TOHOKU UNIVERSITY
Graduate School of Information Sciences

Interconnection Networks for High-Performance Stream Computing with FPGA Clusters

(FPGAクラスタによる高性能ストリーム計算のための相互接続網に関する研究)

A dissertation submitted for the degree of Doctor of Philosophy
(Information Sciences)

Department of Computer and Mathematical Sciences

by

Antoniette Pangilinan MONDIGO

January 14, 2020

Interconnection Networks for High-Performance Stream Computing with FPGA Clusters

Antoniette Pangilinan MONDIGO

Abstract

High Performance Computing (HPC), as a field that relies heavily on cutting-edge technologies, is among the various domains affected by the impending end of Moore's Law and Dennard scaling. Many advancements in computing architecture across the various technology stack levels are being considered and explored to meet and support the growing demands of HPC applications. Field Programmable Gate Arrays (FPGAs), along with other dedicated acceleration platforms, are playing significant roles in this endeavor.

FPGAs, as reconfigurable devices, are recently seen as promising, energy-efficient hardware solutions due to a good balance between their flexibility and efficiency characteristics. Despite their typical lower operating frequency range than the other traditional platforms', creating custom hardware allows massively parallel operations with high utilization rates. Fine-grained and coarse-grained parallelism could be enabled and exploited by creating deep and wide computing pipelines with regular memory accesses. This allows continuous data streams to pass through the pipelines, while increasing the number of operations per memory access; therefore, fully utilizing the available bandwidth. All these make stream computing with a data flow model suitable for low operational intensity applications, such as stencil computing algorithms in FPGAs, which has been successfully demonstrated in numerous case studies. However, the resource budget of a single FPGA limits further performance scaling.

Just as clustered architectures dominate the current HPC trends, the utilization of FPGA clusters is a promising approach. However, despite numerous, successful case studies, FPGAs still lack widespread acceptance in general-purpose HPC installations. With this premise, it is the general direction of this research to aim at making FPGAs accelerators of HPC offloaded applications through system-wide custom computing. In order to achieve high performance, the main strategies are to increase the design space to support the increase of processing units, to reduce interconnection overhead, and to improve HPC application algorithms through customization. In this regard, stream computing with data flow is a suitable approach to fully exploit FPGA capabilities to meet

high performance and high scalability demands in HPC.

Recently, large-scale deployments of FPGA clusters in data centers and cloud services have demonstrated the feasibility of providing a system-wide custom computing infrastructure with FPGAs. However, the inter-FPGA interconnection network is an overhead-inducing region in the extended design space, which could affect the overall performance. Thus, there is a need to investigate a network's performance characteristics in order to achieve low-latency and high-throughput communication, especially for high-performance stream computing. Since the clustering architecture of FPGAs is typically selected based on target workloads and its required performance, choosing an appropriate interconnection network for FPGA clusters becomes an important aspect in this research. This dissertation is focused on investigating scalable interconnection networks for high-performance stream computing FPGA clusters. In particular, this dissertation focuses on the comparison of direct and indirect networks, with specific focus for stream computing requirements.

The main objective of this dissertation is to explore appropriate interconnection networks for high-performance stream computing FPGA clusters, where suitability and feasibility of direct and indirect networks are investigated. Direct networks are a common interconnect approach in existing FPGA clusters due to their low-latency and scalable characteristics. Indirect networks, on the other end, are not widely explored in FPGA clusters due to their overhead in communication latency, but promises a scalable and flexible connectivity in creating a custom network datapath, which is necessary for forward portability in HPC. In this dissertation, a 1D torus or ring topology and a tree topology with switches are adopted in investigating a direct network and an indirect network of FPGAs, respectively.

In Chapter 2, the requirements for stream computing in FPGA clusters are investigated. Most HPC applications, which include stream computing, require a scalable network architecture with a small FPGA footprint, and an efficient, low-latency, high-bandwidth communication. In addition, one functional requirement for stream computing is the support backpressure signals in the backpressure-less channels of the inter-FPGA network. Another challenge is the synchronization of communicating FPGAs. This chapter proposes a lightweight and efficient hardware backpressure mechanism for direct and indirect inter-FPGA communication. This is done by creating a custom network protocol with credit-based flow control for backpressure propagation between communicating FPGAs. Furthermore, to achieve high-performance and highly-efficient communication, which is important to stream computing, it is the goal of this chapter to identify the proposed backpressure mechanism's design parameters and understand how they affect overall performance. While the hardware backpressure mechanism is implemented on a direct network in this chapter, the same design principles and mechanism apply for an

indirect network, which are further discussed in Chapter 4.

Chapter 3 focuses on direct interconnection networks with high-speed transceiver links. Stream computing applications require low-latency and high-bandwidth communication. Since the hardware resource of a single FPGA is limited, one idea to scale the performance of FPGA-based HPC applications is to expand the design space with directly connected FPGAs. This chapter presents a scalable architecture of a deeply pipelined stream computing platform, where available parallelism and inter-FPGA performance characteristics are investigated to achieve a scaled performance. For a practical exploration of this vast design space, a performance model is presented and verified with the evaluation of a tsunami simulation application implemented on Intel Arria 10 FPGAs. Scalability analysis is also performed, where speedup is achieved when increasing the computing pipeline over multiple FPGAs while maintaining the problem size of computation. Performance is scaled with multiple FPGAs; however, performance degradation occurs with insufficient available bandwidth and large pipeline overhead brought by inadequate data stream size. An existing, hardware bandwidth-compression is applied to the communication links to mitigate the performance degradation caused by the bottleneck-prone inter-FPGA links, which resulted to improved efficiency.

In Chapter 4, indirect networks with high-speed Ethernet switches are investigated. As FPGAs become a favorable choice in exploring new computing architectures for the post-Moore era, a flexible network architecture for scalable FPGA clusters becomes increasingly important in HPC. In this chapter, a scalable platform of indirectly-connected FPGAs is presented, where its Ethernet-switching network allows flexibly customized inter-FPGA connectivity. However, for certain applications such as in stream computing, it is necessary to establish a connection-oriented datapath with backpressure between FPGAs. Due to the lack of physical backpressure channel in the network, the Ethernet-switched network utilizes the custom credit-based network protocol with flow control introduced in Chapter 2 in order to provide receiver FPGA awareness and is tailored to minimize overall communication overhead, introduced by the variable latency in using Ethernet switches. To know its performance characteristics, necessary data transfer hardware on Intel Arria 10 FPGAs is implemented, and its communication performance is modeled, which is then compared to a direct network's. Results demonstrate that the connection-oriented Ethernet-switched network achieves equivalent performance to a point-to-point network for stream computing with large data sets, which suggests good performance and scalability for large HPC applications.

Through prototype implementations, obtaining performance characteristics, performance modeling, design space explorations, and performance evaluations, these different evaluation methods in this dissertation have demonstrated the suitability and feasibility of direct and indirect networks for stream computing FPGA clusters. Since stream com-

puting applications generally process large data sets, streaming these sufficiently large data streams scale the performance linearly with more FPGAs on both direct and indirect network types, since they are able to achieve equivalent network throughput. Due to this, both direct and indirect networks would be good choices for inter-FPGA communication for high-performance stream computing. On the other hand, performance of insufficient data stream sizes on both network types demonstrates the communication latency as an overhead-inducing factor, causing degradation of performance. In this case, the indirect network's total transmission time is higher than a direct network's, in which latency dominates, therefore, negatively affecting the overall performance.

For future work, design space exploration should be done with the newly released Intel Stratix 10 FPGAs, where their transceiver links support 100 Gbps data rate. This implies an improved effective network bandwidth, which suggests better performance for both direct and indirect networks. Another area of future work is to provide a standard platform for FPGA cluster management, such as mapping of applications and network configurations. As a general direction, the indirect network provides a scalable and flexible infrastructure for high-level synthesis compilers and virtualization management of a large-scale FPGA cluster.

Table of Contents

List of Figures

List of Tables

List of Acronyms

ALM Adaptive Logic Module

ASIC Application-specific Integrated Circuit

CPU Central Processing Unit

DMA Direct Memory Access

DSP Digital Signal Processing

FIFO First-In First-Out

FLOPS Floating-point Operations per Second

FPGA Field Programmable Gate Array

GPU Graphics Processing Unit

HPC High Performance Computing

LAB Logic Array Block

MLAB Memory Logic Array Block

NIC Network Interface Controller

OSI Open Systems Interconnection

XLAUI 40 Gbps Attachment Unit Interface

Chapter 1

Introduction

1.1 Background and Motivation

Advancements in computer architecture across the various abstraction levels are being explored for many years. Various platforms, compute models, and architectural strategies are being actively considered to improve both performance and power efficiency. This is primarily due to the impending end of Moore's law [5] and Dennard scaling [6], which expedites the rising percentage of transistor capacity under-utilization, known as the *dark silicon* [7].

One of the areas affected by this imminent phenomenon is High Performance Computing (HPC) which heavily relies on cutting-edge technologies to achieve higher performance. HPC deals with modeling and simulation workloads from science, engineering, commerce, and different industries, which requires extreme and complicated computations that often employ ultra-fast, high-capacity, large-scale computing architecture [8]. The use of specialized accelerators, such as graphics processing units (GPUs), is one of the employed strategies to meet the growing demands of HPC applications. However, similar to the fate of CPUs, they are power-consuming devices, which limits deployment size and needs to be addressed in large-scale utilization, such as in supercomputers.

Field Programmable Gate Arrays (FPGAs) are recently playing a major part in the exploration of power-efficient architectural advances. FPGAs are reconfigurable devices

that contain generic logic and interconnect, which allows customized digital circuits on its fabric. As a result, optimized circuits of target applications are often explored and implemented. Traditionally, they were widely used for fixed-point digital signal processing (DSP) but now, they offer high floating-point processing capacity, allowing them to execute high-performance demanding applications [9–13]. Latest variants, such as Intel Generation 10 FPGAs (Arria 10 and Stratix 10), have included hardened floating-point DSP blocks and significantly increased their computing density, while advancing further in performance and power efficiency [12, 14, 15]. The recently released Stratix 10 FPGAs, for instance, can reach peak floating-point performance comparable to that of the latest GPUs [10, 16].

FPGAs are designed for low-power operation, which were originally intended for application-specific integrated circuit (ASIC) emulation, and have been around longer than GPUs. They have lower chip frequency range than typical GPUs' and CPUs', which contributes to their power efficiency. Despite the lower operating frequency, creating custom hardware in the FPGA fabric allows massively parallel operations with high utilization rates. Parallelism is enabled and exploited through a deep pipeline of computational units, which can deploy computation blocks with user-defined circuitry rather than through processors that take instructions, such as in von Neumann architecture [17–19]. Numerous FPGA case studies for relevant HPC problems have been performed, which have shown better performance when compared with CPU's and/or GPU's, such as in geophysics [20], molecular dynamics [21], bioinformatics [22], climate modeling [19], and computational fluid dynamics (CFD) [23]. In particular, FPGAs excel as efficient hardware accelerators because of their ability to customize algorithms and exploit both fine-grained and coarse-grained parallelism in offloaded applications [9, 24–26].

Despite the numerous promising results in reconfigurable computing research, FPGAs have not garnered significant impact on general-purpose HPC systems, which is due to an apparently huge gap between the potential and reality for FPGAs in HPC [27]. An interesting position paper by Underwood et al. [1], which was published a decade ago, proposed 12 specific areas to hasten FPGA adoption in HPC environments. These areas

State of FPGA Research Toward HPC (2009~)

Area	Status	Activity	Difficulty
Step 1: Standardization	poor	moderate	low
Step 2: High Performance Forward Portability	poor	low	high
Step 3: Enhanced Device Performance	good	low	high
Step 4: Enhanced System Architecture	fair	none	moderate
Step 5: Simplified Library Usage	fair	low	low
Step 6: Concurrent APIs	poor	low	low
Step 7: Better Performance Studies	fair	moderate	moderate
Step 8: Improved Programming Environment	good	high	high
Step 9: Improved Infrastructure	poor	low	moderate
Step 10: Enhanced Communications	good	moderate	moderate
Step 11: Enhanced Reliability	poor	low	high
Step 12: Provide OS Support	poor	low	low

Figure 1.1: Critical Areas for FPGA Research Identified by Underwood et al. [1]

are shown in Figure 1.1, where most FPGA research for HPC revolves within. Since then, the ecosystem has been evolving through modifying applications (customization), the need for energy efficiency solutions (low-power operations), and the discovery of *killer applications* for FPGAs, such as in deep neural networks [16, 28–30] (real applications). These developments bring about the continuous efforts in exploring FPGAs in the HPC landscape. Presently, the observations and conclusions by Underwood still apply to some extent, thus the need for further research within these areas to increase acceptance of FPGAs in HPC [27].

