source node communicates with the nearest 124 neighbors ($5^3 - 1$). The patch size is set to 64 packets and injection rate is set to 1 packet per cycle per node. The results show that for the multicast operation, the latency of the offline routing solution is in all cases better than online routing with a geometric mean of over 15% for multicast and 4% for reduction. For the reduction operation, the improvement is limited by the fact that each node injects at most one packet per cycle but can consume more than one.

Figure 5·22 shows the results for the continuous multicast experiments. We add

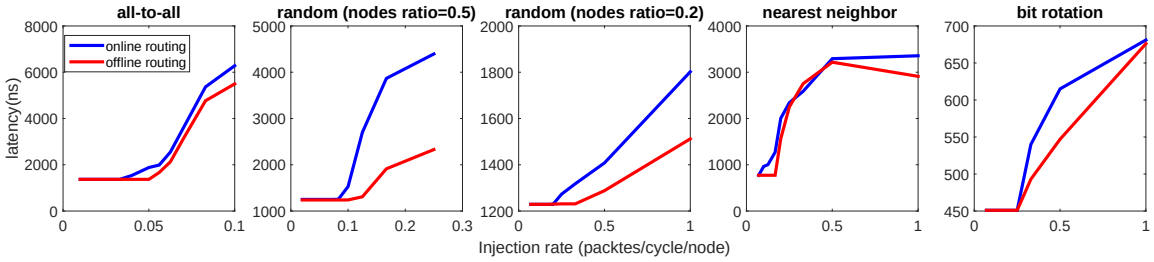


Figure 5·22: The average latency of multicast packets in $4 \times 4 \times 4$ network

two patterns; these are similar to all-to-all, but with a subset of destinations selected at random. We note that unloaded latency is maintained for higher loads for the offline design. Also, the offline design results in better average latency in nearly all cases. The exception is the nearest neighbor pattern. This is because that pattern is already symmetric and balanced leaving little room for improvement.

5.2.3 Discussion

In this section, we describe a complete communication infrastructure to support offline (statically scheduled) routing of collective communication on FPGA-centric clusters. We use table-based routing and a new router design. We propose a new offline collective routing (OCR) algorithm that takes advantage of knowledge of communication patterns to load balance network links and reduce congestion. The experiments show that this offline routing solution has significantly better performance and lower hardware cost than a state-of-the-art online routing solution. The OCR algorithm is not optimal; however, the infrastructure described supports improvements as they are developed with no change in design.

Chapter 6

3D FFT on FPGA clusters

One application we have implemented on our Nove-G# system and its infrastructure is the 3D FFT. In this chapter, we first give an overview of the 3D FFT. We then give implementation details on the Novo-G#. After that, we present our experimental results and compare them with results from previous work. We end this chapter with some discussion.

6.1 3D FFT Overview

The three dimensional Fast Fourier Transform (3D FFT) is essential to numerous applications in diverse domains. In Molecular Dynamics (MD) simulations the 3D FFT reduces the complexity of computing the long range interactions. In molecular docking, the 3D FFT computes the scores for ranking the different conformations of molecular complexes (Katchalski-Katzir et al., 1992; Vancourt et al., 2004; Vancourt and Herbordt, 2005; VanCourt et al., 2006). In imaging, the 3D FFT accelerates algorithms that decrease scan time (Kim et al., 2011).

Particularly interesting to this work is when the 3D FFT is both on the critical path and operating in a fixed sized problem domain, i.e., when strong scaling is needed. An example is MD simulations of biomolecules. These often have from a few 10s of thousands to a few 100s of thousands of particles and need to execute for E+9 to E+15 timesteps (fs) and beyond. The non-FFT part of the computation scales well and takes roughly 1s per timestep per CPU core. With a 1K core cluster,

the non-FFT MD simulation of a protein takes about 10 days for 1us simulated time (E+9 timesteps). To get into the ms range requires, e.g., 100K cores and 100 days. The problem is that, as the cluster size increases, and while the problem size remains fixed, the compute time per timestep necessarily decreases. In these examples, they are 1ms and 10 us, respectively.

These calculations define the time budget for the 3D FFT in protein simulation: preferably in the us range of compute time for FFT sizes of from 16^3 to 128^3 (Sukhwani and , 2008; Sukhwani and Herbordt, 2010; Young et al., 2009). Given the communication latencies of commodity networks, achieving these numbers poses a substantial challenge. The two ways to address the problem are to reduce the number of communicating nodes, e.g., by using accelerators and reducing the communication latency. In conventional clusters these are generally in conflict: accelerators must traverse extra hops to cooperate. This problem was solved by DE Shaw by building a dedicated ASIC-based computer, Anton (Shaw et al., 2008).

The 3D FFT data can be viewed as a cube of points, where each point represents a point of data in an FFT calculation. A N^3 point 3D FFT can be expressed as Equation 6.1.

$$F(k_x, k_y, k_z) = \sum_{z=0}^{N-1} \sum_{y=0}^{N-1} \sum_{x=0}^{N-1} f(x, y, z) W_N^{xk_x} W_N^{yk_y} W_N^{zk_z} \tag{6.1}$$

where $W_N = e^{-i\frac{2\pi}{N}}$.

In practice, the 3D FFT is calculated by decomposing it into 1D FFTs computed successively in each dimension. Since the 3D FFT has N^3 data points, each of the three dimensions requires N^2 N-point 1D FFTs for a total of $3N^2$ 1D FFT calculations. By convention, the 1D FFTs are first computed on the x dimension, then on the y dimension, and last on the z dimension. Each dimension must wait for the previous dimension to finish before it can start.

6.2 Implementation

In this section, we describe (i) the framework to generalize the communication pattern for the 3D FFT, (ii) the problem decomposition and data mapping, and (iii) a formula to estimate the latencies for various 3D FFTs ranging from 16^3 to 128^3

As described in the background section, the N^3 3D-FFT can be decomposed into three phases, one per dimension. Each phase involves N^2 N-point 1D-FFTs. Between each pair of phases, there exists a communication phase to transpose the whole 3D array to get the data ready for the 1D-FFT on the next dimension. Following (Young et al., 2009), we refer to the communication phase between X and Y as the XY corner turn and between Y and Z as the YZ corner turn. In (Young et al., 2009), the authors summarize the communication pattern of XY corner turn and YZ corner turn for two specific cases: 32^3 FFT and 64^3 FFT on 3D-torus network with 8^3 nodes. In our work, we extend these two patterns to a general case of $2^n \times 2^n \times 2^n$ 3D FFT on a 3D torus network of$2^m \times 2^m \times 2^m$ nodes (see Figure 6·1).

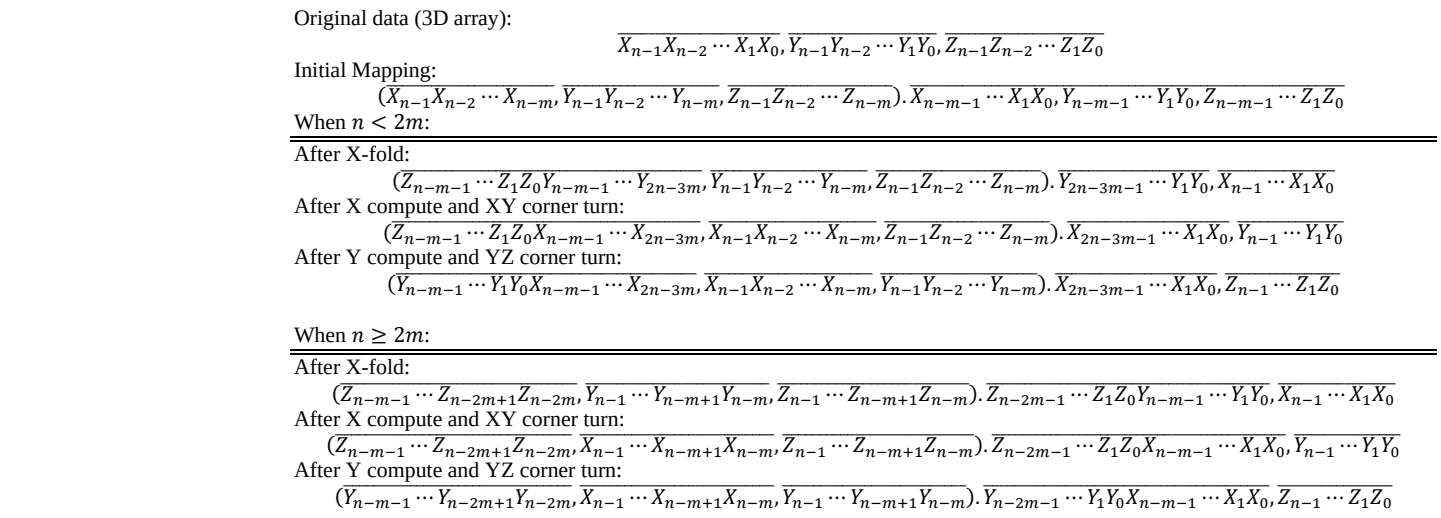


Figure 6·1: The generalized mapping $2^n \times 2^n \times 2^n$ 3D FFT problem on 3D $2^m \times 2^m \times 2^m$ torus network And data permutation pattern during two communication phases (XY corner turn and YZ corner turn).

In Figure 6·1, the first two lines show the most straightforward mapping of N^3

data points onto an M^3 torus cube, where $N = 2^n$ and $M = 2^m$. There are then two cases for the permutation patterns: $n < 2m$ and $n \geq 2m$. Both of them require that $3m \leq 2n$, which guarantees that there is at least one 1D FFT on one node, which is because there are2^{3m} nodes in total and 2^{2n} 1D FFTs in each dimension, which means there are 2^{2n-3m} 1D FFTs per node.

In Figure 6·1, there is a communication phase called X-fold, which takes place before computing the X dimension FFTs. This step transposes data from the initial mapping to get ready for this computation. In our design, this step is done offline, so the X-fold phase in Figure 6·1 does not cost any clock cycles. The permutations for the XY and YZ corner turns are also shown in Figure 6·1. Each expression determines the exact location of each datum on the network and has five overscored terms. The first three terms (in the parentheses) are binary expressions of the coordinates of the node that the data belongs to. The fourth term denotes the index of the 2^{2n-3m} 1D FFTs in that node. The fifth term indicates the index of the 2^n points of the 1D FFT input data on that node.

We now illustrate how the permutations work (XY corner turns, and YZ corner turns). Let n = 6 and m = 3.

Binary expression for data (11,47,19)

$$= \overline{x_5x_4x_3x_2x_1x_0}, \overline{y_5y_4y_3y_2y_1y_0}, \overline{z_5z_4z_3z_2z_1z_0}$$
$$= (001011)_2, (101111)_2, (010011)_2$$
$$= 11, 47, 19$$

Initial Mapping:

$$(\overline{x_5x_4x_3}, \overline{y_5y_4y_3}, \overline{z_5z_4z_3})$$
$$= (001)_2, (101)_2, (010)_2$$
$$= 1, 5, 2$$

After X fold:

$$(\overline{z_2 z_1 z_0}, \overline{y_5 y_4 y_3}, \overline{z_5 z_4 z_3}), \overline{y_2 y_1 y_0}, \overline{x_5 x_4 x_3 x_2 x_1 x_0}$$

$$= ((011)_2, (101)_2, (010)_2), (111)_2, (001011)_2$$

$$= (3, 5, 2), 7, 11$$

After XY corner turn:

$$(\overline{z_2 z_1 z_0}, \overline{x_5 x_4 x_3}, \overline{z_5 z_4 z_3}), \overline{x_2 x_1 x_0}, \overline{y_5 y_4 y_3 y_2 y_1 y_0}$$

$$= ((011)_2, (001)_2, (010)_2), (011)_2, (101111)_2$$

$$= (3, 1, 2), 3, 47$$

After YZ corner turn:

$$(\overline{y_2 y_1 y_0}, \overline{x_5 x_4 x_3}, \overline{y_5 y_4 y_3}), \overline{x_2 x_1 x_0}, \overline{z_5 z_4 z_3 z_2 z_1 z_0}$$

$$= ((111)_2, (001)_2, (101)_2), (011)_2, (010011)_2$$

$$= (7, 1, 5), 3, 19$$

This example shows data movements when mapping a 64^3 3D FFT onto an 8^3 torus. The original X, Y, and Z indexes for this datum are 11, 47, and 19 respectively. It is mapped to the node whose coordinates are (1,5,2). After the X-fold, the coordinate of the node becomes (3,5,2), and the data is mapped to the 7th 1D FFT IP. The relative address of this data on this IP is 11. In our design, the X-fold is done offline, therefore physically the initial location of this data is the 11th slot on the 7th IP on Node(3,5,2). After the XY corner turn, the datum is sent to the 47th slot on the 3rd IP on the Node(3,1,2). Finally, the YZ corner turn puts the data on the 19th data slot on the 3rd IP on the Node(7,1,5). Based on this generalized permutation pattern, we can derive the number of packets that are transmitted during each phase and how large each packet is. These are displayed in Table 6.1.

Table 6.1: The number of packets should be sent in each communication phase and the size of packets

Turn	FFT Size	Torus Size	condition	packets/node	Data/packet
xy turn	2^{3n}	2^{3m}	$2m > n$	2^{3m-n}	2^{2n-3m}
xy turn	2^{3n}	2^{3m}	$2m \leq n$	2^{m}	2^{2n-3m}
yz turn	2^{3n}	2^{3m}	$2m > n$	2^{n}	2^{2n-3m}
yz turn	2^{3n}	2^{3m}	$2m \leq n$	2^{2m}	2^{2n-3m}

To get a better sense of what would be the best network size for any particular 3D FFT, we created a generalized formula to estimate the latency of the entire 3D FFT computation. As the entire process can be decomposed into computations and communications, we can look at these two parts separately.

The computation latency is straightforward and can be easily found in Altera documentation (Altera, 2014a). The communication on the 3D torus is more complicated than the computation: the breakdown of the estimate of the communication latency is shown in Table 6.2. Since the data throughput is limited by the bandwidth of the MGTs, we need to account for the time to output all the data on one node, which is calculated using the total number of data points per node and the bandwidth of the internode links. Based on the general case above, the number of points per node should be 2^{3n-3m}. The delay on the links is another important factor of the overall latency and is calculated based on the length of the longest path in the generalized routing pattern described above.

6.3 Experimental Results

We built a cycle-accurate simulator to gather experimental results for our design. The simulator is configured with real parameters based on the capabilities of current FPGA devices. The torus size in this paper is restricted to 8^3 and 4^3, both of which are actual configurations in state-of-the-art technologies. Each node in the torus

Table 6.2: The estimated latency of communication in microseconds for various problems sizes and network sizes (BW: Bandwidth, LD: Link Delay)

Turn	FFT Size	Torus Size	Conditions	Estimation Formula
xy turn	2^{3n}	2^{3m}	$2m > n$	$\frac{2^{3n-3m}}{BW} + (2^{2m-n-1} + 2^{m-1}) \times LD$
xy turn	2^{3n}	2^{3m}	$2m \leq n$	$\frac{2^{3n-3m}}{BW} + (2^{m-1}) \times LD$
yz turn	2^{3n}	2^{3m}	$2m > n$	$\frac{2^{3n-3m}}{BW} + (2^{m}) \times LD$
yz turn	2^{3n}	2^{3m}	$2m \leq n$	$\frac{2^{3n-3m}}{BW} + (2^{m}) \times LD$

contains an FPGA. Each FPGA has a network switch and a local processing unit. Each processing unit includes some 1D FFT IPs; this number is determined by the FFT size and the chip area. The 1D FFT IPs we adopted in our design are generated using Altera FFT MegaCore (Altera, 2014a). Latency and the maximum number of IPs that fit on a high-end Altera FPGA are shown in Table 6.3. The Altera FFT IP has the best latency in streaming mode: for an N point 1D FFT, the latency cycles is N cycles. The FFT computation latency becomes trivial because of this streaming mode. Once data is produced from FFT IP, it is ready to be sent into the network. In this manner, most of the computation latency is hidden behind communication latency.

Table 6.3: Altera FFT MegaCore latency and max number of IPs could fit on an Altera Stratix V 5SGSMD8

1D FFT Size	Latency (cycles)	DSP Blocks Used	Max # of IPs per node
16	16	8	75
32	32	16	41
64	64	16	41
128	128	24	31
256	256	24	31

The performance results gathered after simulations are displayed in Table 6.4. We simulated two kinds of routers, ring, and VOQ, which are both mentioned in chapter

Table 6.4: Latency in microseconds for various problems sizes and network sizes

FFT Size	Network Size	# of FFT IP per Node	Latency
16^3	4^3	4	3.86us
32^3	4^3	4	5.30us
64^3	8^3	16	9.32us
128^3	8^3	16	25.72us

4. Their results are very close to each other. So we did not specify in Table 6.4. Problem sizes ranging from 16^3 to 128^3 were simulated. We pick the best network size for each problem size. For FFT sizes of 128^3 (and larger), congestion becomes critical (see Table 6.2), but also allows for the table based scheme to show most benefit.