Customization with streamed data flow through pipelined computations is the primary key to achieve high performance gains with FPGAs [17]. To obtain peak performance of FPGA applications, *stream computing* is one of the promising FPGA computing models, which relies on exploiting parallelism by deep and wide pipelining [18,31–33]. Generally, it is also known as a *stream processing* system, which consists of computing units that process data in parallel and interconnected communication channels [34]. In the terminology established by Buck [35], data is sequenced and organized into *streams* or collectively, a *data stream*, in which its data elements are mutually independent. Operations applied to each data element on the stream is called a *kernel*. Since data stream elements are independent, data-level parallelism could be exploited by operating individual elements

in parallel at the kernel level. Task-level parallelism can be exploited through parallel operations of independent kernels in a pipelined fashion [36]. Through deep and wide pipelining, constant throughput is guaranteed, which contributes to high-performance computations.

In contrast to standard processors with random memory accesses, stream computing relies on sequential accesses to external memory to read and write data streams [32]. In FPGA stream computing model, this relies on direct memory access (DMA) to read and supply continuous data stream to a kernel, which its internal pipelines process this. In turn, the pipeline results are output as a stream to be written back to the memory. Through regular memory accesses, the available memory bandwidth is exploited and conceals memory access latency. This makes stream computing with data flow model suitable for low operational intensity applications such as stencil computing algorithms in FPGAs, which typically involve large-scale numerical calculations with large data sets and have been successfully demonstrated in [33, 37–42].

With large data sets to stream, stream computing performance is determined primarily by its throughput since the processing kernels containing multiple operations are pipelined. Factors affecting throughput include operating frequency, bit-width of the pipeline and its datapath, and available memory bandwidth, which in turn affects performance. Furthermore, performance scaling through deep and wide pipelines in the kernels is limited by the FPGA's on-chip resources.

On the other end, HPC research at a global scale has advanced different computing architectures over the years, from a single-processor computer to dominating clustered architecture in the present [2]. As shown in Figure 1.2, the trend has favored clustering of commodity devices over the others, such as Massive Parallel Processing (MPP) and Symmetric Multiprocessing (SMP) architectures. As of November 2019's TOP500 list of supercomputers, 91.6% of them have adopted the clustering architecture, while MPPs are slowly decreasing at now 8.4%.

With this context, extending the focus to FPGA clusters in HPC seems natural, especially when performance scalability is limited by the available resources of a single FPGA.

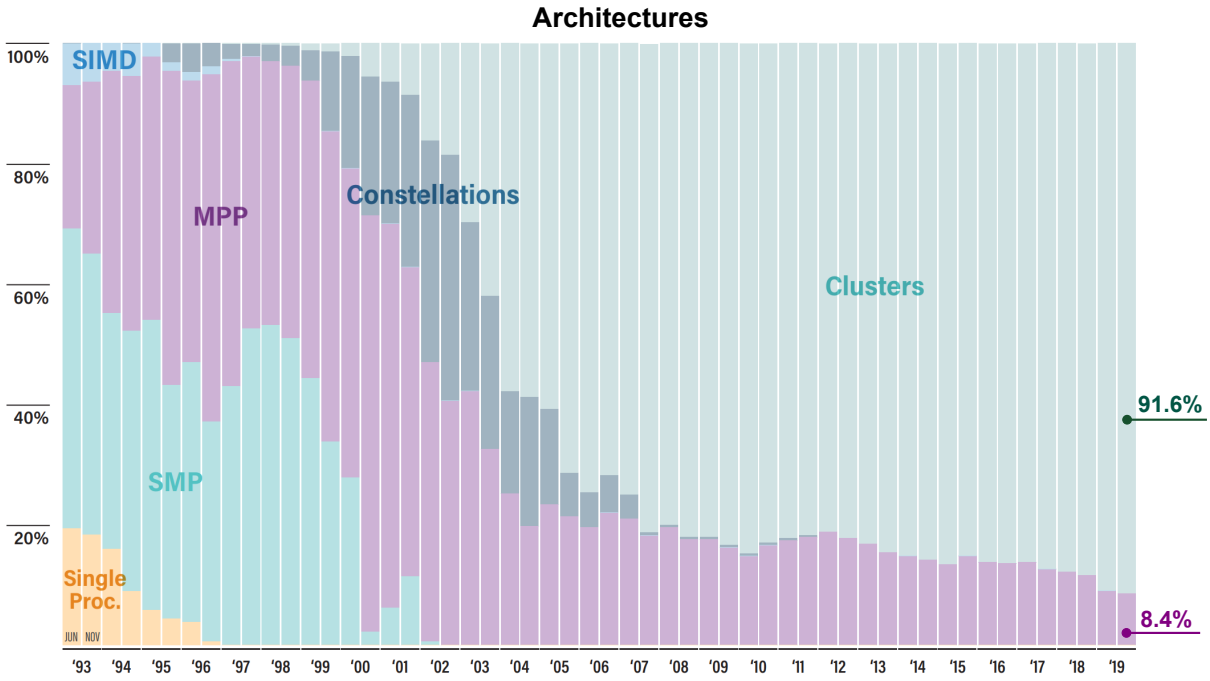


Figure 1.2: HPC architectures over time (November 2019) [2]

Currently, massive investments of FPGA clusters in data centers and cloud services are deployed and explored [43–49], which have demonstrated the viability of mapping large HPC applications into multiple FPGAs that could achieve a scaled performance otherwise limited by a single FPGA chip [46,50–53]. These deployments demonstrate the feasibility of providing a system-wide custom computing infrastructure with FPGAs as accelerators of HPC offloaded applications, which is the general direction of this research.

The clustering architecture of FPGAs is typically selected based on target workloads and its required performance [54]. An FPGA cluster is a set of loosely or tightly connected FPGAs working together as a single system. *Loosely-coupled* clusters do not require communication between FPGAs and only need a connection to host processors [54]. This is a classical organization, which is typical for offloaded applications on a single FPGA. On the other hand, *tightly-coupled* clusters, require frequent communication between FPGAs [54]. For the latter type, an FPGA interconnection network is beneficial and practical since modern FPGAs now have high-speed transceiver links. This dedicated network would provide low-latency communication channels between frequently

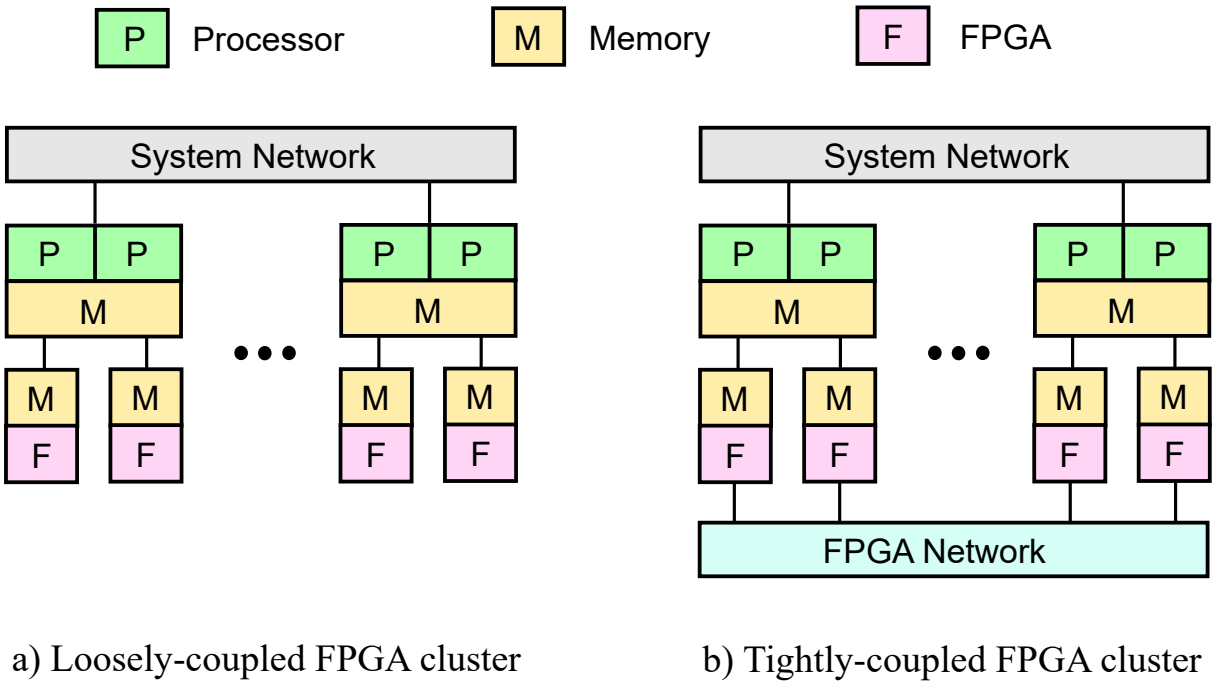


Figure 1.3: FPGA clustering architectures

communicating FPGAs; therefore, eliminating the FPGA-to-processor bottleneck when transmitting through standard network interface controllers (NICs), as with the case for loosely-coupled types. Figure 1.3 shows simplified block diagrams of the two clustering types.

As with HPC clusters, computational units and interconnection network are the main components of an FPGA cluster, which are also the major factors that determine the overall performance. To increase computational performance, there are three main strategies: increase the processing units, reduce interconnection overhead, and improve the application algorithm [55]. The first strategy could be achieved by increasing the number of FPGAs, which expands the design space to accommodate increased parallelism. To realize the second one, building a tightly-coupled FPGA cluster provides low-latency communication channels dedicated for inter-kernel communication across FPGAs. For stream computing applications, leveraging deep and wide pipelines on their customized kernels addresses the third strategy.

Collectively, increasing the number of FPGAs proportionally increases the amount of computational resources. The challenge is in the interconnection network region, where

there are many design factors to consider such as physical constraints, performance requirements, scalability, and expected workloads. Since FPGA clusters are relatively new in HPC, there is a need to investigate the interconnection network types and their performance characteristics in order achieve high performance communication while minimizing overhead. Due to the customizable nature of FPGAs, including their interconnection mechanism, the overall design space is huge.

A classification scheme for interconnection networks was introduced by Duato et al. [3], as shown in Figure 1.4, which categorizes four major network types, based primarily on network topology and was derived on the classification proposed in [56]. These are *shared-medium*, *direct*, *indirect*, and *hybrid* networks. Figure 1.4 also indicates a few of these networks implemented for parallel computers.

Shared-medium networks allow all communicating devices to share the transmission medium, where only one device is allowed to use the network at a certain time, usually with restricted bandwidth and an arbitration mechanism [3]. Due to the limited network bandwidth, performance scalability is affected, especially when the shared medium could only support a limited number of devices before it becomes a bottleneck. Two alternative approaches to the shared-medium networks are the networks with directly connected devices, known as *direct networks*, and networks with indirectly connected devices through switches, known as *indirect networks*. Lastly, *hybrid networks* generally combine the mechanisms of shared-medium networks and direct or indirect networks. They allow an increase of bandwidth with respect to shared-medium networks and reduce distance between devices with respect to direct or indirect networks [3]. However, for applications requiring very high performance, direct and indirect networks achieve better scalability than hybrid networks due to the constraints and performance limitations of the shared-medium buses [3].