In Table 6.5, we compare our results with results from CPUs, GPUs, ASICs, and other FPGA implementations. Except for ASIC(Anton) all are for single sockets. We find that communication overhead does not overwhelm the calculation and that performance of our design is faster than performance on CPUs and GPUs by at least one order of magnitude (achieving strong scaling for the target applications). Also, the FPGA cluster performance is similar to that of Anton (Young et al., 2009). For the 64^3 FFT, the presented design is faster than Anton (Young et al., 2009) by about 30%.

6.4 Discussion

In this section, we present a design of mapping 3D FFTs onto an FPGA-based cluster with a 3D torus, direct connections between FPGAs, and offline routing. Even with extremely conservative assumptions, we demonstrate strong scaling and comparable results with Anton. The main reason why our 3D FFT implementation could achieve similar performance to Anton is that although our network is not as fast as that

Table 6.5: The results for various technologies and problem sizes. Anton is for fixed point; other results are for single precision floating point. All times are in microseconds. The release date is from corporate announcements of availability in quantity. Stratix-V times are our simulation results.

Implementation Technology				Performance in μs		
Tech.	Make	Model	Parallelism	16^3	32^3	64^3
2008 era technology						
CPU	Intel	Nehalem	4 cores	38	116	983
GPU	NVIDIA	Tesla	240 SPs	54	66	257
FPGA	Altera	Stratix-III	single FPGA	4.5	DNFit	DNFit
ASIC	DE Shaw	Anton	512 PEs	Not Av.	4	13
2012 era technology						
CPU	Intel	Sandy Bridge	8 cores	22	55	288
GPU	NVIDIA	Kepler	2688 SPXs	25	29	92
FPGA	Xilinx	Virtex-7	single FPGA	3.6	21	216
FPGA	Altera	Stratix-V	512 FPGAs	3.86	5.30	9.32

of Anton, the streaming FFT IP is directly connected to the network switch, which enables the FFT computation is almost entirely hidden behind the communication. Our results directly prove the advantage of the tight coupling of communication and computation on FPGA. The overall conclusion is to demonstrate the viability of FPGA clusters for long timescale MD simulations of even modest sized proteins.

Chapter 7

3D FFT and Implications for MD on FPGA Cloud

FPGAs are currently unique in how they combine compute and general communication resources in a commodity device. This feature has led to new cloud and cluster architectures where the FPGA is both the accelerator and the communication device, the latter as part of the router or the NIC or as a bump-in-the-wire. For clouds so enhanced, such as Microsoft Catapult (Caulfield et al., 2016), there is the promise of cost-effectively executing HPC applications in a commodity data center that otherwise would require a dedicated (if not proprietary) network to scale efficiently. Here we explore this possibility through the implementation of a well-known communication-bound application, the 3D FFT. We then show the implications of these results on Molecular Dynamics (MD). Contributions are as follows.

- We believe this study to be the first for a communication-bound HPC application, the 3D FFT, on Catapult 2. We use the results from our testbed to create a model for performance of the 3D FFT on large-scale clouds. We repeat this for FPGA-centric clusters, using the Novo-G# as our testbed. We use these models to determine the sensitivity of performance to prospective network enhancements.

- We introduce the idea of phased application elasticity, which is based on the observation that some HPC applications run in phases and that those phases

may differ in scalability. We show that for the 3D FFT, elasticity can improve performance by 13%-14%. Specifically for Molecular Dynamics, the contraction and expansion communication can be merged with the data transform from physical to FFT spaces to completely hide the overhead, which results in the benefit rising to 16%-29%.

- Using these and published results on MD hardware implementations, we build a model for MD performance for likely FPGA clouds and clusters. We propose two different MD designs, one where the nodes are uniform and one where they are specialized. Finally, we present roofline graphs for current FPGA cluster and cloud architectures showing, over a range of problem sizes and node counts, where the bottlenecks lie.

Two of the most important results are as follows. For FFTs, performance is competitive with (at least) HPC technologies with CPUs and GPUs. For MD also, performance is projected to be competitive with throughput well into the range of microseconds/day. Perhaps the most important consequence is showing the efficacy of running communication-bound HPC applications in a commodity cloud if that cloud is enhanced with FPGAs in a bump-in-the-wire architecture.

7.1 FPGA-centric Clouds and Clusters

Basic device characteristics. FPGAs are COTS computing devices whose logic is configurable, rather than fixed as in a CPU or GPU, and so can be adapted to the application to obtain extremely high efficiency. Modern FPGAs have thousands of "hard" ASIC blocks–ALUs, FP Units, and Memories (Block RAMS or BRAMs)–that can be combined into independent parallel pipelines. Since on-chip communication is configurable, there is no logical restriction on the connections among these pipelines, although there are obvious physical limitations. FPGAs run at a fraction of the

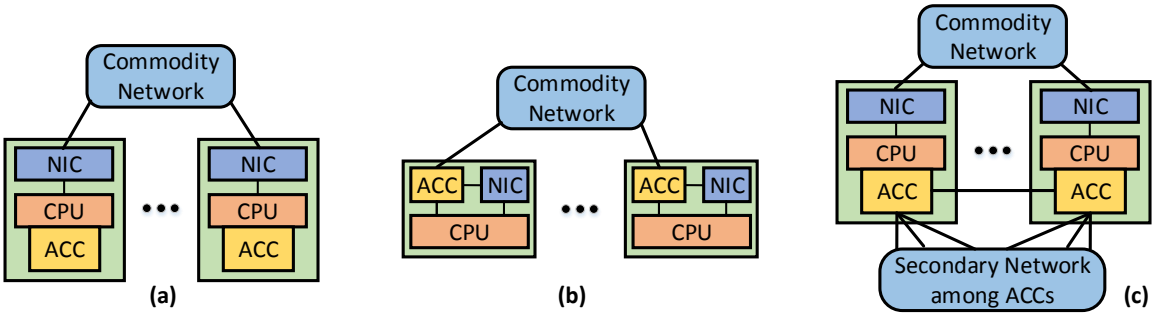


Figure 7·1: Three models for FPGA-based HPC systems. a) Standard HPC cluster. b) Catapult II. c) Catapult I and Novo-G#.

operating frequency of CPUs and GPUs, typically around 200MHz, but also draw a fraction of the power, with 25W per device being typical in production settings (see, e.g., (Putnam et al., 2014)). Configuration times are commonly in the 100s of ms; the implication is that FPGAs are flexible at the level of swapping applications, but are less likely to be reconfigured during a task.

FPGA performance metrics. Creating an HPC application for FPGAs requires creating the parallel, communicating pipelines where most cycles produce results. The best design will maximize the parallelism (number of pipelines) and the operating frequency (by simplifying interconnects). Maximizing parallelism means minimizing the resources used per pipeline. A primary metric is, therefore, the count of the various resources used in design: generic logic (ALMs), memory blocks (BRAMs), and arithmetic units (DSPs).

FPGAs are hybrid accelerators/routers. FPGAs have long been central to high-end commercial routers; the latest FPGAs have on the order of 100 100Gbs transceivers. For HPC, this co-location enables *direct* and *programmable* interconnects. *Direct* enables FPGA-FPGA and FPGA-network connections and so the bypassing of CPU, NIC, device memory, and most of the software stack. *Programmability* enables (i) data transfers to proceed with high efficiency even under substantial loads (e.g., through static scheduling), and (ii) in-network processing, such as for

collectives. These methods are also used by the ASIC-based Anton 2 (Shaw et al., 2014) and in high-frequency trading (Arista, 2013).

FPGA programming. FPGAs are programmed with a combination of HDL code (e.g., OpenCL or Verilog) and IP (function) blocks, often provided by the FPGA vendor. Because of configuration times, load balancing among modules is of particular importance; all should be active most of the time. Also, in FPGA-based HPC systems, parts of the FPGA will be preconfigured with network and memory interfaces and other system support (the *shell*). The shell takes from 10% to 50% of the FPGA resources (Caulfield et al., 2016; George et al., 2016).

Three models for FPGA-based HPC systems are shown in Figure 7·1: (a) standard accelerator as coprocessor, (b) accelerators additionally communicate directly through the primary network from a forward position ahead of the NIC (bump in the wire) and (c) accelerators instead additionally communicate directly with each other through a secondary network with the accelerators also performing NIC and routing functions.

7.1.1 Catapult II

We have two specific target systems. The first is the Catapult system from Microsoft (see (Caulfield et al., 2016) and Figure 7·1b). We refer to this as Catapult II to distinguish it from the earlier and substantially different Catapult design (see below and (Putnam et al., 2014)).

Deployment and scale. We used a 32-node testbed (two racks) to generate results. The Catapult II system has the potential to achieve datacenter scale (Caulfield et al., 2016). We model larger systems using communication data presented in (Caulfield et al., 2016) and validated here independently.

Node architecture. Each node has one FPGA and two Intel Xeon CPUs. Each FPGA is connected to one 4 GB DDR3-1600 DRAM and two independent PCIe Gen

3 x8 duplex connections for an aggregate bandwidth of 16 GB/s between the CPU and FPGA. There is also a 256 Mb Flash chip holds the known-good golden image for the FPGA that is loaded on power-on, as well as one application image.

FPGA board-level architecture. The FPGA board is designed by Microsoft and added to the PCIe expansion slot in a production server SKU. To satisfy the constraints of hardware accelerators for datacenters, the FPGAs are placed as a network-side "bump-in-the-wire." FPGAs sit between the NIC and the top-of-rack switch (ToR).

FPGAs are Altera Stratix V D5s, with 172.6K ALMs of programmable logic (28nm process), which is roughly 60% the capacity of a top-of-the line FPGA such as is used in the Novo-G# (see below).

System support. Each FPGA is partitioned into two parts: application logic (role) and the common I/O and board-specific logic (shell). The components of the shell include Ethernet MACs and PHYs, ER, and LTL. The shell also provides a tap for FPGA roles to inject, inspect, and alter the network traffic as needed, such as when encrypting network flows. The shell takes about 44% of the chip resources.

FPGA communication interface. Applications send packets to the Elastic Router (ER), which handles inter-node communication via a virtual channel. The ER forwards packets to a Lightweight Transport Layer (LTL) engine, which uses UDP for frame encapsulation and IP for routing packets across the datacenter network.

Communication bandwidth. There is a single Ethernet interface with a bandwidth of 40Gpbs.

Communication latency. See Table 7.1.

Table 7.1: The end-to-end latency on Catapult II for small packets.

Number of nodes	< 24	< 1000	$>= 1000$
Latency(us)	2	5	10

7.1.2 Novo-G#

The second particular target system is the Novo-G# (George et al., 2016). The overall architecture is shown in Figure 7·1c and is similar to the original Catapult (Putnam et al., 2014) as well as various other FPGA-centric clusters (Sass et al., 2007; Bunker and Swanson, 2013). The details of its system architecture is already described in section 3.2 and 3.3. The brief introduction is listed below.

Deployment and scale. There are currently two identical copies of the Novo-G#, each with 64 nodes configured in 4^3 3D tori (as the secondary network). The primary networks are Gigabit Ethernet and QDR InfiniBand.

Nodes architecture. Each node has one FPGA and two Xeon E5-2620V2 processors. Each FPGA is connected to two 8GB DDR3 SODIMM and two 36-Mbit SRAM memory banks and communicates with the local CPUs via PCIe v3.

FPGAs. The accelerator boards are from Gidel and populated with Stratix V GSMD8 FPGAs from Altera (28nm process). The GS-series devices are optimized for high performance, high bandwidth applications with support for up to 36 on-chip transceivers that can operate up to 12.5 Gbaud.

On-FPGA system support. The shell has memory and communication interfaces and complete general routing logic. The shell uses $< 10\%$ of chip resources.

FPGA communication interface. Applications send packets directly to each other through the secondary network.

Communication bandwidth. Besides the interface to the CPU, each FPGA has six bidirectional links, each with 40Gbps bandwidth for 480Gbps aggregate bandwidth.

Communication latency. Point-to-point latency is 175ns. Per hop switching time latency within each intermediate FPGA is just a few cycles ($< 20ns$), unless the network is very heavily loaded.

Network programmability. The Novo-G# supports table-based routing which

facilitates static scheduling to balance load and minimize collisions, which is especially useful for 3D FFTs (Young et al., 2009; Sheng et al., 2014; Lawande et al., 2016) and for routing collectives in MD (Grossman et al., 2015).

7.1.3 Methods

We describe methods used to program the two target systems, Catapult II and the Novo-G#. For both we use Verilog and IP from the Altera Megacore Library (Altera, 2014a). Also for both, we use Altera Quartus II v15.0 for synthesis and place-and-route and ModelSim for simulations (behavioral and post-fitting). For Catapult II we use the Microsoft Catapult II tool set to configure the FPGAs. System interfaces are through the shell as described above. For Novo-G#, we use the Gidel ProcWizard tools to generate the framework (shell) and to configure the FPGAs.

7.2 FFTs and Molecular Dynamics

7.2.1 FFT and FPGAs

The input to a 3D FFT an N^3 cube of points. For fine-grained parallel implementations (Lawande et al., 2016; Sheng et al., 2014; Young et al., 2009), the 3D FFT is decomposed into 1D FFTs which are computed successively in each dimension. Each of the three dimensions requires N^2 N-point 1D FFTs for a total of $3N^2$ 1D FFT calculations. Each dimension computation waits for the completion of the previous dimension computation before proceeding.

The 1D FFTs are implemented using vendor IP blocks of size N. The vendor IPs are extremely efficient being maximally tuned to the particular target device. The IPs used here (Altera, 2014a) are single precision floating point, have streaming inputs and outputs of one element per cycle, order (unshuffle) the output stream, and have a latency of N cycles. For $N = 64$, the maximum number of IPs that can fit on one

node in Catapult II is 12; for the Novo-G# it is 64.

A constraint on performance is the finite number of BRAMs. Each IP demands one datum per cycle, which requires that data are distributed across BRAMs such that there are no collisions between IP streams. For a single FPGA and likely N, this is possible only in 1 and 2 dimensions, but not for 3. This problem is solved with an internal routing network (crossbar). The result of this design is that for a single FPGA, given N and I IPs, the 3D FFT is computed in roughly $3N + 3N^3/I$ cycles.

For clusters there is an analogous data ordering problem that is solved by performing a cluster-wide transpose (see, e.g., (Lawande et al., 2016; Sheng et al., 2014; Young et al., 2009)). The latency of the 3D FFT, therefore, has an additional term that is a function of communication performance, which, in turn, is a function of network and the number of processors P.

7.2.2 MD, FPGAs, and Strong Scaling

MD is an iterative application of Newtonian mechanics to ensembles of atoms and molecules (particles). MD simulations proceed in iterations consisting of two phases, force computation, and motion integration; since motion integration is $< 1\%$ of computation and requires little communication, we ignore it. The force computation itself has two phases, range-limited and long range. For the range-limited, each particle interacts with all particles within a cut-off radius (generally from 9A to 12A) which reduces the complexity of this phase from $N^2/2$ to roughly $500N$. Various methods are used to effect this reduction including cell and neighbor lists (in software) and cell lists and distance filtering (in hardware). The long-range force computation, using, e.g., PME, requires multiple steps (Phillips et al., 2002): (i) mapping charges to a grid, (ii) scattering data for efficient Fourier space computation, (iii) 3D FFT, (iv) gathering data back into physical space, and (v) applying forces to the particles. The remaining computations include bonded forces and data movement, but require less

$< 1\%$ of communication and computation.

Hardware implementations of MD have been studied with the Anton series of ASIC-based systems (Grossman et al., 2015; Shaw et al., 2007; Shaw et al., 2014; Young et al., 2009) and numerous single FPGA studies (e.g., (Alam et al., 2007; Chiu and Herbordt, 2010; Scrofano et al., 2008)). Details of range-limited pipelines can be found in (Shaw et al., 2007; Chiu and Herbordt, 2010), range-limited communication in (Grossman et al., 2015), and the 3D FFT in (Young et al., 2009).

Roughly 90% of the FLOPs are used for computing the range-limited force. Communication bandwidth is more even between long range and range-limited phases. But the crucial point is that the long range force *requires global communication while the range-limited only local.* This is what leads to the difficulty in strong scaling MD, especially for smaller simulation sizes such as used to simulate protein folding. We illustrate this idea as follows. It is critical to simulate reality into the ms regime. With standard timesteps of 2E-15s, this requires E12 timesteps. To complete these in say, one week (100 μs simulated time per day), each timestep must be computed in roughly 2-5 μs of real time, depending on other assumptions. For *any* problem size, global communication at this time scale becomes extremely challenging with increasing P.