For tightly-coupled FPGA clusters, *direct networks* involve point-to-point links directly connecting neighboring FPGAs and are common choices for FPGAs, since it scales well. To communicate between non-neighboring FPGAs, a transmitted information or *message* has to pass through several intermediate FPGAs along some route in order to

reach its destination. A common component handling message communication and its traversal, is a *router*, which earned direct networks the term *router-based networks* [3]. An FPGA, usually has an embedded router and is connected to the on-chip routers of its neighbors through the transceiver links with pairs of unidirectional channels in opposite directions. The more FPGAs are connected in the direct network, the total communication bandwidth, memory bandwidth, and processing capability of the FPGA cluster also increases, making them a popular interconnection architecture for large-scale clusters [3]. Figure 1.5a shows a direct network with a 1D torus or ring topology with bidirectional links interconnecting adjacent FPGAs. Meanwhile, Figure 1.5b illustrates a direct network with on-chip routers to construct a 2D torus topology.

Indirect networks or *switched-based networks*, on the other hand, need one or more external switches to interconnect the FPGAs, instead of direct connections [3]. Each FPGA has a dedicated network adapter, which connects to a network switch through its transceiver links. A switch have *ports* with input and output directions, where some ports are connected to FPGAs, while others are connected to ports of other switches in order to provide connectivity to more FPGAs [3]. Since message traversal is centralized in the switches, network flexibility can be achieved through customized network datapath between FPGAs, in which an arbitrary or virtual topology can be built according to a target application's actual communication patterns. Flexibility, in this aspect, supports forward portability for other communication patterns, which is in line with the general direction of this research: for FPGAs to be widely accepted as accelerators for HPC applications. Just as customization of the target application is one of the approaches to obtain high performance in FPGAs, flexibility of the network datapath through customization is likewise promising. Figure 1.5c presents an indirect network with a centralized Ethernet switch, where an arbitrary topology can be configured based on the requirements of the target application.

While interconnection networks for parallel computers are already widely explored [3], only a few studies were done to explore dedicated networks for large-scale, tightly-coupled FPGA clusters targeting high-performance applications, such as with the hardware stream

computing model [17, 18], which is efficient for FPGA computations. For existing FPGA clusters, direct networks are a common choice, but not much exploration has been performed with indirect networks, especially on a large-scale setup. In this work, the scope is focused on the comparison of direct and indirect networks for high-performance stream computing. In particular, a 1D torus or ring topology and a tree topology with switches are adopted for investigating a direct network and an indirect network, respectively.

The main goal of this dissertation is to know the performance characteristics of direct and indirect networks as dedicated interconnect topology for stream computing tightly-coupled FPGA clusters. Since communication affects performance by introducing additional overhead, there is a need to explore these networks in order to obtain low-latency and high-throughput communication necessary for stream computing applications. As for general performance requirements, HPC applications, including stream computing applications, usually demand for scalability, high-bandwidth, low-latency, and efficient communication, with a small footprint on FPGA fabric. However, one of the functional requirements of high-performance stream computing is to provide a high-bandwidth and efficient communication with backpressure throughout the pipeline, which should extend to its backpressure-less, asynchronous interconnection network. In this dissertation, such backpressure mechanism is proposed for both direct and indirect inter-FPGA communication networks.

1.2 Related Works

Several computing FPGA clusters with direct networks for HPC have already been developed. For instance, the single configuration Cube [57] is a massively-parallel FPGA-based platform that used eight boards with 64 Spartan-3 FPGAs. However, its fixed systolic connection makes it less flexible than other existing FPGA clusters and limits the class of suitable applications it could run. The Berkeley Emulation Engine 2 (BEE2) system [58] was developed for event-driven network simulation and have five Xilinx Virtex-2 Pro FPGAs, hosted on a single printed circuit board (PCB). A star topology was used

to connect the four computational FPGAs in a 64-bit ring and a control FPGA at the center of the network. While the outer ring handles computationally intensive tasks, the control FPGA runs a Linux operating system and manages an off-board I/Os, therefore, introducing a complicated programming model. Another scalable FPGA cluster platform, RAPTOR-Xpress [59] is built for rapid prototyping. It consists of a base board with 64 Xilinx Virtex-5 FPGAs on its 16 sub-board systems and communication interfaces. The FPGAs are connected in a ring topology but the study did not examine the case of inter-communication effects on performance. Since it is targeted for prototyping, they offer a large amount of computing power but for specific computing requirements, the platform's resources may not be fully utilized.

These distributed FPGA systems vary in different forms that employ direct interconnects with various topologies that prioritize different requirements. A boxed cluster like BlueHive [54, 60] is a custom 64-FPGA cluster with a full custom interconnect intellectual property (IP), BlueLink, and custom communication protocol with reliability layer. However, its reliability features were at the expense of a lower bandwidth efficiency. Maxwell [61] has 64 FPGAs on a 2D torus with each link using a single multi-gigabit transceiver. In [62], 32 FPGAs on eight enclosures are interconnected in different technologies and topologies on a Berkeley Emulation Engine 3 (BEE3) multi-FPGA platform for network exploration. With these direct topologies, most of them are exploring small to mid-scale clusters.

In academic research projects such as in [63–66], their FPGA clusters have addressed high-speed and low-latency inter-FPGA communication. While these network solutions exhibited good performance, they occupy a large area on the FPGA fabric, which means lesser area for user applications. In EXTOLL research project [63] for instance, they introduced a low-latency message exchange mechanism. However, reported device utilization showed 85% of the FPGA resources were used up even after floorplanning optimization procedures, which highly suggests high complexity of the approach.

Other works targeting large-scale clusters are typically for heterogeneous computing. Datacenter-scale deployments such as Catapult v1 [43] uses a dedicated direct network for

its 48 FPGAs, where they arranged 6x8 2D torus topology. In Catapult v2 [44], they used a "bump-in-the-wire" approach, which accelerates network traffic by routing communication through FPGA. They used a tree topology with top-of-rack (TOR) servers and used User Datagram Protocol/Internet Protocol (UDP/IP) protocol over 40 Gbps Ethernet. Another heterogeneous cloud data center-based FPGA cluster [53] uses OpenStack, a cloud management tool offering several services, to virtualize FPGA utilization with other heterogeneous resources, which involves multiple abstraction layers in its infrastructure implying additional overhead. Direct networks for FPGA clusters are common and widely used, but to the best of our knowledge, performance characteristics of FPGAs with an indirect network using switches have not been extensively explored, particularly in a large-scale setup. Interconnection networks form a vital role in the overall performance of an FPGA cluster. Finding the acceptable balance between many factors affect the appropriate choice of network for high-performance solutions. While there is no general-purpose hardware architecture to harness the full potential of FPGAs for all intended applications [24, 67], this dissertation proposes to have a generalized network framework through custom network datapath, where a target application can reconfigure its necessary topology appropriately.

In FPGA-based custom computations, several approaches can be implemented to achieve high performance, such as latency hiding of independent functions and data streaming through pipelined operations [9]. Azarian and Cardoso [68] investigated the coarse-grained and fine-grained data flow synchronization approaches to achieve pipelining execution of the tasks in FPGA-based multicore architectures, in which results show a speedup in the overall execution through the use of multiple intellectual property (IP) cores provided by FPGAs. Primarily, they used a Xilinx Virtex 5 FPGA with MicroBlaze soft microprocessors, which is inefficient in FPGA-based computations. Ziegler et al. investigated the effects of coarse-grain pipelining on multiple FPGAs [69]. They described the effects of parallelizing sequential imperative programs into pipelined implementations for FPGAs and increased throughput. They specifically focused on coarse-grain inter-loop pipelining with considerations of the memory bandwidth. However, similar to the work

in [59], they did not investigate the effects of inter-FPGA communication overhead on overall system performance.

In stream computing applications, Murtaza et al. [70] demonstrated a Lattice Boltzmann method (LBM) in Maxwell, a multi-FPGA system, by exploiting parallelism to a massively-parallel accelerator implementation of floating-point-based cellular automata. LBM computation is a fluid simulation belonging to computational fluid dynamics (CFD) methods, where its numerical algorithm is based on generalized cellular automata [71], requiring multiple data accesses per unit operation. Results showed that speedup diverges from linear scalability for more than eight FPGAs, since parts of the computations were co-processed by a CPU. In another study [72], a performance model of an LBM accelerator implemented on a tightly-coupled FPGA cluster with 1D ring of accelerator domain network (ADN) was presented. They utilized Intel Stratix IV and also predicted the scalability when Stratix V FPGAs are used instead. It was found that the newer Stratix V are much better than strong scaling with more FPGAs than the older Stratix IV since it has a higher network bandwidth. In addition, it was reported that the memory bandwidth has less impact on strong scalability than the network bandwidth. As an improvement, [73] presented a detailed design of processing elements for LBM with fully-streamed computation for all LBM stages through ADN connection of a tightly-coupled FPGA cluster using Stratix IV FPGAs. In [41], a fully-streamed computation for all LBM computing stages was created and processed on a single FPGA, where the CPU co-processing was eliminated. Results demonstrated 97.9% utilization of the peak performance with a single pipeline of 18 cascaded computing units, where dedicated data flow-based floating-point operations are defined. However, it was also discovered that 99.6% consumption of floating-point digital signal processors (DSPs) in a single FPGA limits the scalability. Another stream computing application, tsunami simulation uses a method of splitting tsunami (MOST) algorithm, as demonstrated in [40], and is capable of delivering high throughput with FPGA-based stream computing approach. For a single Arria 10 FGPA, the highest sustained performance was achieved by a single pipeline with six cascaded computing units, where its scalability is also limited by the available

floating-point DSPs. Similar to [41], this suggests the feasibility of extending the pipeline depth into multiple FPGAs.

The use of multiple FPGAs also suggests the evaluation of the inter-FPGA communication. While Transmission Control Protocol/Internet Protocol (TCP/IP) is popular for internetworking systems, it is resource-heavy and designed for complex, unpredictable network, such as the Internet. A fully-customized protocol, BlueLink [54], showed better area-performance characteristics than existing network protocols for their custom computing requirements. Jun et al. [74] presented a parameterized, low-overhead transport layer network with virtual channels and end-to-end flow control for distributed FPGA applications. Their prototype cluster is made up of 20 Xilinx VC707 FPGA boards connected through their high-speed serial transceiver links. Since their transport layer is parameterized, the communication buffer sizes and flow control features can be configured at FPGA synthesis. In this dissertation, the proposed custom network protocol shares some functional similarities with BlueLink, without fully customizing the entire network stack. In addition, a credit-based flow control mechanism [75] is specifically added for backpressure propagation between FPGAs. Unlike in [74], however, careful analysis based on the physical constraints is done for the protocol's communication buffer requirements, which will be discussed in detail within the next section.

Popular implementations of flow control in existing FPGA interconnection networks utilized either credit-based or backpressure (ON/OFF) techniques with dedicated channels [54, 76, 77]. In particular, the credit-based scheme, which was originally designed for Asynchronous Transfer Mode (ATM) systems [75, 78], is selected for a number of reasons [79, 80]:

1. Credits sent carry numerical information about the available downstream buffer space over a dedicated channel regularly, making credit-based flow control faster than its rate-based flow control counterparts;

2. Receiver buffer allocation should be proportional to round trip time (RTT), implying a smaller memory requirement than the other schemes;

3. There is no data loss if there is any congestion since positive credits are never issued unless there is availability in the downstream receiver buffer space; and

4. Data rate can be as high as the full link speed with no data loss, which promises good network resource utilization.

1.3 Objectives

The main objective of this dissertation is to explore appropriate interconnection networks for stream computing on FPGA clusters. Since the inter-FPGA communication could be an overhead inducing component of the cluster, there is a need to investigate the performance characteristics of its interconnection network in order achieve low-latency and high-throughput communication necessary for stream computing. In this dissertation, the particular focus is in comparing direct and indirect networks for tightly-coupled FPGA clusters. The specific objectives are:

1. Investigate the suitability and feasibility of direct and indirect networks for stream computing FPGA clusters;

2. Design and implement a lightweight and efficient hardware backpressure mechanism for direct and indirect inter-FPGA communication; and

3. Investigate and evaluate performance scalability of stream computing on direct and indirect networks.

1.4 Organization and Contributions

This dissertation is organized as follows. Chapter 2 details the interconnection requirements for stream computing in FPGA clusters and proposes a lightweight custom protocol for inter-FPGA backpressure mechanism. The contributions of this chapter are the mechanism, hardware design, and implementation of a custom protocol for inter-FPGA

backpressure; a design space exploration of its design parameters; and the discussion of its performance trade-offs.

Chapter 3 discusses direct networks, while Chapter 4 introduces indirect networks, in which both chapters evaluate the performance and scalability of a stream computing application. The contributions of Chapter 3 are the design and implementation of a scalable, deeply pipelined hardware platform with inter-FPGA direct network; investigation of point-to-point network's performance characteristics; performance model of stream computing on directly-connected FPGAs; and its performance evaluation.

Similarly, the contributions of Chapter 4 are the design of connection-oriented network with Ethernet switches; investigation of its performance characteristics and its performance model; and performance evaluation of stream computing in an indirect network. This chapter also demonstrates that connection-oriented Ethernet switched network achieves equivalent performance to a point-to-point network for stream computing with large data sets.

Finally, Chapter 5 concludes this dissertation and describes implications, limitations, and future work.

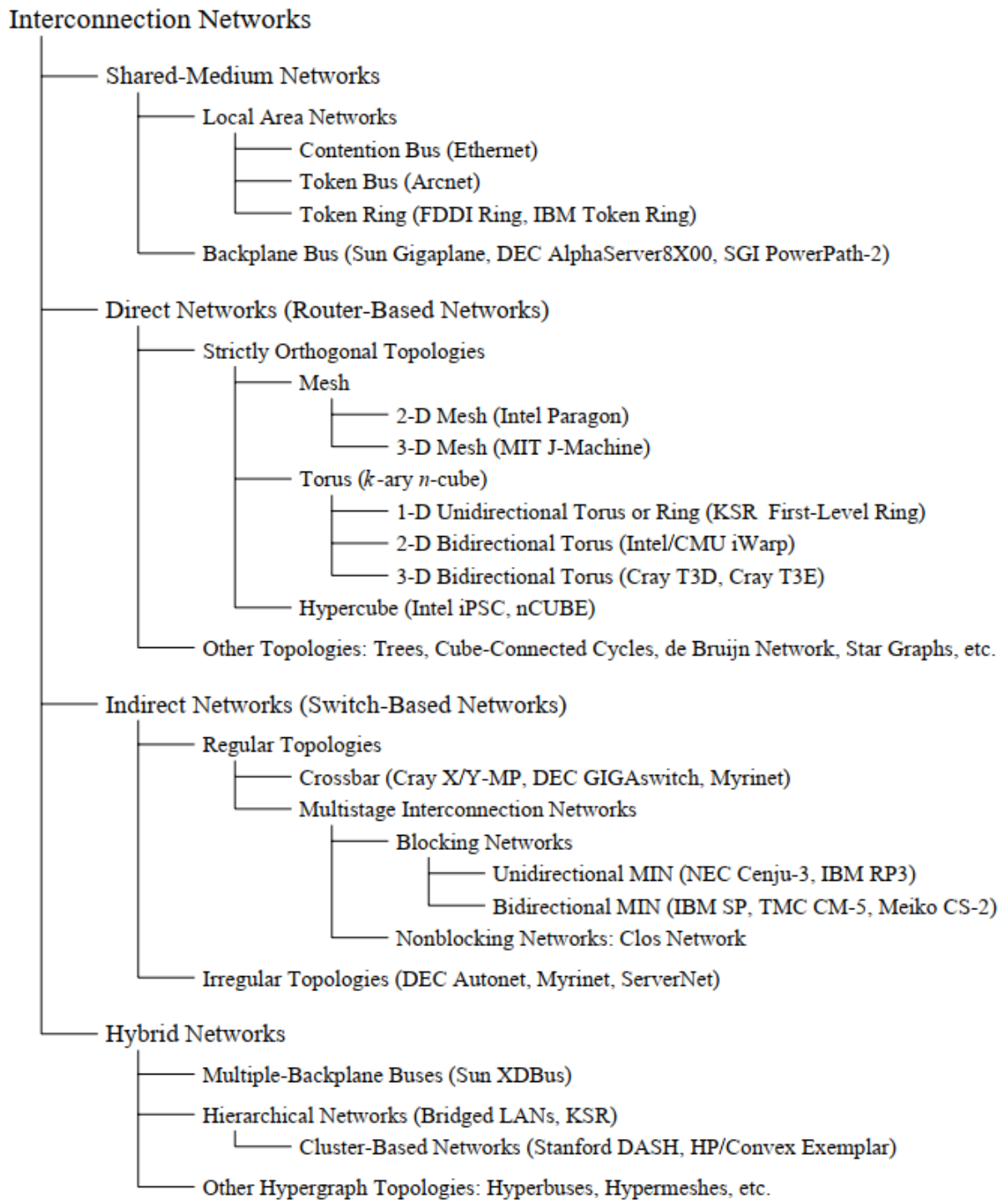
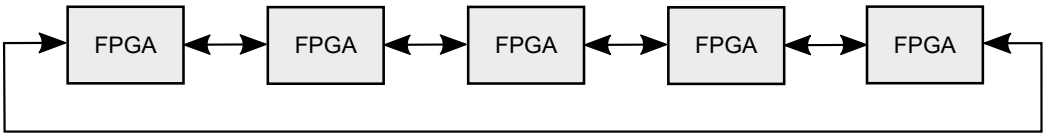
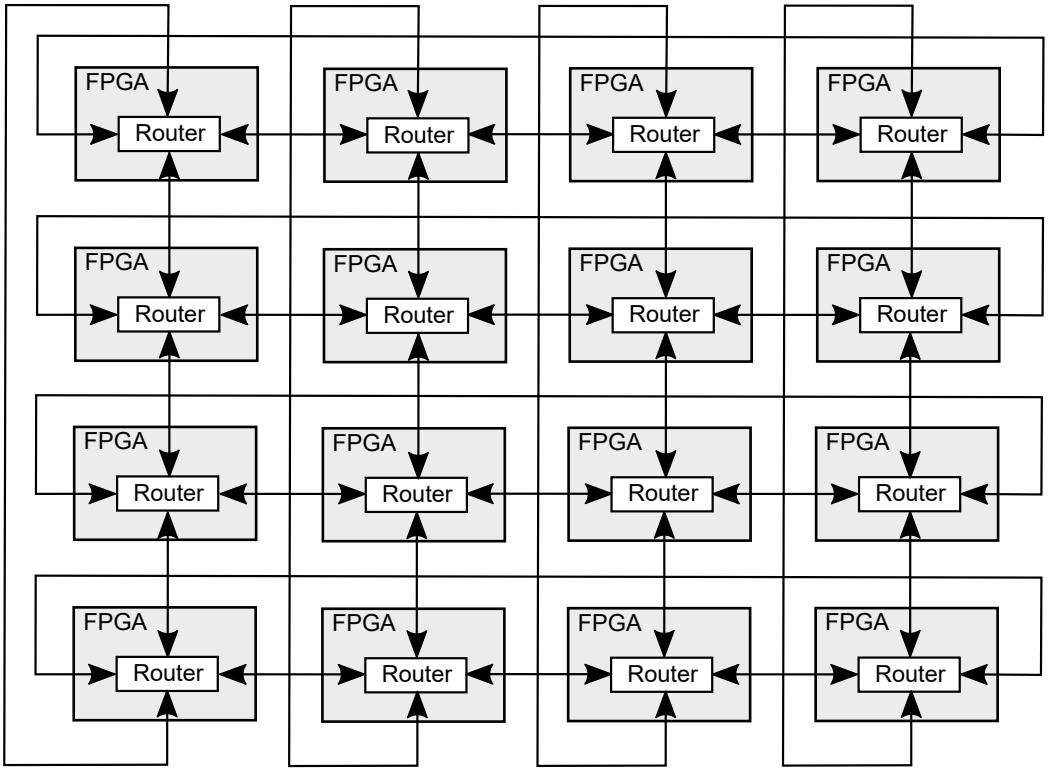


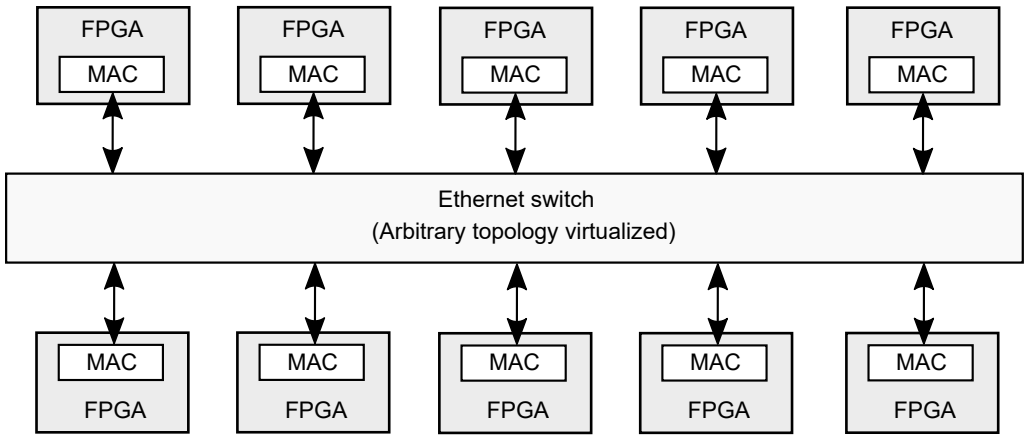
Figure 1.4: Classification of Interconnection Networks [3]



a) Direct network with 1D torus or ring topology



b) Direct network with 2D torus topology



c) Indirect network with Ethernet switch

Figure 1.5: Inter-FPGA Network Choices for Tightly-coupled FPGA Clusters

Chapter 2

Interconnection Network Requirements for Stream Computing in FPGA Clusters

2.1 Introduction

Stream computing in FPGAs is seen as a promising solution in delivering the necessary performance and energy efficiency requirements of compute-intensive applications like numerical simulations. The inherent structure and customizability of FPGAs naturally make them the better alternative in achieving a highly-scalable computing design solution. FPGA's architecture is structured to support custom hardware designs, which could be optimized for efficient processing. At the same time, FPGA's flexibility allows easy connection to other devices via any physical standard or custom interface [81]. Aside from interfacing with other devices like external memory, DSP blocks, I/Os with high-speed transceivers, and other customized chips, FPGAs can be built to form clusters with high-speed and low-latency communication, making them an appealing choice for scalable designs, especially for stream computing.

Although FPGAs operate at clock frequencies lower than their GPU and CPU counterparts, enabling parallelism allows internal processing units to achieve high utilization

rates, even with limited memory bandwidth. This is done by creating a deep pipeline that consists of a significant number of floating-point operations capable of regular memory access. This overlap between memory access and computation conceals the communication latency and is used to extract high-performance gains from the FPGA fabric [9]. This makes stream computing with a data flow model suitable for low operational intensity applications such as stencil computing algorithms in FPGAs. However, the resource budget of a single FPGA limits further performance scaling. In [39] for instance, the utilization of logic elements reached 95% while the other resources only consumed less than 50%. One way to overcome this is to extend the stream computing pipeline with multiple FPGAs.

One identified challenge with stream computing on multiple FPGAs is synchronization of data streams, since the FPGAs are operating in different clock domains. Typically, dual-clock FIFOs at both FPGA transceiver ports handle this by allowing FIFO read and write access at different clocks. Flow control manages data synchronization between different asynchronous units such as FIFOs and the stream computing pipelines by supplying backpressure signals. To provide receiver awareness across the interconnection network, backpressure should also be supplied between the stream computing pipelines of two communicating FPGAs.

In this dissertation, both direct and indirect networks of FPGA clusters are considered. Direct networks involve point-to-point connections between FPGA transceivers while indirect networks involve FPGA connections to a central switching device. For both classifications, different network layers of the Open Systems Interconnect (OSI) are involved. Complex communication protocol stacks are available, such as commercial off-the-shelf protocols for network interface controllers (NICs). However, most of them are inefficient for hardware data transfers and involve unnecessary overhead. On the other hand, designing a fully-customized interconnect benefits from a personalized approach to applications, but at the expense of increased engineering effort. Choosing a partially custom interconnect allows using of commodity parts, such as physical Layer 1 (L1) and data link Layer 2 (L2) hardware modules, while customizing other higher layer functionalities such as the backpressure mechanism for stream computing applications.