7.3 3D FFT on Catapult II and Novo-G#

7.3.1 3D FFT on Catapult II

An overview of FFT implementations on FPGA-based systems was given above; there are three phases of N^2 per-dimension 1D FFTs separated by corner turns. Figure 7·2 shows the state transition diagram of the 3D FFT on Catapult II. All nodes start in the IDLE state. Data and routing tables are loaded into each node after which they enter a wait state. We designate one node as head node; its primary purpose

is to multicast *start* to the other nodes. After receiving *start*, nodes compute the X dimension 1D FFTs and then group data into packets and transmit them (the XY corner turn). In parallel, a monitor on each node starts tracking data arrivals. When all expected Y dimension data have arrived, the node begins the Y dimension computation. The flow works similarly for Z dimension. After X, Y, and Z have completed, data are written back, and nodes return to the wait state.

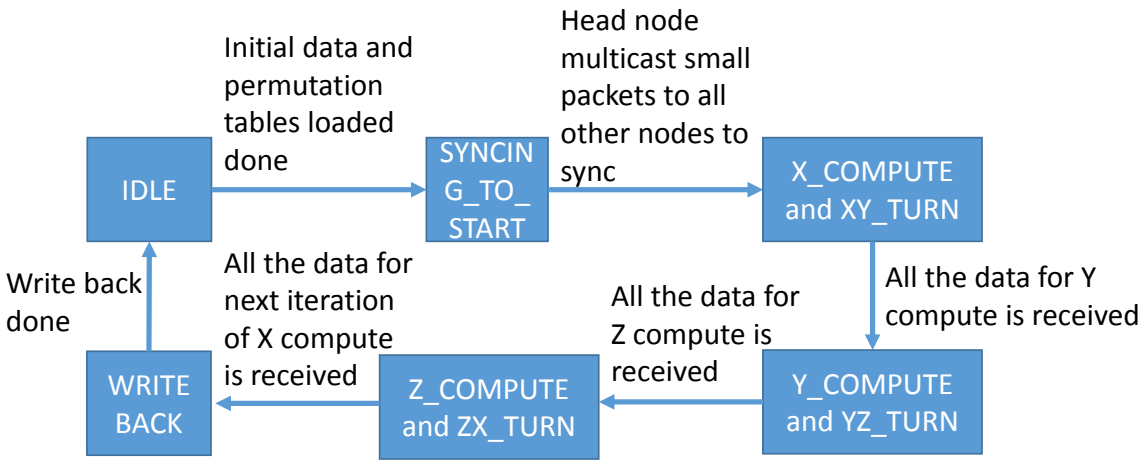


Figure 7·2: The state transition graph for 3D FFT implementation on each Catapult II node

The design for each Catapult II node is shown in Figure 7·3. Recall the 3D FFT has three communication phases: XY corner turn, YZ corner turn, and ZX corner turn. So we instantiate two sets of RAMs to store the temporary results of the three phases. During the XY and ZX corner turns, data is read from RAM set 0 and written to RAM set 1. During the YZ corner turn, data is read from RAM set 1 and written to RAM set 0. After data are read from the RAM, they are streamed into several 1D FFT IPs. After the data are streamed out from the FFT IPs, they are tagged by a precomputed permutation routing header. Each permutation header contains the routing information for each datum. This includes the destination node ID, the RAM slice ID, and the RAM address. If the destination is local, then the data is routed to

the internal crossbar and back to the other set of local RAMs. Otherwise, the data are routed to the external crossbar and then to the network interface. An FSM on each node controls the data flow.

One important design parameter is the number of 1D FFT IPs per node. On Catapult II, there are two types of constraints: chip resources (ALMs, BRAMs, DSPs) and network bandwidth. By examining the resource consumption of the FFT IPs for different problem sizes (N), we can determine that for common N, the number of FFT IPs that should be placed in one Catapult II node is four. The reasoning is as follows. First, four FFT IPs fit for all cases. Second, each FFT IP produces one datum per cycle, which is 85 bits wide. The bandwidth of the network interface on each Catapult II node is 256 bits per cycle. This means that three 1D FFT IPs can saturate the network bandwidth and more IPs per node do not improve performance. We select 4 IPs per node for convenience.

We have parameterized our design for various FFT sizes N and numbers of nodes P. For our experiments we have run FFTs with $N = 16, 32, 64$, and 128 on a 32 node testbed. Figure 7·4 shows the resource utilization for different problem sizes; this includes all FFT logic (IP router, and control logic) as well as the shell. We find that only ALMs and BRAMs are significant. Assuming the maximum 4 IPs per node, the consumption of ALMs increases slightly with N but remains below 65%. For BRAMs, for each N the number required BRAMs decreases with P as the data are distributed. For $N = 16$ and $N = 32$, all data fit in a single node. For $N = 64$ we need at least 2 nodes, for $N = 128$, at least 16 nodes.

Measured performance is displayed in Figures 7·5 and 7·6. In the rest of this subsection, we make basic observations. In the next subsection, we build a model and discuss scalability and sensitivity to architectural parameters. Single-node performance is better than 2-node performance. This is because the bandwidth of the

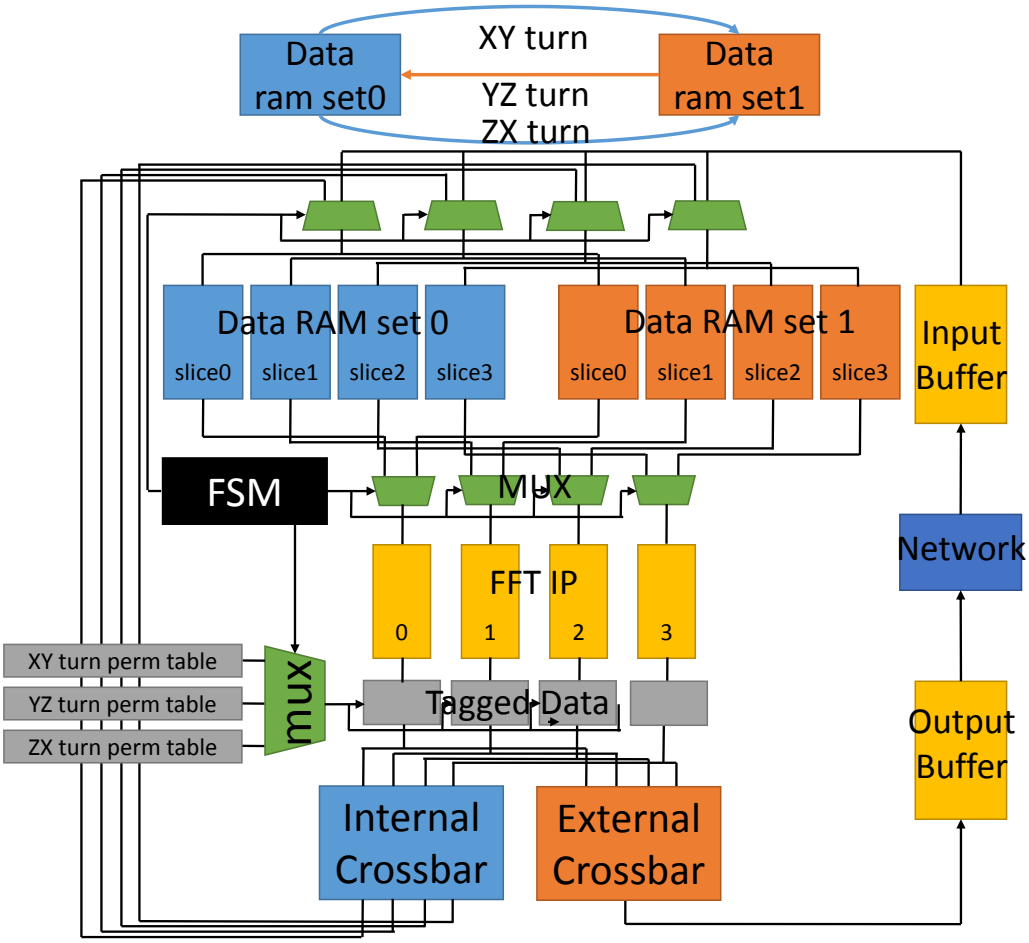


Figure 7·3: 3D FFT implementation on each Catapult II node

internal crossbar is much higher than the bandwidth of the network interface. Performance improves with $P \geq 4$. However when $P > 16$ for $N = 16, 32$, and 64 the benefit diminishes. Missing data points (for $N = 32$ and 64) require off-chip transfers and are less likely to be of practical interest.

The implementation overlaps communication and computation; data streaming from the IPs is directed to the network interface. A small part of the computation latency cannot be hidden. Figure 7·6 shows the total communication time and the marginal (nonoverlapped) computation time. In Catapult II, the communication latency has two parts, transmission, and time-of-flight. Transmission latency is pro-

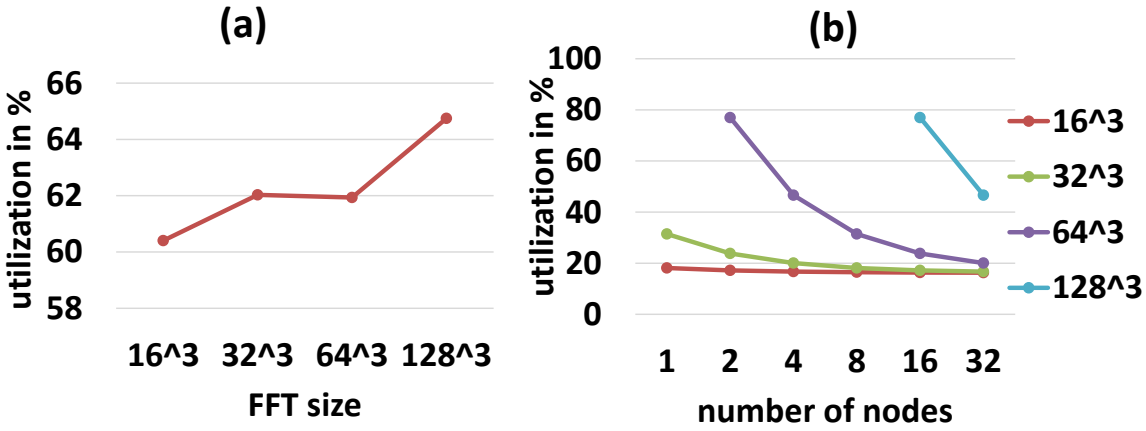


Figure 7·4: Per node resource utilization on Catapult II: (a) ALMs (b)BRAMs

portional to the number of bytes. Time-of-flight includes packet formation and so is proportional to the number of flits in the packet. As a result communication time increases for $N = 16, P = 16$ and $N = 32, P = 32$.

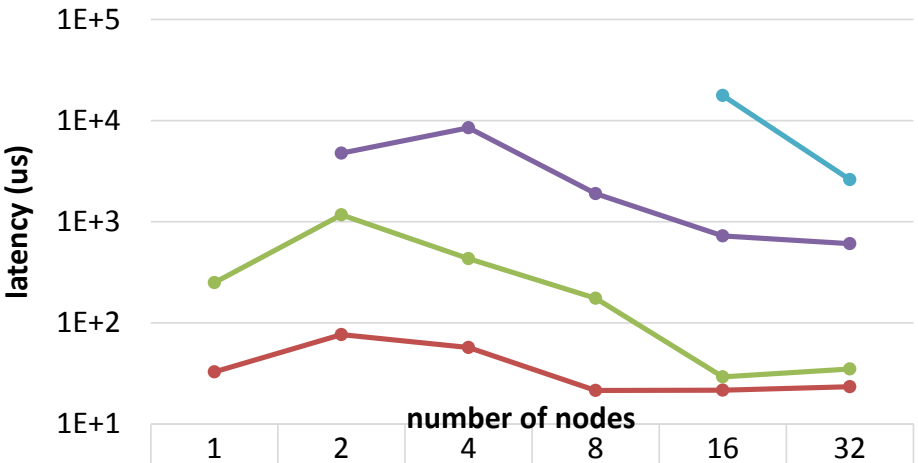
7.3.2 3D FFT Models for Catapult II and Novo-G#

Based on empirical results from the previous section and other published studies (Young et al., 2009; Sheng et al., 2014; Lawande et al., 2016), we have created models for 3D FFT performance at larger scale for both FPGA clouds and clusters.

In both architectures, because the computation latency is almost entirely hidden, $Latency = 3 \times Latency_{cornerturn}$. The corner turn latency has two components, transmission and time-of-flight. The transmission latency is the data size divided by the effective bandwidth. The time-of-flight latency is the number of flits times the latency to send one flit:

$$Latency_{cloudcornerturn} = \frac{N^3 \cdot datalength \cdot (P-1)}{p \cdot BW_{cloud}} + Latency_{oneflit} \cdot \frac{N^3 \cdot datalength}{P^2 \cdot flitsize} \tag{7.1}$$

where N is the FFT size, P is the number of nodes, and $datalength$ is the data size



	1	2	4	8	16	32
16^3	32.8	76.6	57.1	21.5	21.6	23.4
32^3	250.4	1169.4	431.8	175.3	29.3	35.1
64^3		4761.5	8478.4	1895.2	725.7	607.3
128^3					17678	2606.5

Figure 7·5: The performance of 3D FFT on Catapult II

in bits, including tag and payload. In our design, data is 85 bits. BW_{cloud} is the effective bandwidth, which depends on the number of 1D FFT IPs instantiated on each node. For example, if a large number of nodes is used, the number of IPs on each node might be smaller than 3, which would result in less than the maximum bandwidth being used. The term $\frac{P-1}{P}$ in Equation 7.1 accounts for nodes not sending data to themselves.

For the Novo-G# cluster, the corner turn latency again has transmission and time-of-flight terms with transmission latency equal to the data size divided by the effective bandwidth. Time-of-flight, however, is as you would expect on a cluster, i.e., proportional to the number of hops:

$$Latency_{clustercornerturn} = \frac{\frac{N^3 \cdot datalength}{P} \cdot \frac{P-1}{P}}{BW_{cluster}} + 3 \times Latency_{onehop} \cdot 2^{\lfloor \log_2 P/3 \rfloor - 1} \tag{7.2}$$

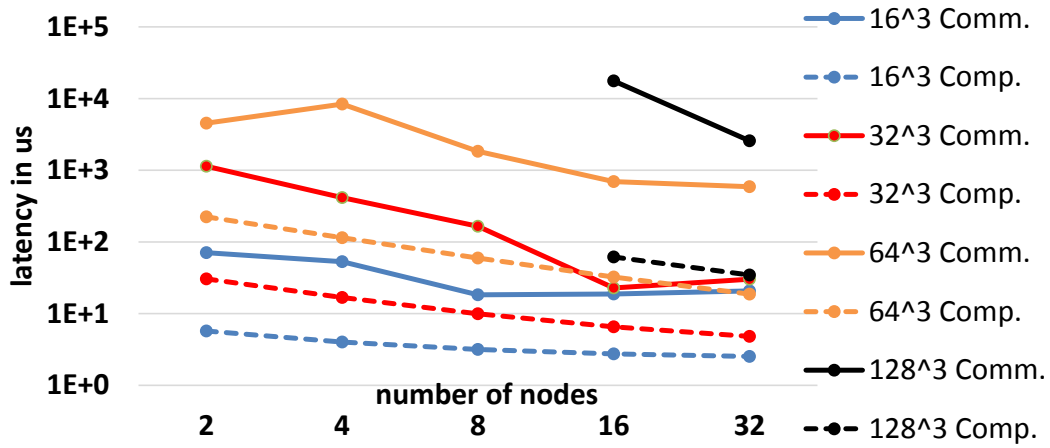


Figure 7·6: The breakdown of performance of 3D FFT on Catapult II into communication and nonoverlapped computation.

N, P, $datalength$, and $BW_{cluster}$ are all as before. Recall that each Novo-G# node has 6 duplex 40 Gpbs links. We therefore need 18 FFT IPs to saturate the bandwidth. In many cases, the number of IPs on each node is smaller than 18. $2^{\lfloor \log_2 N/3 \rfloor - 1}$ is the network diameter.

Figure 7·7 shows modeled performance of the 3D FFT for both cloud and cluster for various N and P. We make the following observations. (i) For very small N and P, a single node may be best. This is because the bandwidth of internal crossbar is higher than the bandwidth of one network interface, and small FFT sizes are easy to fit on a single FPGA. (ii) For bigger N, increasing P improves performance until a plateau is reached. As expected this is reached earlier for smaller N. The reasons (for these particular architectures) are, first, because of increased network latency, and, second, because too many nodes result in too few FFT IPs in one node so that the bandwidth of each node is not fully used. (iii) Even for small problem size ($N = 32$), performance continues to improve up to a surprisingly large number of nodes: 32 for the cloud and 128 for the cluster. (iv) Performance on clusters has a 1-2 order of magnitude advantage over the cloud. This is mostly because the cluster has much shorter time-of-flight latency, but also because there are 3 to 4 times as many IPs per

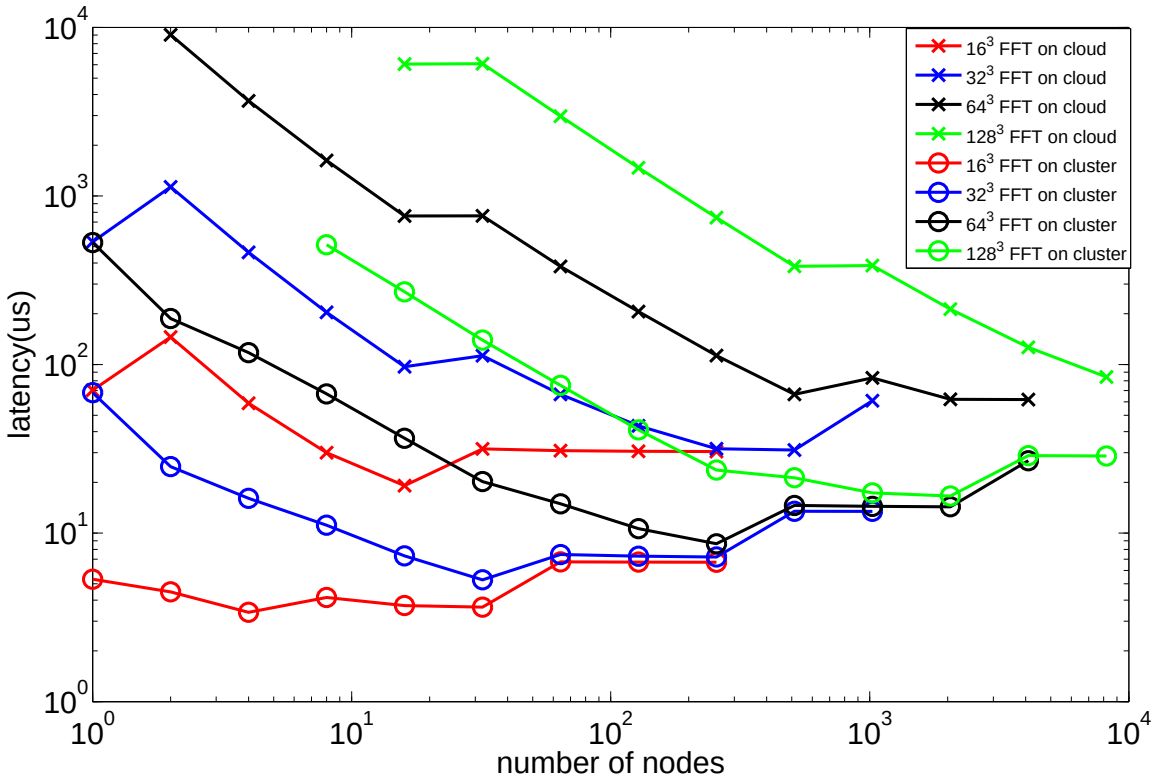


Figure 7·7: The modeled FFT performance

node.

Figure 7·8 compares performance across architectures including ASICs (Anton), CPUs, and GPUs as well as the FPGA clouds and clusters (from Figure 7·7). Each data point represents the best performance recorded for *any* P for that particular N (and architecture), i.e., the limit of strong scaling for that problem size N. Perfect scaling would be a horizontal line.

Data points for "ASICs" are for Anton (Young et al., 2009); performance for Anton II is certainly better, but we know of no published results. Because of the difficulty in scaling (which is the primary motivation for this work), results for CPUs and GPUs for small FFTs ($N \leq 64$) are rarely available. The ones we found are dated and inferior to those run obtained on recent workstations.

The CPU data points for $N = 16$, 32, and 64 are from our measurements on an

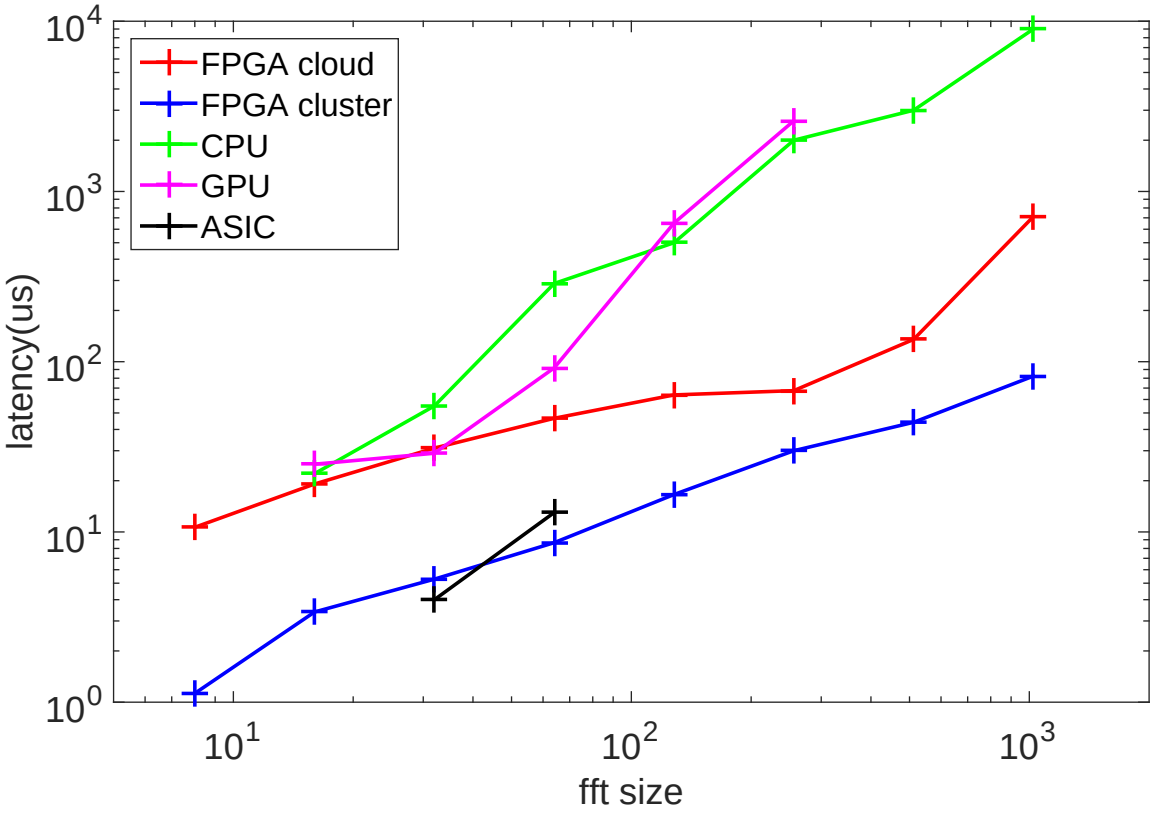


Figure 7·8: The comparison of 3D FFT performance on different platforms

Intel Xeon E5-2680 CPU with eight cores by running MKL. The CPU data points for $N = 128$ and 256 are from FFTW on 4K cores of a BlueGene/Q FERMI cluster (Guarrasi et al., 2014). The CPU data point $N = 512$ is from P3DFFT on 262144 cores of the BlueGene/Q at the Julich Research Center (Pippig, 2013). The CPU data point $N = 1024$ is from 262152 cores on the K computer (Jung et al., 2016). The GPU data points for $N = 16$, 32, and 64 are from our measurements on a Nvidia Tesla K20c running CuFFT. The data points $N = 128$ and 256 come from measurements on a GeForce GTX 570GPU (Abdellah et al., 2012). For these last points, performance on new GPU clusters coupled with NVLink is likely to improve performance, perhaps by an order of magnitude. This, however, does not change the overall trend.

Overall, we observe the well-known difficulty in scaling the 3D FFT. We also note that the FPGA cluster and cloud perform well. This is mainly because these architectures have dedicated high-bandwidth and low-latency inter-accelerator interconnect, which is critical to communication-bound applications.

For the rest of this section, we present results concerning architectural exploration, in particular with respect to sensitivity of performance of the two FPGA architectures to variations in latency and bandwidth. Figures 7·9 and 7·10 show results from a bandwidth sensitivity test based on our model. Latency is fixed to current values. For FPGA clouds, the current 40-Gbps bandwidth is sufficient; the FPGA would need to be upgraded for there to be a benefit. Decreasing bandwidth to $<$ 10Gbps results in linear performance degradation. For FPGA clusters, for small FFT sizes ($N < 32$), the current 40-Gbps bandwidth is sufficient (given the current FPGAs). But for big FFT size, higher bandwidth is effective.

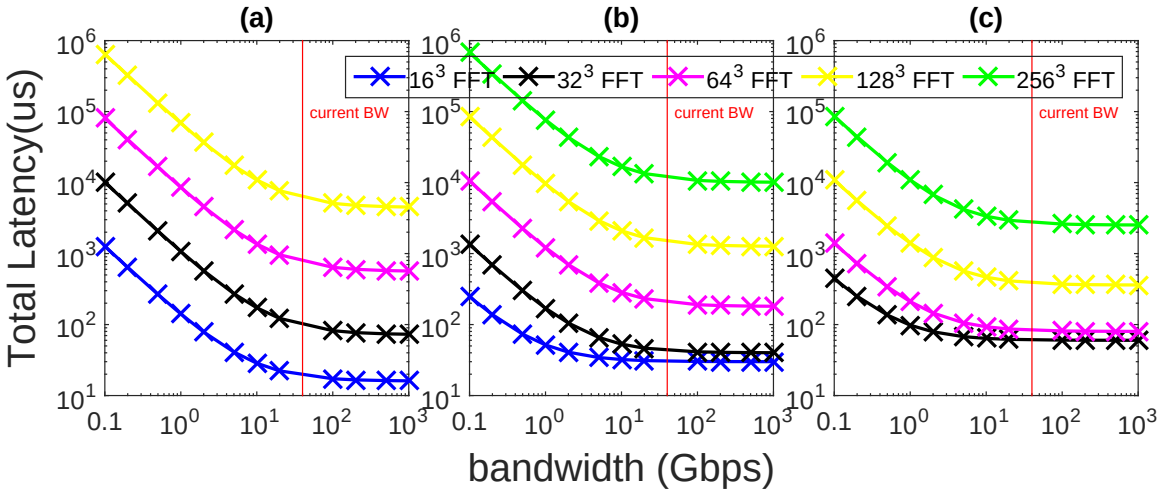


Figure 7·9: The network bandwidth sensitivity test for 3D FFT on FPGA cloud, (a) 16 nodes, (b) 128 nodes, (c) 1024 nodes

Figures 7·11 and 7·12 show results from a latency sensitivity test. Bandwidth is fixed to current values. For FPGA clouds, current end-to-end latency is shown in Table 7.1. From Figure 7·11, we see that in all cases performance is sensitive to

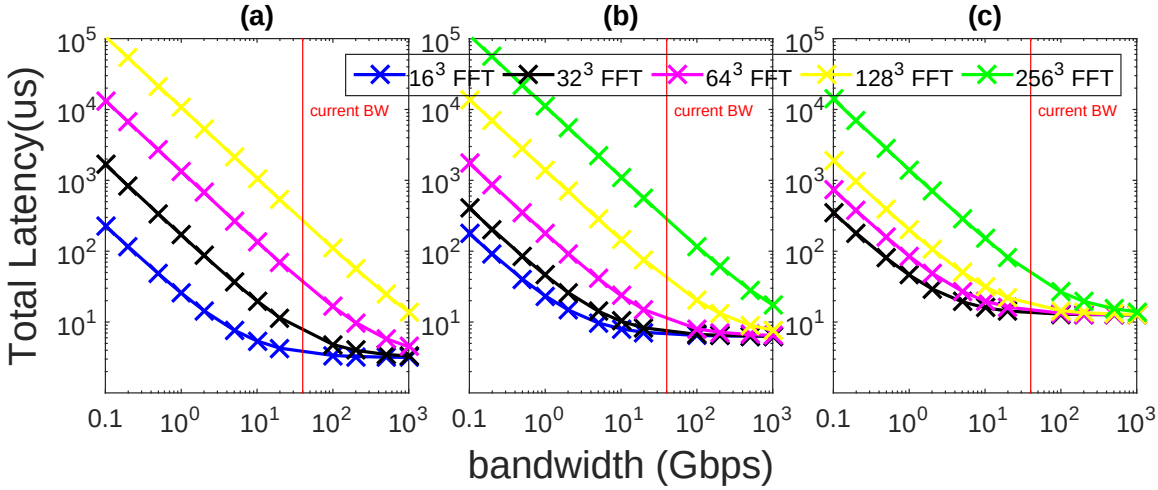


Figure 7·10: The network bandwidth sensitivity test for 3D FFT on FPGA cluster, (a) 16 nodes, (b) 128 nodes, (c) 1024 nodes

latency down to the current minimum of 2μs. Larger networks are more sensitive, benefitting from reduced latency down to 0.1μs. For FPGA clusters, current per hop latency (0.2μs) can satisfy the need of big FFT sizes ($N > 128$); performance is currently bound by the network bandwidth. But for smaller FFT sizes there is sensitivity to the network latency.

We also observe that when increasing the network latency, FFT performance on FPGA clusters converges for different FFT sizes, while FFT performance on FPGA clouds does not. This is because time-of-flight latency for the FPGA cloud is linear in the number of flits, while for the FPGA cluster it is a constant.

7.4 Optimizing Performance with Contraction

7.4.1 Contraction Model

Some applications have multiple phases each with different scalability. That is, it is possible for the maximum *useful* number of nodes P to be different for the different phases. This is true in MD: the range-limited and bonded force computations require only local communication, and so scale well, while the long-range force computation

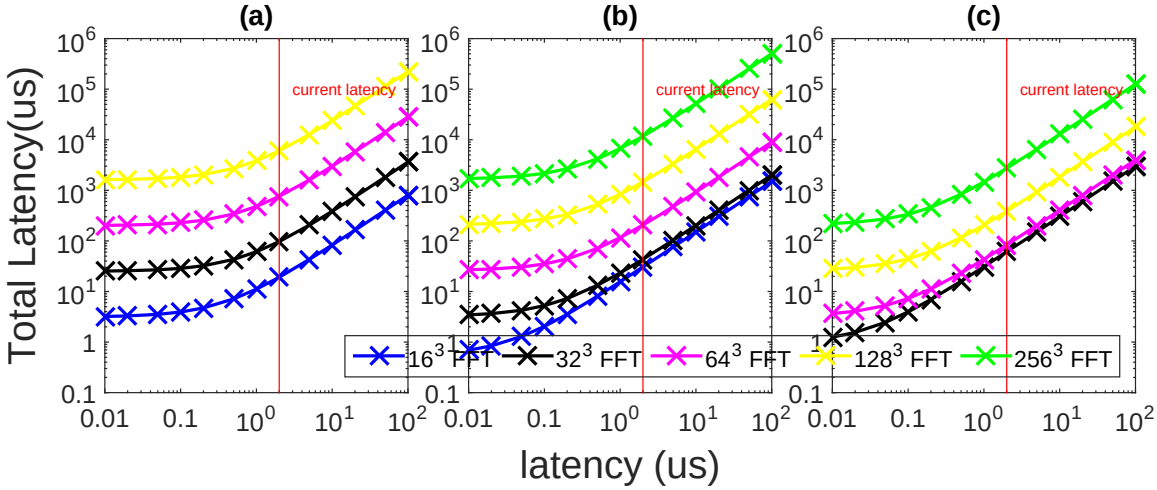


Figure 7·11: The network latency sensitivity test for 3D FFT on FPGA cloud, (a) 16 nodes, (b) 128 nodes, (c) 1024 nodes

(with the 3D FFT) does not. For the 3D FFT, each N has an optimal P: too few nodes and not enough parallelism is applied; too many nodes and the application becomes unnecessarily communication bound. It is therefore plausible in MD for P nodes to be used for one phase and a subset with $< P$ nodes to be used for another. The prospect is particularly promising for FPGA accelerators because of their configuration time: Fewer FPGAs configured to do the 3D FFT means more FPGAs to do other work.

To summarize: elasticity of phased applications using contraction and expansion can improve performance of certain HPC applications. The method requires that, at the end of the "P" phase, data be transferred to the subset "$< P$" (contraction). After the "$< P$" phase, data are returned to the original nodes (expansion). Apparently, the benefit of using fewer nodes must outweigh the overhead of the expansions and contractions.