In this chapter, a lightweight and efficient hardware backpressure mechanism for direct and indirect inter-FPGA communication is proposed. This is done by creating a custom network protocol with credit-based flow control [75] for backpressure propagation between communicating FPGAs. By removing higher level network layers, a minimal overhead is maintained. To keep high performance communication through low-latency and high-bandwidth transfers, design considerations are also entailed in the custom network protocol, which is discussed in this chapter. The specific contributions are:

1. Mechanism, hardware design, and implementation of a custom protocol for inter-FPGA backpressure;

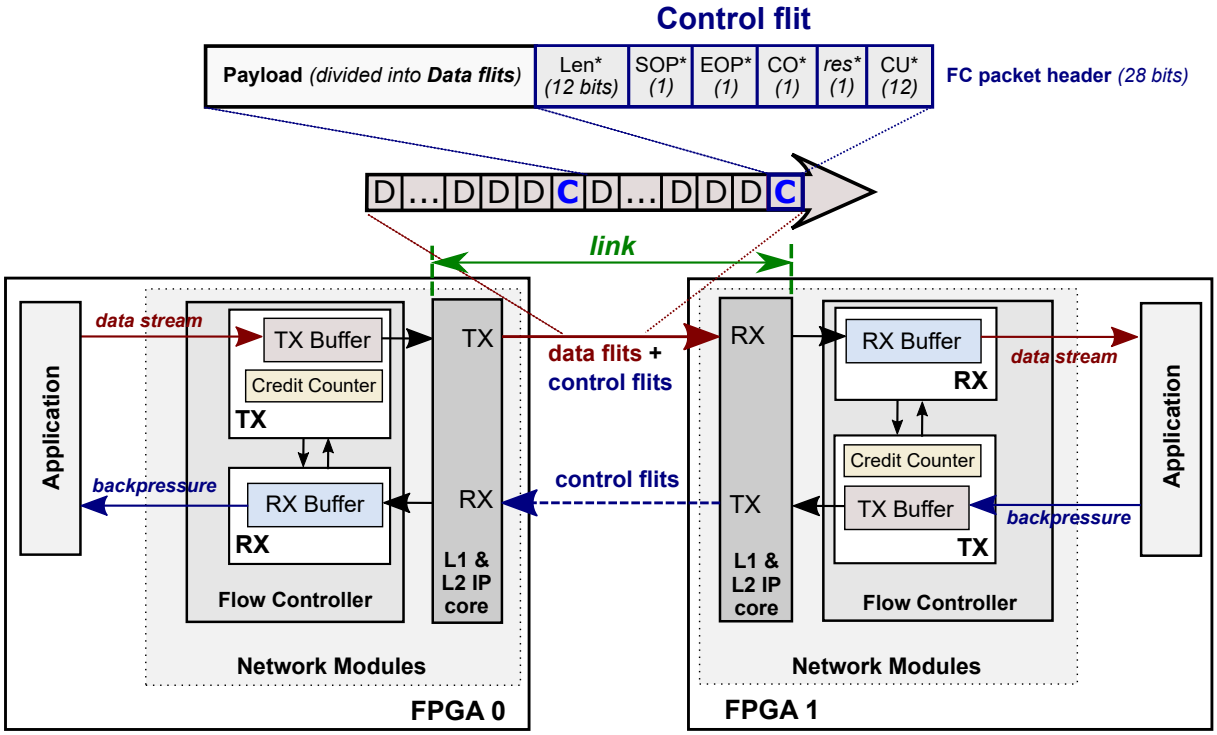
2. Design space exploration of its design parameters; and

3. Discussion of its performance trade-offs.

2.2 Custom Network Protocol with Backpressure

Synchronization is an identified challenge that involves streaming data across multiple FPGAs that are operating in different clock domains across an interconnection network. Since the inter-FPGA transceiver links are asynchronous along with other hardware modules, data streams coming in and out of communicating FPGAs across the network is not always guaranteed to be successfully received by the other end. In this dissertation, a lightweight and efficient custom credit-based network protocol is proposed to be utilized along with L1 and L2 network functions for both direct and indirect networks.

For universality, the proposed protocol targets both point-to-point FPGA transfers (direct network) and a switched communication (indirect network). In addition, it performs an initial synchronization process to establish a *connection-oriented* datapath between the FPGAs, which guarantees the communication line and the other transceiver end to be active and ready for data transfers. Since FPGA's transceiver links are capable of half-duplex and full-duplex communication, this protocol supports both types, without



***Len:** Length of payload in packet, **SOP:** Start of packet flag, **EOP:** End of packet flag, **CO:** Credit only flag, **res:** reserved bit, **CU:** Credit update

Figure 2.1: Hardware implementation of credit-based flow control in half-duplex transfers

any user or application intervention. As for the inter-FPGA backpressure propagation, a credit-based flow control [75] mechanism is adopted and implemented.

2.2.1 Credit-based Flow Control Mechanism

The main purpose of flow control is to provide receiver status awareness between two communicating FPGAs through the exchange of *credits*, which provides transmission reliability to any chosen network protocol for the transceiver links. Figure 2.1 shows two FPGAs with their respective network modules, which involve flow controllers (FC) and L1/L2 modules for data link and physical layer functionalities. The FC module has a full-duplex symmetry, where the same modules are placed on both communicating ends. It also operates autonomously in either half or full-duplex data transfers, for both direct and indirect networks.

FC handles initial inter-FPGA synchronization, where both FPGAs are ensured to be available for communication. This follows after a system reset, where the network

datapath is configured. For a direct network, this means the physical cabling connections between FPGAs. For an indirect network on the other hand, this means the intermediate switches and the FPGAs are ready for communication. For both network types, this synchronization process establishes the logical link connection, which is similar to a handshaking procedure. In the transmit direction, this procedure involves continuous sending of *sync flits*, with each one composed of a unique bit sequence recognized by the receiver FPGA's FC. Here, a flow control digit *flit*, is defined as a smaller unit of data from a larger payload size that is sent in one cycle, in which a single flit has a:

$$(\text{Flit size}) = \frac{w\text{-bit width}}{8} \quad [\text{bytes}]. \tag{2.1}$$

Sync flits are sent until receiving FPGA responds with its own sync flits and are received by the transmitting FPGA.

After synchronization, FC receives data from the application containing the stream processing pipelines. The incoming data stream or payload is divided into smaller packets composed of *data flits*. In each FC packet, a *header* is inserted and is the first flit to be sent out into the network. This is also known as a *control flit*, in which other information are embedded in order to reconstruct the original payload in the receive direction. As shown in Figure 2.1, the FC header includes the payload length, start-of-packet (SOP) and end-of-packet (EOP) flags, and credit only (CO) flag to indicate a zero-payload packet for half-duplex transfers.

Credit update (CU) field embeds the *credit* sent for the other FPGA's *credit counter*, which keeps track of its own sent data flits. The FC transmitter (TX) only sends a flit when its credit counter has a positive numerical value (non-zero), indicating that the receiver FPGA can still accommodate incoming data flits, and thus, should send credit updates regularly. The credit-based scheme allows TX to transmit only when there is available buffer space (credits) in the downstream receiver (RX) FPGA, which mimics the backpressure effect of a physical channel. Figure 2.1 shows a half-duplex transfer, where one direction inserts control flits within data flits, while the other direction is only

sending control flits for the credit updates. In the case of a full-duplex communication, both channels will be sending control flits and data flits in both directions.

2.2.2 Flow Control Design Parameters and Performance Trade-offs

The CU frequency depends on the size of the FC packet, since the credits are embedded in the packet headers. This is important since it also allows the FC receiver (RX) to identify incoming flits as control or data flits. The size of the FC packet is dependent on the depth of the store-and-forward TX buffer, which is set as a parameter.

In this dissertation, the allocated size for both TX and RX buffers is considered to minimize area consumption without sacrificing performance. However, sufficient depth is necessary to reduce overhead while minimizing additional latency. In particular, the TX buffer should have shallow depth to minimize induced latency, but large enough to occupy more data flits in a single packet for a lesser TX overhead. Equation (2.2) shows the inter-FPGA link delay:

$$\begin{aligned} D_{\text{link}} = ((\text{link latency}) \times F) + (\text{TX buffer depth}) + \\ (\text{RX buffer write-forward cycles}) \quad [\text{cycles}], \end{aligned} \tag{2.2}$$

where (link latency) is the time it takes for a TX-sent flit to reach RX, F is the operating frequency, (TX buffer depth) is the TX buffering delay, and (RX buffer write-forward cycles) is the number of cycles before received flits become available from the RX buffer.

Transmission overhead is the ratio of control flit to the total number of flits sent in one packet, and is defined in Equation (2.3). Transmission of more data flits per packet leads to a lesser overhead, which maximizes network bandwidth utilization. Since there is one control flit in one FC packet, then:

$$(\text{TX overhead}) = \frac{1}{1 + (\text{TX buffer depth})}. \tag{2.3}$$

To operate at a high rate, RX buffer depth must be sufficiently larger than the round-trip time and credit update delay [75]. For bursty traffic, this large allocation allows high link utilization. Equation (2.4) summarizes RX buffer depth requirement:

$$(\text{RX buffer depth}) > (D_{\text{link}} \times 2) + D_{\text{CU}}, \tag{2.4}$$

where $(D_{\text{link}} \times 2)$ is the round-trip time or the round-trip link delay, and D_{CU} is the CU frequency or the interval at which RX sends a credit upstream. Since D_{CU} is equal to the TX buffer allocation: (TX buffer depth) = D_{CU}, which has implications for performance and area, an upper bound is set for the CU frequency, which is:

$$D_{\text{CU}} <= (\text{link latency}). \tag{2.5}$$

2.3 Results and Discussion

2.3.1 Simulation of Inter-FPGA Backpressure Mechanism

To verify the interaction between credit counter and the design parameters (TX/RX buffer allocations and CU frequency), timing diagrams are shown in Figure 2.2 and Figure 2.3. Credit counter in the transmitter (TX) is initially set to the receiving (RX) buffer allocation and decrements whenever a data flit is sent. In this simulation, a link latency of 128 cycles is assumed. Based on Equation (2.5), the maximum D_{CU} is set to 128 ($D_{\text{CU}} = 128$), which is equal to (TX buffer depth).

Figure 2.2 shows TX0 from FPGA 0 continuously sends data stream to RX1 of FPGA 1, where no backpressure is applied from the receiver FPGA. This implies a constant throughput at RX1 buffer, which sends a credit every ($D_{\text{CU}} = 128$) buffer reads to update TX0 of RX1 buffer status. When credit is sent by RX1 to TX0, it takes a number of cycles (represented by the link latency), 128 cycles in this case, before it arrives at TX0, which in turn, updates its credit counter. As shown in Figure 2.2, the credit counter is always updated every 128 cycles, which implies the availability of RX1 buffer.