It is also likely, however, that much of the overhead communication can be hidden. That is because the transition from one phase to another, e.g., range limited to long range, is itself likely to require data movement. In MD we see that all of the

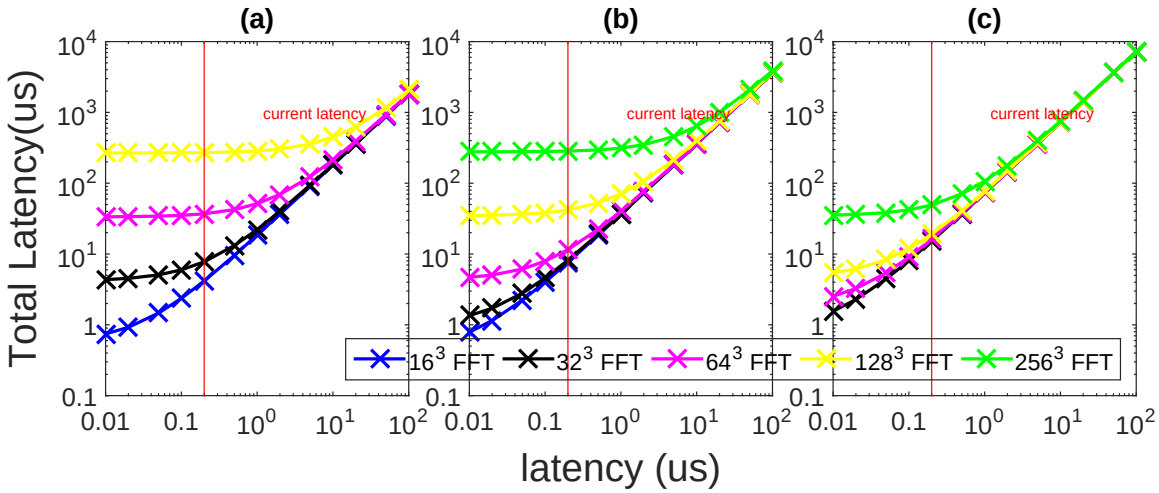


Figure 7·12: The network latency sensitivity test for 3D FFT on FPGA cluster, (a) 16 nodes, (b) 128 nodes, (c) 1024 nodes

communication overhead of expansion and contraction is hidden.

Expansions and contractions are mirror inverses with similar latency. For FPGA clouds, the contraction latency is

$$Latency_{contraction_{cloud}} = \frac{\frac{N^3 \cdot datalength}{P}}{BW_{cloud}} + Latency_{oneflit} \cdot \frac{N^3 \cdot datalength/P^2}{flitsize} \tag{7.3}$$

Most of the variables are the same as in Equation 7.1. But P is the number of nodes after contraction and BW_{cloud} is the raw cloud bandwidth (40Gbps). The latency spent on the corner turn is the same as in Equation 7.1. For FPGA clusters, the contraction latency is

$$Latency_{contraction_{cluster}} = \frac{\frac{N^3 \cdot datalength}{P}}{BW_{cluster}} + 3 \times Latency_{onehop} \cdot 2^{\lfloor \log_2 P/3 \rfloor - 1} \tag{7.4}$$

The equation is analogous to the previous except for the difference in raw bandwidth (240Gbps) and the term for network diameter.

7.4.2 Contraction Results

We again model the 3D FFT on FPGA clouds and clusters, this time with contraction. We find that we can always find an optimal P_{FFT} to run the FFT. If P_{FFT} is too large, then the corner turn latency is significant, and the bandwidth cannot be fully utilized. If P_{FFT} is too small, the contraction/expansion latency becomes too large because of the limited fan-in bandwidth of the FFT nodes limit on IPs. Table 7.2 shows the optimal contraction size for different 3D FFT sizes on FPGA clouds and clusters.

Table 7.2: The optimal contraction size for 3D FFT on FPGA cloud and cluster

FFT size	Best contraction size for cloud	Best contraction size for cluster
8^3	1	1
16^3	16	32
32^3	512	32
64^3	512	256
128^3	512	2048
256^3	8192	2048

Table 7.3 shows the performance benefits of contraction, geometric means of 13% and 14% for cloud and cluster architectures respectively. The benefits are greater for smaller problem sizes because performance at this level is bounded by network latency. For larger problem sizes, the main bottleneck is the per node bandwidth on each node.

For MD, the contraction and expansion latencies can be *entirely hidden* for the P_{FFT}. This is because there is already communication required to set up and tear down the FFT: the transformation from physical to FFT spaces and vice versa. The MD-specific results are also shown in Table 7.3. The geometric means are 16% and 29% for cloud and cluster architectures, respectively.

Table 7.3: The 3D FFT speedups by applying contraction. "in MD" means that contraction overhead is hidden.

FFT size	cloud	cloud in MD	cluster	cluster in MD
8^3	1.45	1.33	1.37	2.00
16^3	1.21	1.45	1.14	1.26
32^3	1.09	1.11	1.15	1.31
64^3	1.11	1.14	1.12	1.24
128^3	1.01	1	1.06	1.10
256^3	1	1	1.01	1.02
Geo Mean	1.13	1.16	1.14	1.29

7.5 MD Strong Scaling

Complete implementation of MD into FPGA cloud or cluster (for an Anton-like system) is still work in progress, but we can accurately model the performance of such systems using published results of hardware implementations (e.g., from DE Shaw and various FPGA projects cited earlier).

7.5.1 Models for MD performance estimation

In this section, we model complete MD, including integration of the 3D FFT for FPGA clouds and clusters. Long-range and range-limited force computations both require two kinds of resources, bandwidth, and on-chip resources. We propose two designs. In the first, all nodes are uniform. Each FPGA is configured so that all parts of the computation are supported. In the second, FPGAs are specialized, some for range-limited (including mapping) and some for long range. *Specialized* uses the contraction method from the previous section.

The range-limited force must be computed every iteration while the long-range force is generally computed every other iteration. So latency per MD iteration can

be expressed as

$$Latency_{MD} = \max(0.5 \times Latency_{LongRange}, Latency_{ShortRange}) \tag{7.5}$$

The long-range force computation latency can be expressed as

$$Latency_{LongRange} = Latency_{chargemapping} + Latency_{3DFFT} \tag{7.6}$$

For the 3D FFT and the *uniform* design we use Equations 7.1 and 7.2. Otherwise we use Equations 7.3 and 7.4. Charge mapping latency can be expressed as

$$Latency_{chargemapping} = \frac{N/P}{chargemappingthroughput} \tag{7.7}$$

where N is the number of particles and P is the number of nodes. Other work shows that *chargemappingthroughput* is 450 particles per cycle.

The range-limited latency is determined by the larger of the latencies for computation and communication. This can be expressed as

$$Latency_{ShortRange} = \max(Latency_{ShortRangeComp}, Latency_{ShortRangeComm}) \tag{7.8}$$

The computation latency of the range-limited force is determined by the number of range-limited force pipelines that can fit on one node and the number of forces that can be evaluated per force pipeline. Each force pipeline evaluates one range-limited force per clock cycle; the latency (pipeline depth) is around 100 cycles. For each particle, we need to evaluate the range-limited force between it and the other particles within a cut-off radius r. According to Newton's 3rd Law, we need to evaluate only half that number. Let the side length of cube that each node holds be c, then the

number of particles held by one node is $D \cdot (\frac{c}{r})^3$, where D is the number of particle in the volume r^3. The latency of the range-limited force can then be expressed as

$$Latency_{ShortRangeComp} = T(\frac{2}{3}\pi \cdot D^2 \cdot (\frac{c}{r})^3/K + 100) \tag{7.9}$$

where T is the clock cycle time, and K is the number of force pipelines that can fit on one node. Since each force pipeline takes about 10K ALMs, the maximum number of force pipelines that can fit on a Catapult II node is 8 and on a Novo-G# node is 20. But the actual value of K depends on the ALM allocation between long-range and range-limited. For the *specialized* design, K equals the maximum. For the *uniform* design, K depends on the left space after the FFT IPs are mapped.

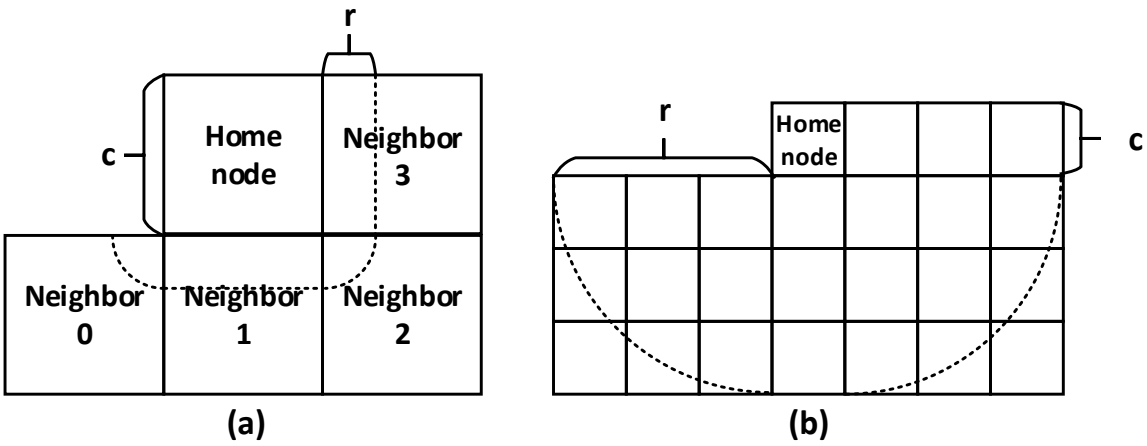


Figure 7·13: Short range communication: (a) $c > r$, (b) $c < r$

To prepare for the range-limited computation, each node must acquire positions of all the particles from neighboring nodes that are within the cut-off of the particles it holds. This is illustrated in Figure 7·13. Based on N3L, we only need to acquire positions of particles within half of the cut-off radius. After the range-limited forces are computed, they need to be distributed back to the outside particles. So there are two phases of collective communication: particle position multicast and range-limited force collection. These two communication phases are the inverse of each

other so (with hardware support) their latencies are similar. The range-limited force communication latency can be expressed as

$$Latency_{ShortRangeComm} = 2 \times Latency_{Multicast} \tag{7.10}$$

The particle position multicast latency is different for FPGA clusters and clouds. But they both can be expressed as

$$Latency_{Multicast} = \frac{128 \times MulticastParticleNum}{BW} + Latency_{onthefly} \tag{7.11}$$

Each position datum is 128 bits. In order to compute the number of particle to be multicast, we need to compute D times the volume of a shell with $r/2$ thickness outside a c^3 cube, which can be expressed as

$$\begin{aligned} MulticastParticleNum &= D \cdot (8V_{corner} + 12V_{side} + 6V_{face}) \\ &= D \cdot (8\frac{\pi}{6}(\frac{r}{2})^3 + 12c\frac{\pi}{4}(\frac{r}{2})^2 + 6c^2\frac{r}{2}) \end{aligned} \tag{7.12}$$

For FPGA clouds, the BW equals the allocated bandwidth. But for FPGA clusters, the allocated bandwidth is shared among multiple nodes (because of the multicast pattern), which can be estimated using

$$BW_{cluster} = \frac{BW_{allocated}}{(8\frac{\pi}{6}(\frac{r}{2})^3 + 12c\frac{\pi}{4}(\frac{r}{2})^2 + 6c^2\frac{r}{2})/c^3} \tag{7.13}$$

where $BW_{allocated}$ is the allocated bandwidth for the range-limited force. In the *uniform* design, it equals to the total bandwidth (240 Gbps) minus the bandwidth allocated for long-range force computation. For the *specialized* design, it equals to the total bandwidth (240 Gbps).

The time-of-flight latency for the FPGA cloud is linear in the number of flits in

the largest packet, which can be expressed as

$$Latency_{onthefly} = \begin{cases} Latency_{oneflit} \left\lfloor \frac{128(c/r)^2 D}{2 \cdot flitsize} \right\rfloor, & \text{if } \frac{c}{r} \geq 0.5 \\ Latency_{oneflit} \left\lfloor \frac{128(c/r)^3 D}{flitsize} \right\rfloor, & \text{otherwise} \end{cases} \tag{7.14}$$

The time-of-flight latency for FPGA clusters is linear in the number of hops on the longest path, which can be expressed as

$$Latency_{onthefly} = 3 \times Latency_{onehop} \cdot \left\lceil \frac{r}{2\sqrt{3}c} \right\rceil \tag{7.15}$$

7.5.2 Evaluation

Figure 7·14 shows the resulting performance estimate for the *uniform* design. To generate the data points, for each N and P we sweep through the different allocation ratios of bandwidth and ALMs between long-range and range-limited components. All the data points in Figure 7·14 are based on the best allocation ratios. We find that, on each node, to balance the speed of long-range and range-limited phases, the number of 1D FFT IPs per nodes can be as small as one, and the allocated bandwidth for long-range communication can be as small as 5% of the total bandwidth. For this design, clusters are on average about 2.1× faster than clouds.

An important result is the range of strong scaling, that is, for a given problem size N, the cluster size P at which there is no longer performance benefit to adding nodes. In Figure 7·14, we see that for very small problem sizes (N = 10K) this point is reached at around 256 nodes. For medium sized problems (N = 250K), this point is greater than 4K nodes.

Figure 7·15 shows the resulting performance for the *specialized* design. Again, all the data points in Figure 7·15 are based on best allocation ratios for each case, but this time at the level of the node rather than resources within the node. Here average cluster performance is near 3× that of cloud performance. The somewhat larger ratio

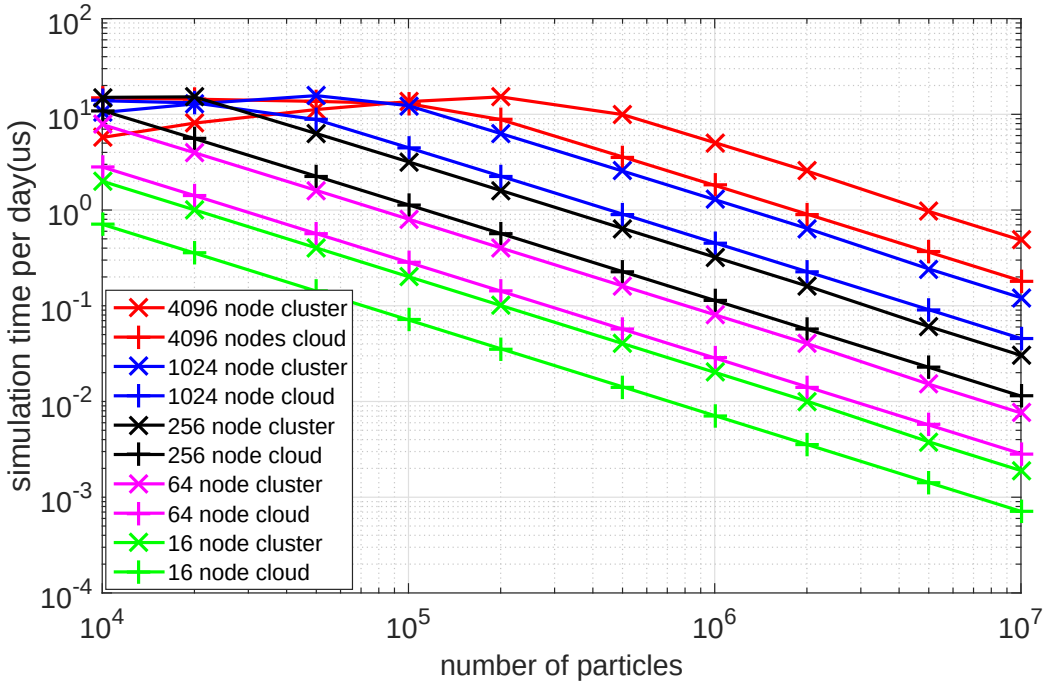


Figure 7·14: MD performance for *uniform* node design

for this design reflects the better ability of the cluster to build subnetworks. Strong scaling numbers are similar to those of the *uniform* design.

We compare the two designs on FPGA clusters and clouds. On average, the *specialized* design is 13% faster on FPGA clusters and 3% faster on clouds. The main reason is that *specialized* uses resources more efficiently. In the *uniform* design, there must be at least one 1D FFT IP on each node. For small problem sizes, many FFT IPs stay idle. Also, in the *specialized* design, we contract the range-limited and long-range computations to two smaller networks, which results in shorter time-of-flight.