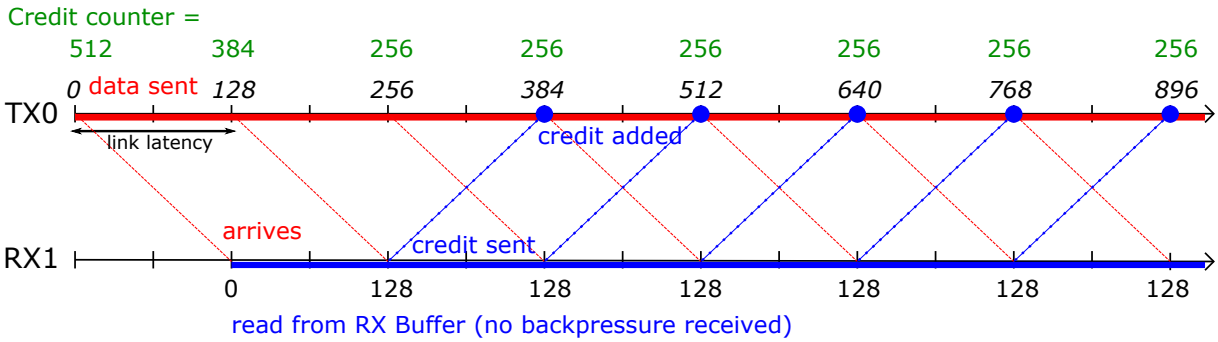


Figure 2.2: Timing diagram with constant RX buffer reads (no backpressure)

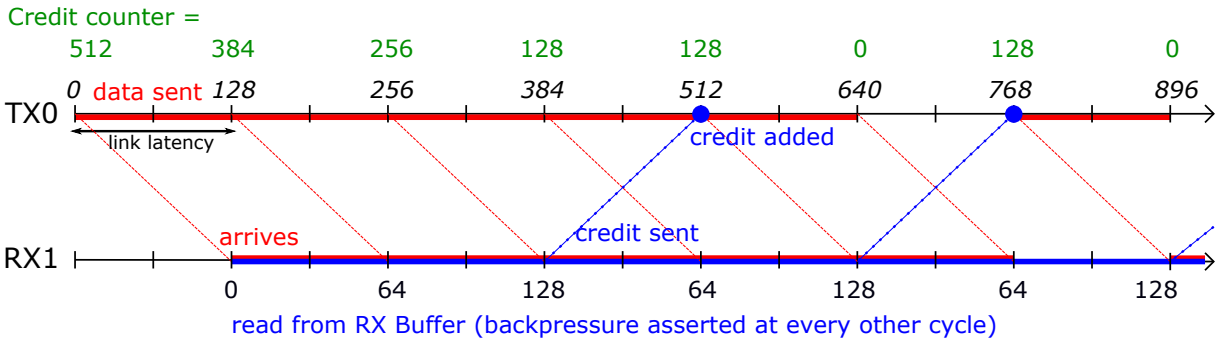


Figure 2.3: Timing diagram with intermittent RX buffer reads (with backpressure)

Figure 2.3, on the other hand, initially shows a high data transmission rate. However, the receiving end's RX1 buffer is simulated to receive backpressure signals at every other cycle from its application's sink port, which reads the buffer. This causes the RX1 buffer reads twice as long to reach ($D_{\mathrm{CU}} = 128$) compared with the other case, which delays the sending of credits to TX0. On the TX0's end, its credit counter eventually gets depleted due to its infrequent updates, which mimics the backpressure signal to stop transmission, as propagated from the RX1 buffer. In Figure 2.3, data transmission rate is shown to be balanced between the transmitting FPGA and the receiving FPGA. This demonstrates the correct assumption of setting an upper bound for D_{CU} in Equation (2.5).

2.3.2 Case Studies for Full and Half-Duplex Inter-FPGA Communication with High and Low Data Transmission Rates

The interaction between the communicating FPGAs in both full-duplex and half-duplex operations are investigated in this section. Here, the same design parameters are assumed: link latency is 128 cycles, $D_{\mathrm{CU}} = 128$, and TX buffer depth is 128.

Figure 2.4 shows a loop back configuration, where FPGA 0 sends data streams and FPGA 1 receives them, then sends it back. TX0 initiates transmission when its TX0 buffer is full. After 128 cycles of propagating the link, the header or control flit, along with a credit of the first FC packet arrives at RX1, followed by its succeeding data flits. After $D_{\mathrm{CU}} = 128$ RX1 buffer reads, TX1 begins sending back the data streams to RX0. For both communicating ends, the received credits are added to their respective credit counters to update the RX buffer status of the receiving FPGA. Since this case illustrates a high data transmission rate and there is no backpressure assumed from both RX0 and RX1 buffers, then continuous transfers for both directions occur seamlessly.

Figure 2.5, on the other hand, shows a low data transfer rate case for full-duplex channels, where intermittent data transfers are initiated by TX0. As with the other case, RX1 receives the incoming data streams, and transmits them back through TX1 when its TX buffer is full. No backpressure from application is likewise assumed in this case, and credits are still added as soon as they arrive. In this figure, a *force send* occurs when data flits occupy a TX buffer after a certain period (TX1 buffer in the figure), which guarantees the transmission of all TX buffer entries in the event that it never becomes full.

Meanwhile, Figure 2.6 shows a high transmission data rate in half-duplex data transfers. The same operations and interaction between the design parameters are observed here, except that only one direction is utilized (from TX0 to RX1). After $D_{\mathrm{CU}} = 128$ RX1 buffer reads, its corresponding TX1 sends credits encapsulated in control flits to RX0, in order to update FPGA0's credit counter. Likewise, credits are added when they arrive and since no backpressure from FPGA1's application is assumed, then data transmission

from TX0 to RX1 is uninterrupted.

Figure 2.7 shows half-duplex intermittent data transfers from TX0 to RX1. No backpressure is still assumed from the application in FPGA 1, so the transmission rate is balanced with the receiving rate. Sending credits by TX1 takes longer since RX1 buffer also takes a longer time to fill and reach $D_{\mathrm{CU}} = 128$ buffer reads.

In Figure 2.8, continuous transmission is performed by TX0, in which transmitted data streams are continuously received by RX1. However, a backpressure signal is assumed to be asserted by the application in FPGA 1, where RX1 buffer reads are stopped. This eventually results to depletion of credits in TX0 since no credit updates are sent by FPGA 1, which stops TX0 transmission, therefore implying backpressure propagation from the receiving FPGA.

2.3.3 Implementation and Evaluation

To obtain the inter-FPGA link delay for FC evaluation, a prototype platform with the network modules is implemented with a Terasic DE5A-NET board [82], which has an Intel Arria 10 FPGA with four high-speed, low-latency quad small form-factor pluggable (QSFP+) transceiver links, each with 40 Gbps data rate. For this initial evaluation, a point-to-point connection is implemented, where two transceiver links are bundled together to obtain a peak bandwidth of 80 Gbps or 10 GB/s. Here, the network modules for a direct network are considered and implemented, as shown in Figure 2.1, where point-to-point L1/L2 IP cores are utilized with FC modules on both communicating ends. In this experiment, a direct network is considered; however, the FC is universally designed to be compatible with both network types. Further discussion on the implementation and evaluation for an indirect network with an FC module is found in Chapter 4.

Based on the design parameters introduced in the previous section, the TX/RX communication buffer depths are selected. Figure 2.9 shows the effective link throughput when sending different data stream sizes with different TX buffer depths. For shorter data streams, smaller allocation has a higher effective throughput due to a smaller TX

buffering overhead. However, this overhead becomes negligible in longer data streams, where the three different buffer depths converged to an effective link throughput of 7.92 GB/s. Since the target is for a generalized FC design, TX buffer depth is set to 32. Meanwhile, point-to-point link latency is 365 ns, which is 82 cycles at $F = 225$ MHz. With this, $D_{\text{link}} = (82) + (32) + (4) = 118$ cycles (see Equation (2.2)), where TX buffer depth = 32 and RX buffer write-forward = 4 cycles. RX buffer depth = 512 is selected, which is sufficiently larger than $(118 \times 2) + (32) = 268$ (see Equation (2.2)). Here, $D_{\text{CU}} = 32$ cycles and is a statically chosen interval that satisfies Equation (2.4).

Figure 2.10 shows possible design parameters, where as expected, the network modules with TX buffer depth of $2^5 = 32$ and RX buffer depth of $2^9 = 512$ consume the least resources with only about 1.12% of the internal memory resources (Kbits), indicating a tiny footprint.

For an indirect network, it is expected that the link latency is slightly higher than the point-to-point's, due to the switch. In this case, TX buffer depth and CU frequency may be kept constant for both networks; however, the RX buffer may need to have a larger allocated size. This is further discussed in Chapter 4.

2.4 Conclusions

This chapter presents a lightweight and efficient hardware backpressure mechanism for inter-FPGA communication in direct and indirect networks, which is required for certain applications such as stream computing. This is achieved through a general-purpose hardware design of a credit-based flow control mechanism for communicating FPGAs, which results to a custom network protocol for additional reliability. The flow control design handles both synchronization, which ensures both communicating FPGAs' active status, and the sending of credits, which serves as backpressure mechanism for receiver status awareness. Furthermore, flow control autonomously supports half-duplex and full-duplex transfers.

To keep low-latency and high-bandwidth communication requirements, design parame-

ters of flow control mechanism were identified and explored. A point-to-point inter-FPGA connection is implemented with network modules, including flow controllers for both ends. Important design parameters such as the TX and RX buffer depths are evaluated in relation to performance and area consumption. For the store-and-forward TX buffer, which is equal to the credit update frequency, sufficient depth is necessary to reduce transmission overhead while minimizing additional latency. The RX buffer depth is based on the actual inter-FPGA's TX-to-RX link latency and must be more than its round-trip time and credit update frequency to allow high link utilization and reduced transmission time. However, it should have a minimum depth to allocate more area for the application. Equations (2.2), (2.3), and (2.4) summarize these important relationships. By utilizing the design parameters with the least resource utilization, the network link throughput averages at 7.92 GB/s.

Although the evaluation of the flow controller module is through the implementation and investigation of design parameters in a direct network, the same concepts follow for an indirect network. With a longer link latency, it is expected that the flow controller's RX buffer allocation for an indirect or switched connection will be larger than the point-to-point's (direct's), which affects area consumption. This is investigated further in Chapter 4.

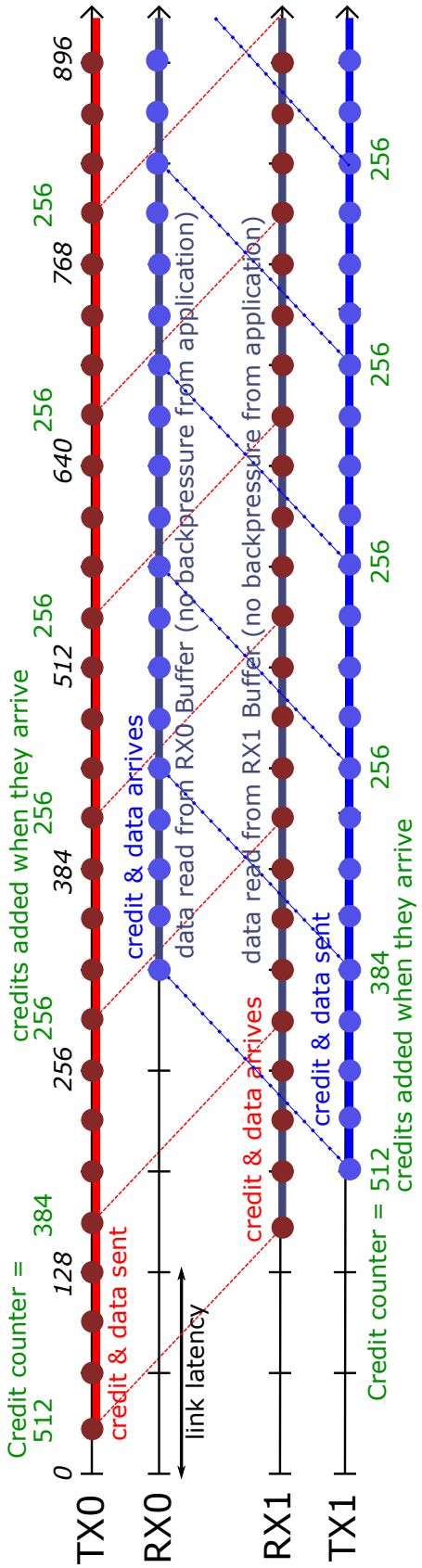


Figure 2.4: Full-duplex transmission with high data transfer rate

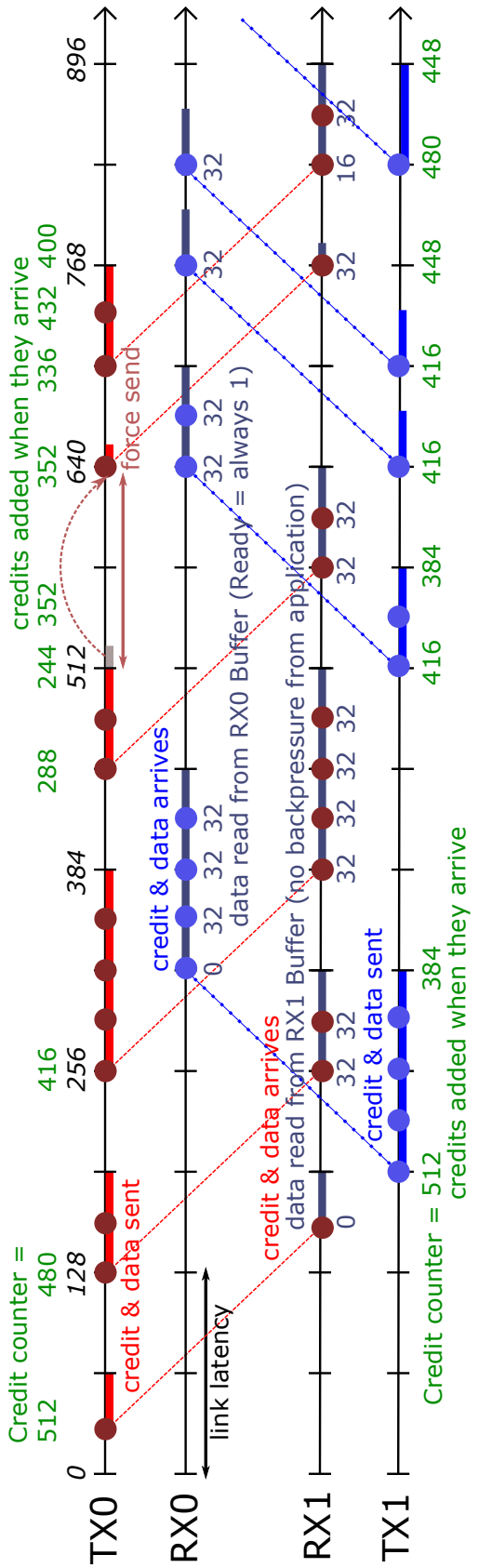


Figure 2.5: Full-duplex transmission with low data transfer rate

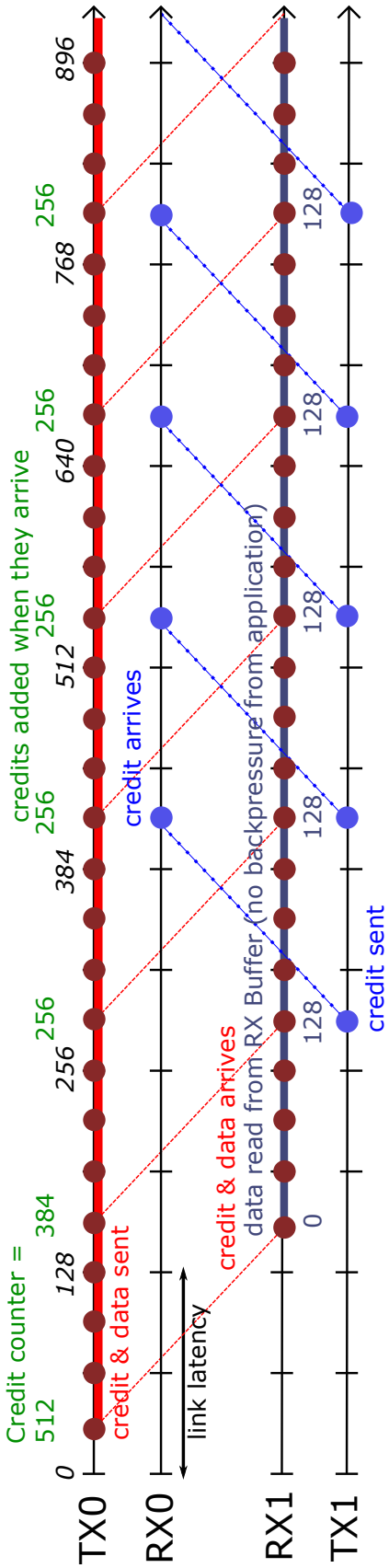


Figure 2.6: Half-duplex transmission with high data transfer rate

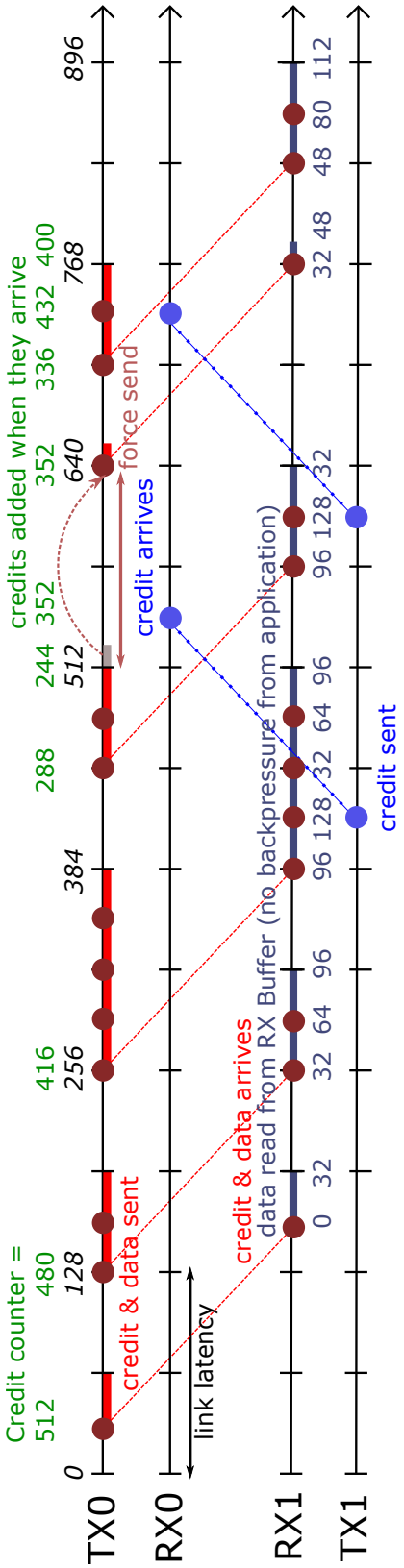


Figure 2.7: Half-duplex transmission with low data transfer rate

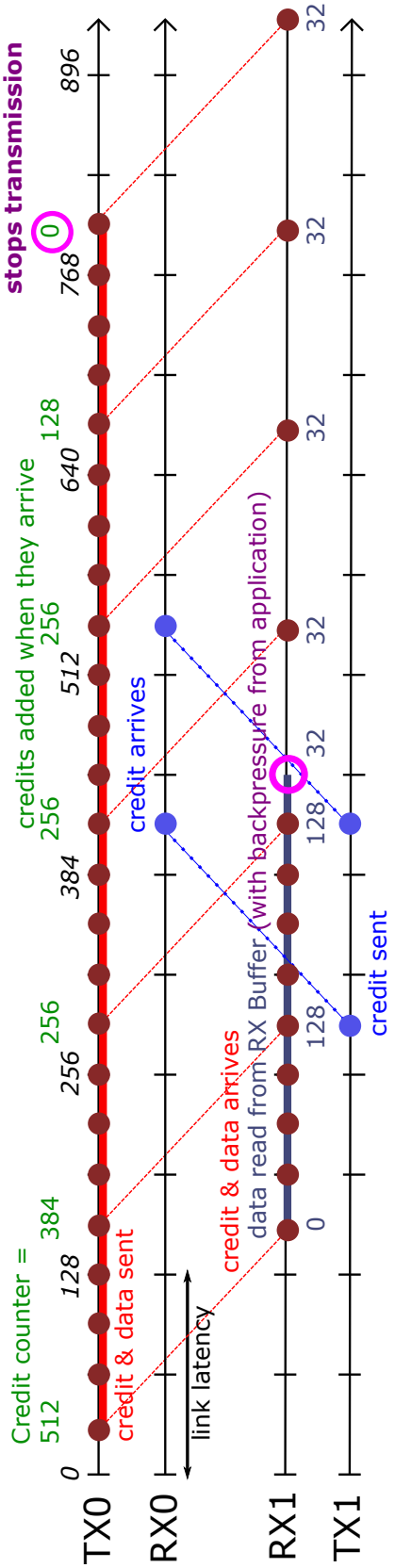


Figure 2.8: Half-duplex transmission with backpressure from receiver FPGA

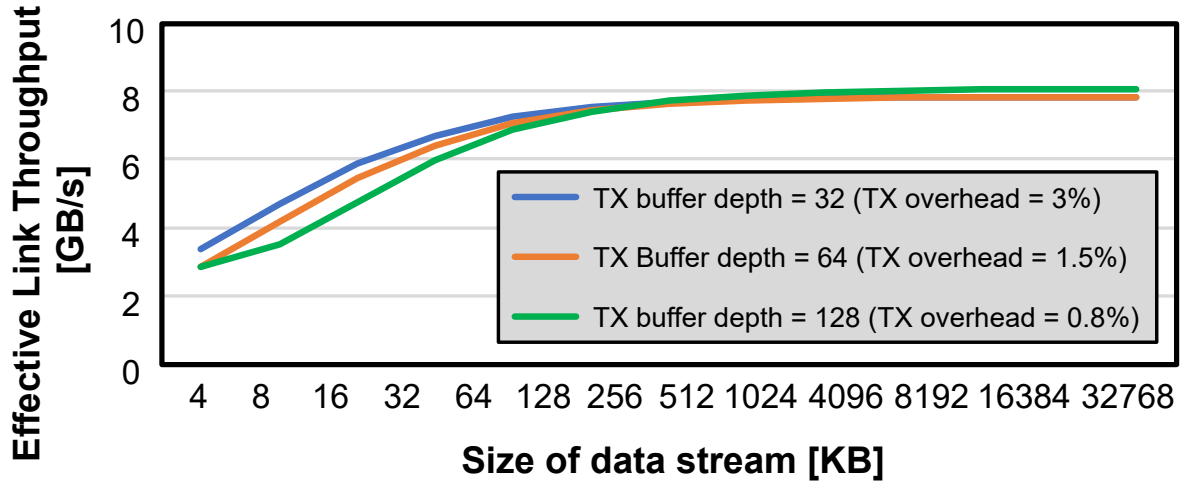


Figure 2.9: Effective link throughput

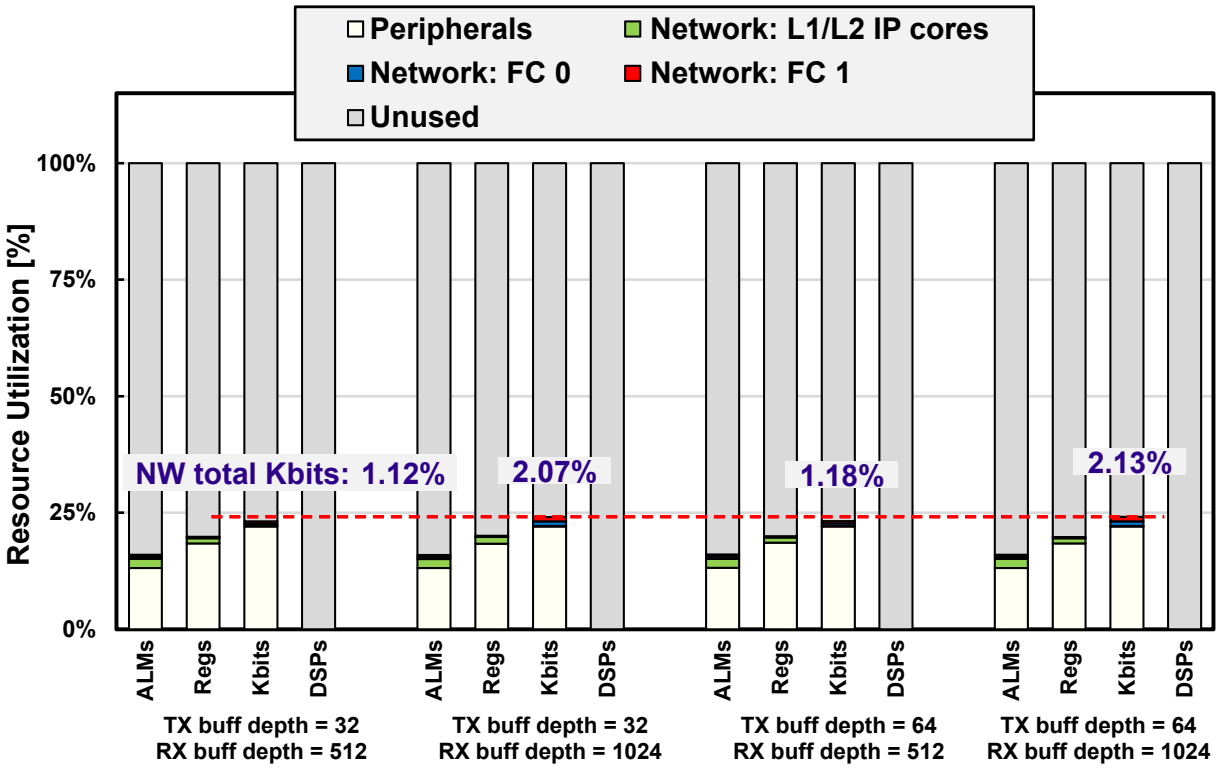


Figure 2.10: Resource Utilization using varied design parameters

Chapter 3

Direct Networks with High-Speed Transceiver Links

3.1 Introduction

FPGA's architecture is structured to support custom hardware designs, which could be optimized for efficient processing. At the same time, FPGA's flexibility allows easy connection to any other device via any physical standard or custom interface [81]. Aside from interfacing to other devices like external memory, DSP blocks, I/Os with high-speed transceivers, and other customized chips, FPGAs can be built to form clusters with low-latency communication, making them an appealing choice for scalable designs.