We now explore strong scaling in more detail. From Figures 7·14 and 7·15, we see that MD performance on FPGA clouds and clusters can maintain strong scaling as long as the number of particles per node is > 100: at that point performance is mainly bound by range-limited force computation. When particles per node < 100, it is not clear whether performance is bound by network bandwidth or latency. So we examine the performance bottlenecks for different problem sizes and number of

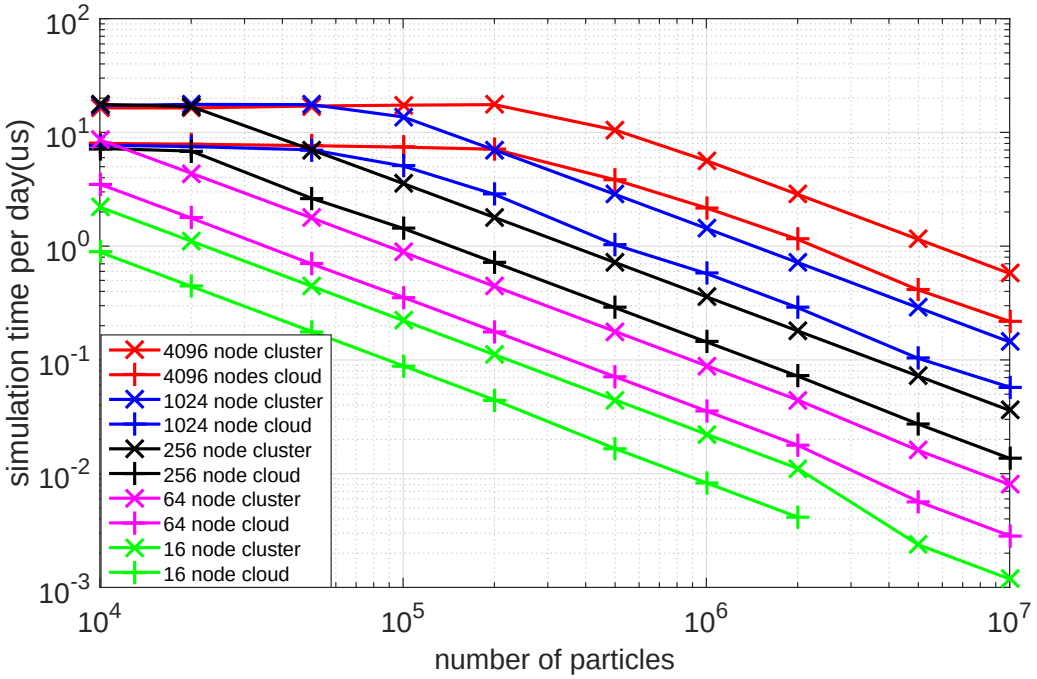


Figure 7·15: MD performance for *specialized* node design

nodes; results are plotted in Figure 7·16.

Figure 7·16 gives a roofline graph. When the number of particles per node is more than 100, MD performance on both FPGA cloud and cluster is bounded by the short range force computation time. When the number of particles per node is less than 100, MD performance on FPGA cloud is bound by the short-range force communication latency. On FPGA clusters, when the number of particles per node is less than 100, and the total number of particles is over 200K, MD performance is bounded by the communication latency of 3D FFT. When the number of particles per node is less than 100, and the total number of particles is fewer than 200K, MD performance is bounded by the range-limited communication latency, specifically by the effect of network bandwidth.

In Table 7.4, we compare the MD performance derived from this model with that of other technologies. It shows that MD performance on FPGA clusters and clouds is faster than state-of-the-art CPU and GPU performance by an order of magnitude

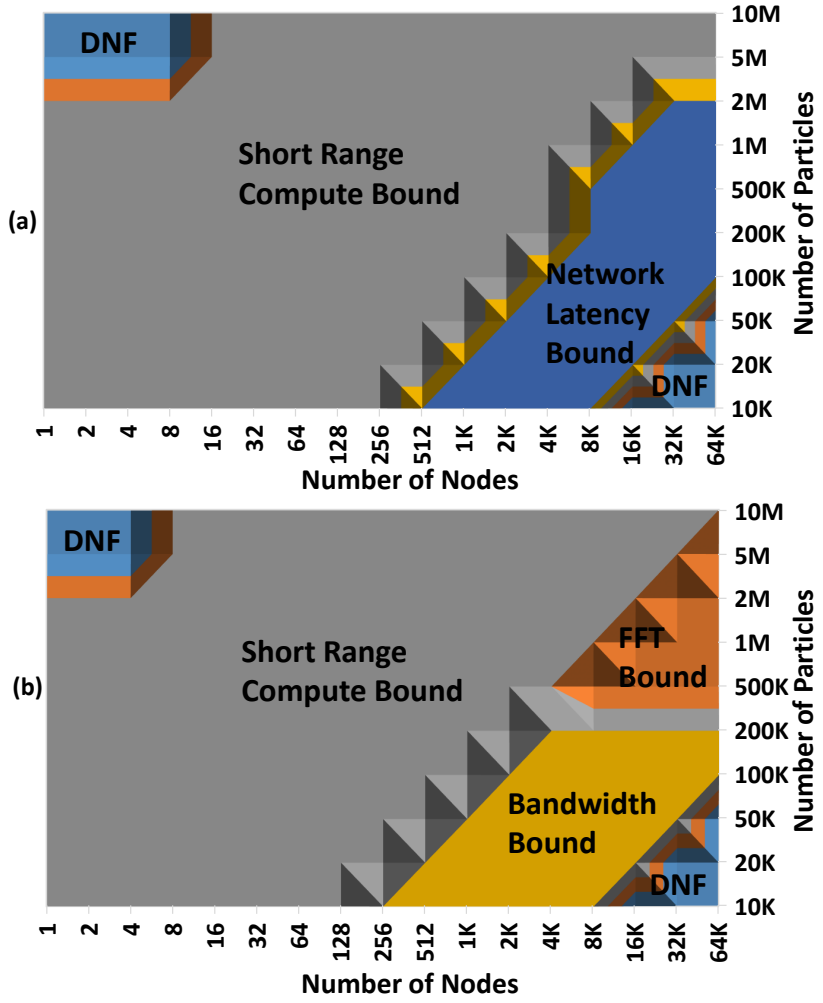


Figure 7·16: The MD performance bottleneck for (a) FPGA cloud and (b) FPGA cluster (DNF means do-not-fit)

in all cases. The performance of Anton 2 is shown for reference.

7.6 Discussion and Future Work

A well-known limitation of HPC in the cloud is the commodity network which tends not to perform well for communication-bound HPC applications. Because of their unique combination of capabilities, FPGA-enhanced clouds such as Catapult II offer a mechanism to address this limitation. In this study, we describe the implementation of a communication-bound application, the 3D FFT, on Catapult II. We believe this to

Table 7.4: The comparison of MD performance in μs/day of best FPGA clouds and clusters with the best of other technologies: (a) GROMACS on a Xeon E5-2690 processor with an NVIDIA GTX TITAN GPU (Lindahl, 2013), (b) Desmond on 1,024 cores of a Xeon E5430 cluster (Chow et al., 2008), (c) NAMD on 16,384 cores of Cray Jaguar XK6 (Sun et al., 2012). For FPGAs "(#)" denotes nodes.

N	FPGA cloud	FPGA cluster	Anton2	CPU or GPU
13K	14.7 (4K)	17.7 (256)	85.8	1.1 (a)
100K	13.9 (4K)	17.8 (2K)	59.4	0.29(b)
1M	4.3 (8K)	2.9 (2K)	9.5	0.035 (c)

be the first such implementation. We find that performance is competitive with, if not superior to, HPC clusters with dedicated networks. We also describe implementations of the 3D FFT on FPGA clusters. Another result introduces the idea of phased elasticity. When applied to FFTs within MD we note performance benefits averaging 16% and 29%. Finally, we present two MD designs and show through an empirically derived performance model that MD on FPGA clouds is also likely to be competitive with MD run on HPC clusters.

We have reason to be confident in the quality of our models. They are derived from post-place-and-route timing and area results which are generally within a few percent of the implementation. We note that some of the comparisons are with somewhat dated technology. But we also note that the FPGA technology used is similarly dated, if not more so. For this reason, we have tried to be careful with our conclusions.

Chapter 8

Conclusion and Future Work

In this chapter, we first state our conclusion and then propose potential future work.

8.1 Conclusion

In this dissertation, we have explored several aspects of building communication infrastructure for FPGA-centric cluster. We first investigate jitter of MGT link latency and clocks of multiple FPGAs; we then evaluate their impact on performance. Our results show that the communication infrastructure based on MGTs and its IP is sufficient to make these jitters manageable. It requires only simple flow control and little or no additional buffering on the application side. We can assume a near-synchronous model of multi-FPGA communication, which simplifies the statically scheduled routing design.

We also propose two different router microarchitectures for workloads with dynamic and static communication patterns, respectively. The first router is based on the classical wormhole VC-based router (Dally and Towles, 2004). We modify as determined by hardware constraints of FPGA clusters. There are four significant differences between our router and the canonical router. First, we use a flit size which is smaller than the phit size to meet the timing requirement. Second, we have multiple injection and ejection ports, which offers maximum injection rates. Third, we connect VCs to the switch directly, which reduces the chances of packet blocking. Fourth, we have a reduction-tree-based scalable switch design, which can meet the

timing requirement. The second router microarchitecture is the statically-scheduled collective router. This router is a Virtual Output Queue (VOQ) router with hardware support for multicast and reduction operations. It uses routing tables to enable statically-scheduled routing.

For both routers we implement a sets of routing algorithms and arbitration policies. For the VC-based router, we currently support five of the most commonly used routing algorithms and three switch arbitration policies. Besides that, we designed a cycle-accurate simulator to simulate the routers and the entire Novo-G# network based on the routers. We also proposed a framework to search for the optimal router configuration among the fifteen possible configurations. Our results show that in 8^3 torus, the optimal configurations can improve batch latency by 23%, average latency by 6%, worst-case latency by 34%, throughput by 17%, and the area consumption of the router by 30% in average. Our results also show that our application-aware router configurations reduce batch latency by 6%, average latency by 2%, worst-case latency by 21%, router area size by 18%, and improve throughput by 10%. For the statically-scheduled collective acceleration router, we developed an offline collective routing (OCR) algorithm. Compared with a state-of-the-art online routing algorithm, our algorithm reduces the latency of multicast by 15% and of reduction by 4%. We also find that *a priori* knowledge of communication can be used to reduce buffer sizes and enable higher bandwidth utilization.

We applied our communication infrastructure to the 3D FFT. The experimental results demonstrate that communication overhead does not overwhelm the calculation and that performance of our design is faster than performance on CPUs and GPUs by at least one order of magnitude (achieving strong scaling for the target applications through large cluster sizes). Also, the FPGA cluster performance is similar to that of Anton (Young et al., 2009). For the 64^3 FFT, the presented design is faster than

Anton (Young et al., 2009) by about 30%.

We also implemented the 3D FFT on a second multi-FPGA platform: the Microsoft Catapult II FPGA cloud (Caulfield et al., 2016). We described the implementation of the 3D FFT on Catapult II. We believe this to be the first such implementation. We find that performance is competitive with, if not superior to, HPC clusters with dedicated networks. Another result introduces the idea of phased elasticity. When applied to FFTs within MD, we observe performance benefits averaging 16% and 29% for two different scenarios. Finally, we present two MD designs and show through an empirically derived performance model that MD on FPGA clouds is also likely to be competitive with MD run on HPC clusters. Our modeled results demonstrate that MD on FPGA clusters/clouds can achieve more than one order of magnitude advantage than CPUs and GPUs, and can fill in the gap between ASIC and CPUs/GPUs.

8.2 Future Work

Recall our framework of the four levels of work in constructing state-of-the-art of FPGA clusters. Overall, the future work encompasses what remains in making the Novo-G# a usable high performance cluster, plus finishing particular research investigations already described.

8.2.1 Future Work on Inter-FPGA Links

Currently, the two link protocols (SerialLite III and Interlaken) still have high latency variance. If we would like to achieve more efficient statically-scheduled routing, we would like the latency to be more deterministic and predictable. (Liu et al., 2014; Giordano and Aloisio, 2011; Giordano and Aloisio, 2012) have researched how to implement inter-FPGA links with small variance. However, they used different FPGA

devices, and much additional effort is needed to port their solutions to our FPGA cluster.

8.2.2 Future Work on Inter-FPGA Communication Middleware

So far the work on middleware has been limited to the framework that searches for the optimal router configuration. It is still incomplete and inconvenient. The goal of the remaining work on middleware is to benefit programmability and performance at the same time. We have come up with a step-by-step plan to implement this middleware.

- Implement a basic wrapper that hides all the status signals of MGT IP from user applications. This basic wrapper should just expose a simple FIFO interface.
- Unify the two router microarchitectures into a single design. This is straightforward because they have similar pipelined architectures. Integrate the unified router into the basic wrapper.
- Extend the router specification framework to support collective workloads. Part of this involves finding a more better selection algorithm than the current exhaustive search. The cycle-accurate simulator needs to be implemented in a multi-threaded version.
- Integrate the DRAM controller into the wrapper and also encapsulate its r/w interface into a simple FIFO-like interface.
- Connect this wrapper to the PCIe controller so that application users could directly access network and DRAM by APIs from CPU side. Also, application developers could load routing tables using API as well.
- Design an autotuner with GUI that automates the entire flow. It accepts three kinds of configuration specifications from users: the type of application, the

number of nodes, and the size of the application. This program could generate an entire set of codes to run the application including HDL code for the wrapper and the application logic, Quartus project files to compile this project, the software code that contains APIs to execute this application, and the optimal router configuration. This program should be able to support several applications that have been well-studied.

Upon completion, our middleware would truly benefit programmability and performance at the same time.

8.2.3 Future Work on Routing Algorithms

We have proposed and implemented an offline collective routing algorithm. But for unicast operations, we currently use the existing algorithms from previous studies (Kinsy et al., 2009). A statically-scheduled congestion-free routing algorithm is also needed for unicast operations.

In our OCR algorithm, we mainly focused on the optimization of the routing paths. However, there is another dimension we have not explored, which is the ordering of packets. In (Kapre, 2016), the algorithm delays packets to reduce network congestion. But they do not utilize any application knowledge, and their routing algorithms are as simple as DOR.

8.2.4 Future Work on Applications

We will apply our work to other applications besides the 3D FFT. One important application is other parts of the Molecular Dynamics simulations. MD simulation has two main kinds of communication pattern. One is the all-to-all communication pattern like the 3D FFT. A second is the collective operations among neighboring nodes. There are many trade-offs worth exploration to accelerate simultaneously the two kinds of patterns. Besides MD, other applications are also throttled by

communication (Asanovic et al., 2006). An advantage of our middleware is that we could always switch to the best routing configurations for different applications because of the inherent reconfigurability of FPGA.

References

Abad, P., Puente, V., and Gregorio, J. (2009). Mrr: Enabling fully adaptive multicast routing for cmp interconnection networks. In *Proceedings of the International Symposium on High-Performance Computer Architecture (HPCA)*, pages 355–366. IEEE.

Abdellah, M., Saleh, S., Eldeib, A., and Shaarawi, A. (2012). High performance multi-dimensional (2D/3D) FFT-shift implementation on graphics processing units (GPUs). In *Proceedings of the Cairo International Biomedical Engineering Conference*, pages 171–174. IEEE.

Abdellah-Medjadji, K., Senouci, B., and Petrot, F. (2008). Large Scale On-Chip Networks : An Accurate Multi-FPGA Emulation Platform. In *Proceedings of the EUROMICRO Conference on Digital System Design Architectures, Methods and Tools (DSD)*, pages 3–9.

Agarwal, A., Iskander, C., and Shankar, R. (2009). Survey of network on chip (noc) architectures & contributions. *Journal of engineering, Computing and Architecture*, 3(1):21–27.

Agne, A., Happe, M., Keller, A., Lubbers, E., Plattner, B., Platzner, M., and Plessl, C. (2014). Reconos: An operating system approach for reconfigurable computing. *IEEE Micro*, 34:60–71.

Alam, S. R., Agarwal, P. K., Smith, M. C., Vetter, J. S., and Caliga, D. (2007). Using fpga devices to accelerate biomolecular simulations. *Computer*, 40(3).

Altera (2013). *SerialLite III Streaming MegaCore Function User Guide*. Altera.

Altera (2014a). *FFT MegaCore Function: User Guide*. Altera.

Altera (2014b). *Stratix V Device Handbook volume2: Transceivers*. Altera.

Altera (2015). *Altera Transceiver PHY IP Core User Guide*. Altera.

Anson, H., Thomas, D., Tsoi, K., and Luk, W. (2010). Dynamic scheduling monte-carlo framework for multi-accelerator heterogeneous clusters. In *Proceedings of the International Conference on Field-Programmable Technology (FPT)*, pages 233–240. IEEE.

Arista (2013). *Arista Products*. Arista Networks, Inc.

Asanovic, K., Bodik, R., Catanzaro, B., Gebis, J., Husbands, P., Keutzer, K., Patterson, D., Plishker, W., Shalf, J., Williams, S., and et. al. (2006). The landscape of parallel computing research: A view from berkeley. Technical report, Technical Report UCB/EECS-2006-183, EECS Department, University of California, Berkeley.

Babb, J., Tessier., R., and Agarwal, A. (1993). Virtual wires: Overcoming pin limitations in fpga-based logic emulators. In *Proceedings of the IEEE Workshop on FPGAs for Custom Computing Machines (FCCM)*, pages 142–151. IEEE.

Badr, H. and Podar, S. (1989). An optimal shortest-path routing policy for network computers with regular mesh-connected topologies. *IEEE Transactions on Computers*, 38(10):1362–1371.

Baxter, R., Booth, S., Bull, M., and et al. (2007). Maxwell - a 64 FPGA Supercomputer. In *Proceedings of the NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, pages 287–294.

Birrittella, M. S., Debbage, M., Huggahalli, R., Kunz, J., Lovett, T., Rimmer, T., Underwood, K. D., and Zak, R. C. (2015). Intel omni-path architecture: Enabling scalable, high performance fabrics. In *IEEE Symposium on High-Performance Interconnects (HOTI)*, pages 1–9. IEEE.

Bjerregaard, T. and Mahadevan, S. (2006). A survey of research and practices of network-on-chip. *ACM Computing Surveys (CSUR)*, 38(1):1.

Borkar, S., Cohn, R., Cox, G., Gross, T., Kung, H., Lam, M., Levine, M., Moore, B., Moore, W., Peterson, C., and et. al. (1990). Supporting systolic and memory communication in iwarp. In *Proceedings of the ACM/IEEE International Symposium on Computer Architecture (ISCA)*.

Brebner, G. and Levi, D. (2003). Networking on chip with platform fpgas. In *Proceedings of the International Conference on Field-Programmable Technology (FPT)*, pages 13–20. IEEE.

Bunker, T. and Swanson, S. (2013). Latency-Optimized Networks for Clustering FPGAs. In *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 129–136.

Caulfield, A., Chung, E., Putnam, A., and et al. (2016). A cloud-scale acceleration architecture. In *Proceedings of the IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 13–24.

Chang, C., Kuusilinna, K., Richards, B., and Brodersen, R. (2003). Implementation of bee: A real-time large-scale hardware emulation engine. In *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA)*, FPGA '03, pages 91–99, New York, NY, USA. ACM.

Chang, C., Wawrzynek, J., and Brodersen, R. (2005). Bee2: A high-end reconfigurable computing system. *IEEE Design and Test of Computers*, 2005:114–125.

Chen, D., Eisley, N. A., Heidelberger, P., Senger, R. M., Sugawara, Y., Kumar, S., Salapura, V., Satterfield, D. L., Steinmacher-Burow, B., and Parker, J. J. (2011). The ibm blue gene/q interconnection network and message unit. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, pages 1–10. IEEE.

Chiu, M., Herbordt, M., and Langhammer, M. (2008). Performance potential of molecular dynamics simulations on high performance reconfigurable computing systems. In *Proceedings of the High Performance Reconfigurable Technology and Applications*.

Chiu, M. and Herbordt, M. C. (2009). Efficient particle-pair filtering for acceleration of molecular dynamics simulation. In *Proceedings of the International Conference on Field Programmable Logic and Applications (FPL)*, pages 345–352. IEEE.

Chiu, M. and Herbordt, M. C. (2010). Molecular dynamics simulations on high-performance reconfigurable computing systems. *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, 3(4):23.

Chiu, M., Khan, M. A., and Herbordt, M. C. (2011). Efficient calculation of pairwise nonbonded forces. In *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 73–76. IEEE.

Chow, E., Rendleman, C., Bowers, K., Dror, R., Hughes, D., Gullingsrud, J., Sacerdoti, F., and Shaw, D. (2008). Desmond performance on a cluster of multicore processors. *DE Shaw Research DESRES/TR–2008-01*.

Chung, E., Hoe, J., and Mai, K. (2011). CoRAM: an In-Fabric Memory Architecture for FPGA-based Computing. In *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA)*, pages 97–106.

Cortina and Cisco (2008). *Interlaken Protocol Definition v1.2*. Cortina Systems Inc. and Cisco Systems Inc.

Craciun, S., Wang, G., George, A., Lam, H., and Principe, J. (2013). A scalable rc architecture for mean-shift clustering. In *Proceedings of the IEEE International Conference on Application-Specific Systems, Architectures and Processors*, pages 370–374. IEEE.

Dally, W. (1992). Virtual-channel flow control. *IEEE transactions on Parallel and Distributed Systems (TPDS)*, 3(2):194–205.

Dally, W. and Aoki, H. (1993). Deadlock-free adaptive routing in multicomputer networks using virtual channels. *IEEE transactions on Parallel and Distributed Systems (TPDS)*, 4(4):466–475.

Dally, W. and Seitz, C. (1986). The torus routing chip. *Distributed computing*, 1(4):187–196.

Dally, W. and Seitz, C. (1987). Deadlock-free message routing in multiprocessor interconnection networks. *IEEE Transactions on computers*, 100(5):547–553.

Dally, W. and Towles, B. (2004). *Principles and practices of interconnection networks*. Elsevier.

Davis, J., Thacker, C., Chang, C., and Thacker, C. (2009). Bee3: Revitalizing computer architecture research. Technical report, Microsoft Research.

Duato, J., Yalamanchili, S., and Ni, L. M. (2003). *Interconnection networks: an engineering approach*. Morgan Kaufmann.

Ebrahimi, M., Daneshtalab, M., Liljeberg, P., Plosila, J., Flich, J., and Tenhunen, H. (2014). Path-based partitioning methods for 3d networks-on-chip with minimal adaptive routing. *IEEE Transactions on Computers*, 63(3):718–733.

Feng, T.-y. (1981). A survey of interconnection networks. *Computer*, 14(12):12–27.

Feng, W. and Shin, K. (1997). Impact of selection functions on routing algorithm performance in multicomputer networks. In *Proceedings of International Conference on Supercomputing (SC)*, pages 132–139. ACM.

Fleming, K., Adler, M., Pellauer, M., Parashar, A., Mithal, A., and Emer, J. (2012). Leveraging latency-insensitivity to ease multiple fpga design. In *Proceedings of the ACM/SIGDA international symposium on Field Programmable Gate Arrays (FPGA)*, pages 175–184. ACM.

Fleming, K., Yang, H., Adler, M., and Emer, J. (2014). The leap fpga operating system. In *Proceedings of the International Conference on Field Programmable Logic and Applications (FPL)*, pages 1–8. IEEE.

Gao, S., Schmidt, A., and Sass, R. (2009). Hardware implementation of mpi_barrier on an fpga cluster. In *Proceedings of the International Conference on Field Programmable Logic and Applications (FPL)*, pages 12–17. IEEE.

Gao, S., Schmidt, A., and Sass, R. (2010). Impact of reconfigurable hardware on accelerating mpi_reduce. In *Proceedings of the International Conference on Field-Programmable Technology (FPT)*, pages 29–36. IEEE.

George, A., Herbordt, M., Lam, H., Lawande, A., Sheng, J., and Yang, C. (2016). Novo-G#: Large-Scale Reconfigurable Computing with Direct and Programmable

Interconnects. In *Proceedings of the IEEE High Performance Extreme Computing Conference (HPEC)*.

George, A., Lam, H., and Stitt, G. (2011). Novo-g: At the forefront of scalable reconfigurable supercomputing. *Computing in Science and Engineering*, 13(1):82–86.

Gidel (2016). *Gidel Products*. Gidel.

Giordano, R. and Aloisio, A. (2011). Fixed-latency, multi-gigabit serial links with xilinx fpgas. *IEEE Transactions on Nuclear Science*, 58(1):194–201.

Giordano, R. and Aloisio, A. (2012). Protocol-Independent, Fixed Latency Links with FPGA-Embedded SerDeses. *Journal of Instrumentation*, 7(1).

Glass, C. and Ni, L. (1992). The turn model for adaptive routing. In *Proceedings of the International Symposium on Computer Architecture (ISCA)*, pages 278–287. IEEE.

Glass, C. and Ni, L. (1994). The turn model for adaptive routing. *Journal of the ACM (JACM)*, 41(5):874–902.

Grossman, J., Towles, B., Greskamp, B., and Shaw, D. (2015). Filtering, reductions and synchronization in the anton 2 network. In *Proceedings of the IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 860–870. IEEE.

Gu, Y. and Herbordt, M. C. (2007). Fpga-based multigrid computation for molecular dynamics simulations. In *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 117–126. IEEE.

Gu, Y., VanCourt, T., and Herbordt, M. C. (2006a). Accelerating molecular dynamics simulations with configurable circuits. *IEEE Proceedings-Computers and Digital Techniques*, 153(3):189–195.

Gu, Y., VanCourt, T., and Herbordt, M. C. (2006b). Improved interpolation and system integration for FPGA-based molecular dynamics simulations. In *Proceedings of the International Conference on Field Programmable Logic and Applications (FPL)*, pages 1–8. IEEE.

Gu, Y., VanCourt, T., and Herbordt, M. C. (2006c). Integrating FPGA acceleration into the protomol molecular dynamics code: Preliminary report. In *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 315–316. IEEE.

Gu, Y., VanCourt, T., and Herbordt, M. C. (2008). Explicit design of FPGA-based coprocessors for short-range force computation in molecular dynamics simulations. *Parallel Computing*, 34(4-5):261–271.

Guarrasi, M., Erbacci, G., and Emerson, A. (2014). Auto-tuning of the fftw library for massively parallel supercomputers. *PRACE white paper*, 1.

Guneysu, T., Kasper, T., Novotny, M., Paar, C., and Rupp, A. (2008). Cryptanalysis with copacobana. *IEEE Transactions on Computers (TC)*, 2008:1498–1513.

Hauck, S. (1995). *Multi-FPGA Systems*. PhD thesis, University of Washington, Seattle, WA, USA. UMI Order No. GAX96-16615.

Hauck, S., Borriello, G., and Ebeling, C. (1998). Mesh routing topologies for multi-fpga systems. *IEEE Transactions on Very Large Scale Integration (TVLSI) Systems*, 6(3):400–408.

Herbordt, M. and Swarztrauber, P. (2003). Towards Scalable Multicomputer Communication through Offline Routing. Technical Report TR2003-01, ECE Dept, Boston University.

Herbordt, M. C. (2013). Architecture/algorithm codesign of molecular dynamics processors. In *Proceedings of the Asilomar Conference on Signals, Systems and Computers*, pages 1442–1446. IEEE.

Herbordt, M. C., Gu, Y., VanCourt, T., Model, J., Sukhwani, B., and Chiu, M. (2008). Computing models for fpga-based accelerators. *Computing in Science & Engineering*, 10(6):35–45.

Herbordt, M. C., VanCourt, T., Gu, Y., Sukhwani, B., Conti, A., Model, J., and DiSabello, D. (2007). Achieving high performance with FPGA-based computing. *IEEE Computer*, 40(3):42–49.

Hluchyj, M. G. and Karol, M. J. (1988). Queueing in high-performance packet switching. *IEEE Journal on selected Areas in Communications*, 6(9):1587–1597.

Huan, Y. and DeHon, A. (2012). Fpga optimized packet-switched noc using split and merge primitives. In *Proceedings of the International Conference on Field-Programmable Technology (FPT)*, pages 47–52. IEEE.

Humphries, B. (2013). Using Offline Routing to Implement a Low Latency 3D FFT in a Multinode FPGA System. Master's thesis, Department of Electrical and Computer Engineering, Boston University.

Humphries, B., Zhang, H., Sheng, J., Landaverde, R., and Herbordt, M. C. (2014). 3d ffts on a single fpga. In *Proceedings of the IEEE International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 68–71. IEEE.

Intel (2013). *Intel Xeon Phi System Software Developer's Guide*. Intel.

Jerger, N., Peh, L., and Lipasti, M. (2008). Virtual circuit tree multicasting: A case for on-chip hardware multicast support. In *Proceedings of the International Symposium on Computer Architecture (ISCA)*, pages 229–240. IEEE.

Jun, S., Liu, M., Xu, S., and Arvind (2015). A Transport-Layer Network for Distributed FPGA Platforms. In *Proceedings of the International Conference on Field Programmable Logic and Applications (FPL)*, pages 1–4.

Jung, J., Kobayashi, C., Imamura, T., and Sugita, Y. (2016). Parallel Implementation of 3D FFT with Volumetric Decomposition Schemes for Efficient Molecular Dynamics Simulations. *Computer Physics Communications*, 200:57–65.

Kapre, N. (2016). Marathon: Statically-scheduled conflict-free routing on fpga overlay nocs. In *Proceedings of the IEEE International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 156–163. IEEE.

Kapre, N. and Gray, J. (2015). Hoplite: Building austere overlay nocs for fpgas. In *Proceedings of the International Conference on Field Programmable Logic and Applications (FPL)*, pages 1–8. IEEE.

Kapre, N. and Gray, J. (2017). Hoplite: A deflection-routed directional torus noc for fpgas. *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, 10(2):14.

Kapre, N., Mehta, N., Rubin, R., Barnor, H., Wilson, M., Wrighton, M., and DeHon, A. (2006). Packet switched vs. time multiplexed fpga overlay networks. In *Proceedings of the International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 205–216. IEEE.

Karol, M., Hluchyj, M., and Morgan, S. (1987). Input versus output queueing on a space-division packet switch. *IEEE Transactions on communications*, 35(12):1347–1356.

Katchalski-Katzir, E., Shariv, I., Eisenstein, M., Friesem, A., Aflalo, C., and Vakser, I. (1992). Molecular surface recognition: Determination of geometric fit between proteins and their ligands by correlation techniques. *Proceedings of the National Academy of Sciences*, 89(6):2195–2199.

Khalid, M. (1999). *Routing Architecture and Layout Synthesis for Multi-FPGA Systems*. PhD thesis, Department of Electrical and Computer Engineering, University of Toronto.

Khan, M. and Herbordt, M. (2012). Communication requirements for fpga-centric molecular dynamics. In *Proceedings of the Symposium on Application Accelerators for High Performance Computing (SAAHPC)*.

Khan, M. A., Chiu, M., and Herbordt, M. C. (2013). Fpga-accelerated molecular dynamics. In Benkrid, K. and Vanderbauwhede, W., editors, *High Performance Computing Using FPGAs*, pages 105–135. Springer Verlag.

Khan, M. A. and Herbordt, M. C. (2011). Parallel discrete molecular dynamics simulation with speculation and in-order commitment. *Journal of computational physics*, 230(17):6563–6582.

Kim, D., Trzasko, J., Smelyanskiy, M., Haider, C., Dubey, P., and Manduca, A. (2011). High-performance 3d compressive sensing mri reconstruction using many-core architectures. *Journal of Biomedical Imaging*, 2011:2.

Kim, J., Balfour, J., and Dally, W. (2007). Flattened butterfly topology for on-chip networks. In *Proceedings of the IEEE/ACM International Symposium on*

Microarchitecture (MICRO), pages 172–182. IEEE Computer Society.

Kim, J., Dally, W.and Towles, B., and Gupta, A. (2005a). Microarchitecture of a high-radix router. *ACM SIGARCH Computer Architecture News*, 33(2):420–431.

Kim, J., Nicopoulos, C., and Park, D. (2006). A gracefully degrading and energy-efficient modular router architecture for on-chip networks. In *Proceedings of the International Symposium on Computer Architecture (ISCA)*, pages 4–15. IEEE.

Kim, J., Park, D., Theocharides, T., Vijaykrishnan, N., and Das, C. (2005b). A low latency router supporting adaptivity for on-chip interconnects. In *Proceedings of the Design Automation Conference (DAC)*, pages 559–564. ACM.

Kinsy, M. A., Cho, M. H., Wen, T., Suh, E., Van Dijk, M., and Devadas, S. (2009). Application-aware deadlock-free oblivious routing. In *Proceedings of the International Symposium on Computer Architecture (ISCA)*, pages 208–219. IEEE.

Kinsy, M. A., Pellauer, M., and Devadas, S. (2013). Heracles: A tool for fast rtl-based design space exploration of multicore processors. In *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA)*, pages 125–134. ACM.

Kono, Y., Sano, K., and Yamamoto, S. (2012). Scalability Analysis of Tightly-Coupled FPGA-Cluster for Lattice Boltzmann Computation. In *Proceedings of the International Conference on Field Programmable Logic and Applications (FPL)*, pages 120–127.

Krasnov, A., Schultz, A., Wawrzynek, J., Gibeling, G., and Droz, P. (2007). RAMP BLUE: A Message-Passing Manycore System in FPGAs. In *Proceedings of the International Conference on Field Programmable Logic and Applications (FPL)*, pages 54–61.

Krishna, T. and Peh, L. (2014). Single-cycle collective communication over a shared network fabric. In *Proceedings of the International Symposium on Networks-on-Chip (NOCS)*, pages 1–8. IEEE.

Krishna, T., Peh, L., Beckmann, B., and Reinhardt, S. (2011). Towards the ideal on-chip fabric for 1-to-many and many-to-1 communication. In *Proceedings of the IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 71–82. ACM.

Kritikos, W., Schmidt, A., Sass, R., Anderson, E., and French, M. (2012). Redsharc: a programming model and on-chip network for multi-core systems on a programmable chip. *International Journal of Reconfigurable Computing (IJRC)*, 2012:1–11.

Kumar, A., Peh, L.-S., Kundu, P., and Jha, N. K. (2007). Express virtual channels: towards the ideal interconnection fabric. *ACM SIGARCH Computer Architecture News*, 35(2):150–161.

Kung, H. (1988). Systolic communication. In *Proceedings of the International Conference on Systolic Arrays*, pages 695–703. IEEE.

Laffely, A., Liang, J., Jain, P., Burleson, W., and Tessier, R. (2001). Adaptive systems on a chip (asoc) for low-power signal processing. In *Proceedings of the Asilomar Conference on Signals, Systems and Computers*, volume 2, pages 1217–1221. IEEE.

Lam, B., Pascoe, C., Schaecher, S., Lam, H., and George, A. (2013). Bsw: Fpga-accelerated blast-wrapped smith-waterman aligner. In *Proceedings of the International Conference on Reconfigurable Computing and FPGAs (ReConFig)*, pages 1–7. IEEE.

Lan, H. (1995). *Architecture and Computer-aided Design Tools for a Field programmable Multi-chip Module*. PhD thesis, Stanford University, Stanford, CA, USA. AAI9602912.

Lawande, A. G., George, A. D., and Lam, H. (2016). Novo-g#: a multidimensional torus-based reconfigurable cluster for molecular dynamics. *Concurrency and Computation: Practice and Experience*, 28(8):2374–2393.

Lee, V. W., Kim, C., Chhugani, J., Deisher, M., Kim, D., Nguyen, A. D., Satish, N., Smelyanskiy, M., Chennupaty, S., Hammarlund, P., et al. (2010). Debunking the 100x gpu vs. cpu myth: an evaluation of throughput computing on cpu and gpu. *ACM SIGARCH computer architecture news*, 38(3):451–460.

Leighton, F. T. (2014). *Introduction to parallel algorithms and architectures: Arrays· trees· hypercubes*. Elsevier.

Liang, C., Wu, C., Zhou, X., Cao, W., Wang, S., and Wang, L. (2013). An fpga-cluster-accelerated match engine for content-based image retrieval. In *Proceedings of the International Conference on Field-Programmable Technology (FPT)*, pages 422–425.

Lindahl, E. (2013). Evolutions & Revolutions in Peta- and Exascale Biomolecular Simulation. In *Proceedings of the Conference on Scientific Computing*.

Liu, X., Deng, Q., and Wang, Z. (2014). Design and FPGA Implementation of High-Speed, Fixed-Latency Serial Transceivers. *IEEE Transactions on Nuclear Science*, 61(1).

Mahr, P., Lorchner, C., Ishebabi, H., and Bobda, C. (2008). SoC-MPI: A Flexible Message Passing Library for Multiprocessor Systems-on-Chips. In *Proceedings of*

the International Conference on Reconfigurable Computing and FPGAs (ReConFig), pages 187–192.

Marescaux, T., Nollet, V., Mignolet, J.-Y., Bartic, A., Moffat, W., Avasare, P., Coene, P., Verkest, D., Vernalde, S., and Lauwereins, R. (2004). Run-time support for heterogeneous multitasking on reconfigurable socs. *Integration, the VLSI journal*, 38(1):107–130.

Markettos, A., Fox, P., Moore, S., and Moore, A. (2014). Interconnect for Commodity FPGA Clusters: Standardized or Customized? In *Proceedings of the International Conference on Field Programmable Logic and Applications (FPL)*, pages 1–8.

Matsutani, H., Koibuchi, M., Amano, H., and Yoshinaga, T. (2009). Prediction router: Yet another low latency on-chip router architecture. In *Proceedings of the International Symposium on High-Performance Computer Architecture (HPCA)*, pages 367–378. IEEE.

Matthey, T., Cickovski, T., Hampton, S., Ko, A., Ma, Q., Nyerges, M., Raeder, T., Slabach, T., and Izaguirre, J. A. (2004). Protomol, an object-oriented framework for prototyping novel algorithms for molecular dynamics. *ACM Transactions on Mathematical Software (TOMS)*, 30(3):237–265.

Mencer, O., Tsoi, K., Craimer, S., T.Todman, Luk, W., Wong, M., and Leong, P. (2009). Cube: A 512-fpga cluster. In *Proceedings of the Southern Conference on Programmable Logic (SPL)*, pages 1–3.

Meng, J., Llamosí, E., Kaplan, F., Zhang, C., Sheng, J., Herbordt, M., Schirner, G., and Coskun, A. K. (2016). Communication and cooling aware job allocation in data centers for communication-intensive workloads. *Journal of Parallel and Distributed Computing (JPDC)*, 96:181–193.

Moore, S., Fox, P., Marsh, S., Markettos, A., and Mujumdar, A. (2012). Bluehive-a field-programable custom computing machine for extreme-scale real-time neural network simulation. In *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 133–140. IEEE.

Moorthy, P. and Kapre, N. (2015). Zedwulf: Power-performance tradeoffs of a 32-node zynq soc cluster. In *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 68–75. IEEE.

Mullins, R., West, A., and Moore, S. (2004). Low-latency virtual-channel routers for on-chip networks. *ACM SIGARCH Computer Architecture News*, 32(2):188.

Mullins, R., West, A., and Moore, S. (2006). The design and implementation of a low-latency on-chip network. In *Proceedings of the Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 164–169. IEEE Press.

Murali, S., Atienza, D., Benini, L., and De Micheli, G. (2007). A method for routing packets across multiple paths in nocs with in-order delivery and fault-tolerance gaurantees. *VLSi DeSign*, 2007.

Nesson, T. and Johnsson, S. (1995). ROMM routing on mesh and torus networks. In *Proceedings of the ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, pages 275–286.

Nidhi, U., Paul, K., Hemani, A., and Kumar, A. (2013). High performance 3d-fft implementation. In *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 2227–2230. IEEE.

Papamichael, M. and Hoe, J. (2012). Connect: Re-examining conventional wisdom for designing nocs in the context of fpgas. In *Proceedings of the ACM/SIGDA*

International Symposium on Field Programmable Gate Arrays (FPGA), pages 37–46. ACM.

Patel, A., Madill, C., Saldana, M., Comis, C., Pomes, R., and Chowl, P. (2006). A Scalable FPGA-Based Multiprocessor. In *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 111–120.

Peh, L. and Dally, W. (2001). A delay model and speculative architecture for pipelined routers. In *Proceedings of the International Symposium on High Performance Computer Architecture (HPCA)*, pages 255–266. IEEE.

Peng, Y., Saldana, M., Madill, C., Zou, X., and Chow, P. (2014). Benefits of adding hardware support for broadcast and reduce operations in mpsoc applications. *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, 7(3):17.

Pham, D., Aipperspach, T., Boerstler, D., and et al. (2006). Overview of the architecture, circuit design, and physical implementation of a first-generation cell processor. *IEEE Journal of Solid-State Circuits (JSSC)*, 41(1):179–196.

Phillips, J., Areno, M., Rogers, C., Dasu, A., and Eames, B. (2007). A reconfigurable load balancing architecture for molecular dynamics. In *Proceedings of the IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 1–6. IEEE.

Phillips, J. C., Zheng, G., Kumar, S., and Kalé, L. V. (2002). Namd: Biomolecular simulation on thousands of processors. In *Proceedings of the ACM/IEEE Conference on Supercomputing*, pages 36–36. IEEE.

Pippig, M. (2013). Pfft: An extension of fftw to massively parallel architectures. *SIAM Journal on Scientific Computing*, 35(3):C213–C236.

Putnam, A., Caulfield, A., Chung, E., Chiou, D., and et al. (2014). A reconfigurable fabric for accelerating large-scale datacenter services. In *Proceeding of the ACM/IEEE International Symposium on Computer Architecuture (ISCA)*, pages 13–24.

Rodrigo, S., Flich, J., Duato, J., and Hummel, M. (2008). Efficient unicast and multicast support for cmps. In *Proceedings of the IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 364–375. IEEE Computer Society.

Saldana, M. and Chow, P. (2006). TMD-MPI: An MPI Implementation for Multiple Processors Across Multiple FPGAs. In *Proceedings of the International Conference on Field Programmable Logic and Applications (FPL)*, pages 1–6.

Samman, F. A., Hollstein, T., and Glesner, M. (2008). Multicast parallel pipeline router architecture for network-on-chip. In *Proceedings of the conference on Design, automation and test in Europe (DATE)*, pages 1396–1401. ACM.

Sano, K., Hatsuda, Y., and Yamamoto, S. (2014). Multi-fpga accelerator for scalable stencil computation with constant memory bandwidth. *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, 25(3):695–705.

Sass, R., Kritikos, W., Schmidt, A., Beeravolu, S., and et al. (2007). Reconfigurable computing cluster (rcc) project: Investigating the feasibility of fpga-based petascale computing. In *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 127–138.

Schmidt, A., Kritikos, W., Gao, S., and Sass, R. (2012). An evaluation of an integrated on-chip/off-chip network for high-performance reconfigurable computing. *International Journal of Reconfigurable Computing (IJRC)*, 2012.

Schmidt, A., Kritikos, W., Sharma, R., and Sass, R. (2009). Airen: A novel integration of on-chip and off-chip fpga networks. In *Proceedings of the IEEE International Symposium on Field Programmable Custom Computing Machines*, pages 271–274. IEEE.

Scrofano, R., Gokhale, M. B., Trouw, F., and Prasanna, V. K. (2008). Accelerating molecular dynamics simulations with reconfigurable computers. *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, 19(6):764–778.

Scrofano, R. and Prasanna, V. K. (2006). Preliminary investigation of advanced electrostatics in molecular dynamics on reconfigurable computers. In *Proceedings of the ACM/IEEE conference on Supercomputing*, page 90. ACM.

Seo, D., Ali, A., Lim, W., Rafique, N., and Thottethodi, M. (2005). Near-optimal worst-case throughput routing for two-dimensional mesh networks. In *Proceedings of the International Symposium on Computer Architecture (ISCA)*, pages 432–443. IEEE Computer Society.

Shaw, D., Deneroff, M., Dror, R., Kuskin, J., Larson, R., Salmon, J., Young, C., Batson, B., Bowers, K., Chao, J., and et. al. (2008). Anton, a special-purpose machine for molecular dynamics simulation. *Communications of the ACM*, 51(7):91–97.

Shaw, D., Grossman, J., Bank, J. A., Batson, B., Butts, J. A., Chao, J. C., Deneroff, M. M., Dror, R. O., Even, A., Fenton, C. H., et al. (2014). Anton 2: Raising the bar for performance and programmability in a special-purpose molecular dynamics supercomputer. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, pages 41–53. IEEE Press.

Shaw, D. E., Deneroff, M. M., Dror, R. O., Kuskin, J. S., Larson, R. H., Salmon,

J. K., Young, C., Batson, B., Bowers, K. J., Chao, J. C., et al. (2007). Anton, a special-purpose machine for molecular dynamics simulation. *ACM SIGARCH Computer Architecture News*, 35(2):1–12.

Shaw, P. and Milne, G. (1992). A highly parallel fpga-based machine and its formal verification. In *Proceedings of the International Workshop on Field Programmable Logic and Applications (FPL)*, pages 162–173. Springer.

Sheng, J., Humphries, B., Zhang, H., and Herbordt, M. (2014). Design of 3d ffts with fpga clusters. In *Proceedings of the High Performance Extreme Computing Conference (HPEC)*, pages 1–6. IEEE.

Sheng, J., Xiong, Q., Yang, C., and Herbordt, M. C. (2016). Application-aware collective communication. In *Proceedings of the IEEE International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 197–197. IEEE.

Sheng, J., Xiong, Q., Yang, C., and Herbordt, M. C. (2017). Collective communication on fpga clusters with static scheduling. *ACM SIGARCH Computer Architecture News*, 44(4):2–7.

Sheng, J., Yang, C., and Herbordt, M. (2015). Towards Low-Latency Communication on FPGA Clusters with 3D FFT Case Study. In *Proceedings of the International Symposium on Highly Efficient Accelerators and Reconfigurable Technologies (HEART)*.

Shoemaker, D., Honor, F., Metcalf, C., and Ward, S. (1996). Numesh: An architecture optimized for scheduled communication. *The Journal of Supercomputing*, 10(3):285–302.

Singh, A., Dally, W., Gupta, A., and Towles, B. (2003). Goal: a load-balanced adaptive routing algorithm for torus networks. *Proceedings of the International*

Symposium on Computer Architecture (ISCA), 31(2):194–205.

Singh, A., Dally, W., Towles, B., and Gupta, A. (2002). Locality-preserving randomized oblivious routing on torus networks. In *Proceedings of the ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, pages 9–13. ACM.

Skalicky, S., Schmidt, A., Lopez, S., and French, M. (2015). A Unified Hardware/Software MPSoC System Construction and Run-Time Framework. In *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE)*, pages 301–304.

So, H. and Brodersen, R. (2008). A unified hardware/software runtime environment for fpga-based reconfigurable computers using borph. *ACM Transactions on Embedded Computing Systems (TECS)*, 7:14:1–14:28.

Sridharan, R., Cooke, G., Hill, K., Lam, H., and George, A. (2012). Fpga-based reconfigurable computing for pricing multi-asset barrier options. In *Proceedings of the Symposium on Application Accelerators in High Performance Computing (SAAHPC)*, pages 34–43. IEEE.

StremDSP (2015). *Interlaken PHY Datasheet*. StremDSP LLC.

Sukhwani, B. and , M. C. (2008). Acceleration of a Production Rigid Molecule Docking Code. In *Proceedings of the International Conference on Field Programmable Logic and Applications (FPL)*, pages 341–346.

Sukhwani, B. and Herbordt, M. (2010). Fpga acceleration of rigid-molecule docking codes. *IET Computers & Digital Techniques*, 4(3):184–195.

Sullivan, H. and Bashkow, T. R. (1977). A large scale, homogeneous, fully distributed parallel machine, i. In *Proceedings of the International Symposium on Computer Architecture (ISCA)*, pages 105–117. ACM.

Sun, Y., Zheng, G., Mei, C., Bohm, E., Phillips, J., Kalé, L., and Jones, T. (2012). Optimizing Fine-Grained Communication in a Biomolecular Simulation Application on Cray XK6. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*.

Tamir, Y. and Frazier, G. (1988). High-performance multiqueue buffers for vlsi communication switches. In *Proceedings of the International Symposium on Computer Architecture (ISCA)*, pages 343–354. IEEE.

Tsoi, K. and Luk, W. (2010). Axel: A heterogeneous cluster with fpgas and gpus. In *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA)*, pages 115–124.

Unnikrishnan, D., Zhao, J., and Tessier, R. (2009). Application-Specific Customization and Scalability of Soft Multiprocessors. In *Proceedings of the IEEE International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 123–130.

Vancourt, T., Gu, Y., and Herbordt, M. C. (2004). FPGA acceleration of rigid molecule interactions. In *Proceedings of the International Conference on Field Programmable Logic and Applications (FPL)*.

VanCourt, T., Gu, Y., Mundada, V., and Herbordt, M. (2006). Rigid molecule docking: Fpga reconfiguration for alternative force laws. *EURASIP Journal on Advances in Signal Processing*, 2006(1):1–10.

Vancourt, T. and Herbordt, M. C. (2005). Three dimensional template correlation: Object recognition in 3D voxel data. In *Proceedings of Computer Architectures for Machine Perception*, pages 153–158.

Vancourt, T. and Herbordt, M. C. (2009). Elements of high-performance reconfigurable computing. *Advances in Computers*, 75:113–157.

Varma, B. S. C., Paul, K., and Balakrishnan, M. (2013). Accelerating 3d-fft using hard embedded blocks in fpgas. In *Proceedings of the International Conference on VLSI Design and International Conference on Embedded Systems*, pages 92–97. IEEE.

Wang, L., Jin, Y., Kim, H., and Kim, E. J. (2009). Recursive partitioning multicast: A bandwidth-efficient routing for networks-on-chip. In *Proceedings of the ACM/IEEE International Symposium on Networks-on-Chip (NOCS)*, pages 64–73. IEEE Computer Society.

Wang, X., Yang, M., Jiang, Y., and Liu, P. (2011). On an efficient noc multicasting scheme in support of multiple applications running on irregular sub-networks. *Microprocessors and Microsystems*, 35(2):119–129.

Wang, Y., Zhou, X., Wang, L., Yan, J., Luk, W., Peng, C., and Tong, J. (2013). Spread: A streaming-based partially reconfigurable architecture and programming model. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 21(12):2179–2192.

Wawrzynek, J., Patterson, D., Oskin, M., Lu, S., Kozyrakis, C., Hoe, J., Chiou, D., and Asanovic, K. (2007). Ramp: Research accelerator for multiple processors. *IEEE Micro*, 27(2):46–57.

Weisz, G. and Hoe, J. (2015). CoRAM++: Supporting Data-Structure-Specific Memory Interfaces for FPGA Computing. In *Proceedings of the International Conference on Field Programmable Logic and Applications (FPL)*, pages 1–8.

Wolinski, C., Trouw, F. R., and Gokhale, M. (2003). A preliminary study of molecular dynamics on reconfigurable computers. Technical report, Los Alamos National Laboratory.

Xilinx (2009). *Virtex-5 FPGA RocketIO GTP Transceiver*. Xilinx.

Yiannacouras, P., Steffan, J., and Rose, J. (2006). Application-Specific Customization of Soft Processor Microarchitecture. In *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA)*, pages 201–210.

Young, C., Bank, J., Dror, R., Grossman, J., Salmon, J., and Shaw, D. (2009). A 32x32x32, spatially distributed 3d fft in four microseconds on anton. In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis (SC)*, pages 1–11. IEEE.

CURRICULUM VITAE