Even with all the advantages, FPGAs still have a lower memory bandwidth than GPUs, which ultimately, could cause a lower sustained performance. In addition, fitting custom numerical computations into an FPGA-mapped implementation has become a design challenge. Motivated by these drawbacks, previous studies such as [38, 83–85] have observed that stream computation is a suitable and scalable approach for stencil computing algorithms. This was achieved by creating a deep pipeline over successive iterations with a constant external memory bandwidth. This dedicated hardware stream computing design on the FPGA provides parallel computations due to regular memory access for data streaming. Furthermore, the deep pipelines also increase the number of

operations per memory access; therefore, fully utilizing the available bandwidth.

Increasing the pipeline depth of stream-based computation over multiple FPGAs could further scale up the computing performance. Unlike system-on-chips (SoCs) that need network-on-chips (NoCs) for intercommunication, FPGAs could be directly connected to each other through their high-speed transceiver links. Although present FPGAs have multiple high-speed serial transceiver channels, the bandwidth for inter-FPGA communication may still be insufficient compared to the external memory bandwidth, leading to a bottleneck. To address this matter, a previous work on hardware-based bandwidth compression [4, 86–88] is applied to enhance the communication bandwidth for floating-point data streams. The compression scheme is based on a prediction-based lossless data compression algorithm [89], which is essential in keeping the integrity of the data streams. It was also demonstrated in [4, 86–88] that a single-precision floating-point data can be compressed down to a quarter of its original size.

Since interconnecting FPGAs does not provide infinite scalability, the main goal of this chapter is to know its performance characteristics of a direct network by performing scalability analysis. A performance model for multiple FPGAs is presented, which considers parallelism options of the stream computing pipeline [41]. In this chapter, the network modules presented in Chapter 2 including the flow controller (FC) modules, are implemented on the FPGA cluster with a direct network. Performance characteristics of its inter-FPGA communication links, which interconnect FPGAs through their high-speed transceivers, is also investigated. In addition, the extended design space is explored, where a stream computing application is implemented and the performance model is verified. To achieve the final goal, a scalability analysis of the deeply pipelined FPGAs is performed by evaluating speedup against its parallel efficiency.

While Chapter 2 introduced and discussed interconnection network requirements for extending a stream computing pipeline over multiple FPGAs, this chapter focuses on the suitability of point-to-point FPGA interconnection for stream computing applications. The specific contributions of this chapter are:

1. Design and implementation of a scalable, deeply pipelined hardware platform with inter-FPGA direct network;
2. Investigation of point-to-point network's performance characteristics;
3. Performance model of stream computing on directly-connected FPGAs; and
4. Performance evaluation of stream computing on a direct network.

3.2 Design and Architecture

3.2.1 Stream Computing and Available Parallelism

Stream computing is an approach that can be effectively utilized to achieve high throughput even with constant and limited memory bandwidth. To obtain computational results from a custom computing region in an FPGA, data is read from an external memory and continuously supplied as inputs to the computing units. Figure 3.1a presents a generalized stream computing unit, which is a computing pipeline with floating-point operations of a custom application. It takes in W_{in} words of input stream and generates a computational output stream of W_{out} words synchronously every clock cycle. Figure 3.1a shows a unit pipeline, where it takes D_{pipe} cycles to produce the computational output, which is proportional to the number of pipeline stages. A deeper pipeline means more computational operations are performed at a constant throughput. One domain appropriate for stream computing is iterative stencil kernels.

Figure 3.1 also shows the available parallelism on the design space with the computational pipeline. Here, stream processing element (SPE) is defined to contain a single unit pipeline. With each input stream, an SPE computes for its corresponding single time-step output. When $W_{\text{in}} = W_{\text{out}}$ words of stream width, the computational output of an SPE can be connected as the next time-step input to a replicated SPE. Figure 3.1b illustrates cascading m SPEs to form a single deep pipeline, which enables multiple time-step computations with a single data stream. This is similar to loop unrolling approach,

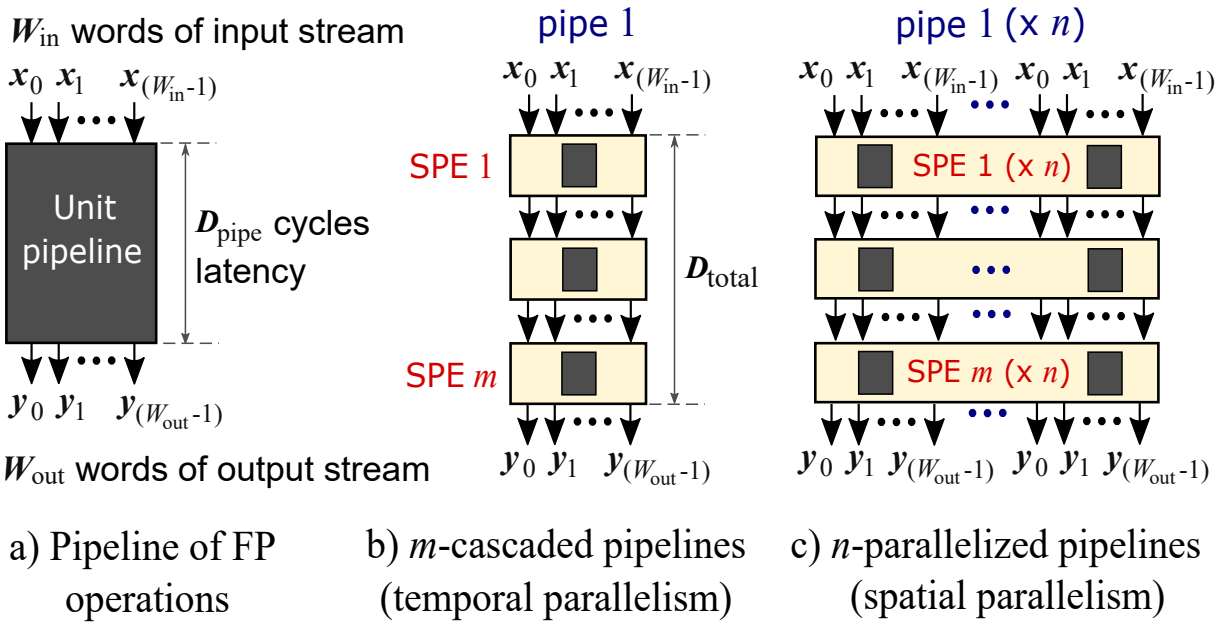


Figure 3.1: Generalized steam computing model with stream processing elements (SPEs)

which exploits temporal parallelism. This allows an increase in performance without increasing the number of memory accesses while concealing memory access latency. However, a deeper pipeline produces a large inefficient overhead and while computations are processed, the intermediate results are not stored to memory.

An SPE can also contain n-parallelized unit pipelines, which has n times higher performance in a single time-step, which exploits spatial parallelism. These pipelines can take in successive words from the input data stream, which made domain decomposition unnecessary for parallel computation. For the same number of operations utilized in temporal parallelism, there is lesser overhead in spatial parallelism since the pipeline depth is reduced. However, this approach increases the input stream bandwidth requirement because it needs an n times wider data stream. If available bandwidth is insufficient, stalls may occur, often leading to a decrease in performance. Figure 3.1c shows m-cascaded SPEs with n-parallelized unit pipelines.

In the FPGA's design space, SPEs can be either cascaded or parallelized to exploit fine-grained temporal or spatial parallelism, as presented in Figures 3.1b and c, respectively. This (n, m) SPE configuration is placed in the FPGA's user-defined logic region, called a computing core or simply a *core*, which contains custom implementation of a stream

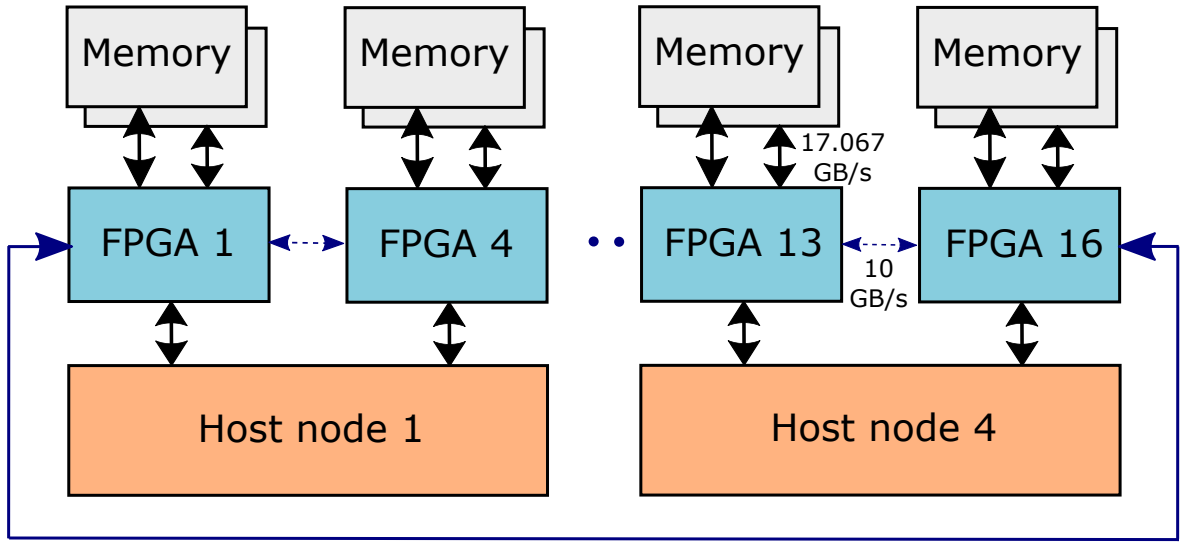


Figure 3.2: FPGA cluster in ring connection

computing algorithm.

Since there is a trade-off between temporal versus spatial parallelism, careful analysis should be done to balance the performance versus pipelining effect. Furthermore, a single FPGA has limited resources; thus, the number of SPEs is also limited. A workaround for this is to increase the number of SPEs over multiple FPGAs to further scale the performance.

3.2.2 Direct Networks for FPGA Clusters

Multiple FPGAs could be directly connected to form a cluster, which ideally scales the computing performance. A general overview of an FPGA cluster for a numerical stream computing approach is shown in Figure 3.2. It is composed of four host nodes with four FPGA boards each. When the (n, m) SPE configuration on a computing core is replicated over multiple FPGAs, an even deeper computing pipeline is implemented. There are several choices on how to connect the FPGAs, but a 1D ring is the most straightforward topology, where it allows the inter-FPGA transceiver links to be bundled together to double the available bandwidth and achieve a higher network throughput. A ring topology connection of FPGAs is formed by connecting them through the FPGA's high-speed links. Inside each FPGA, a pipeline of custom computing cores with corresponding SPEs is

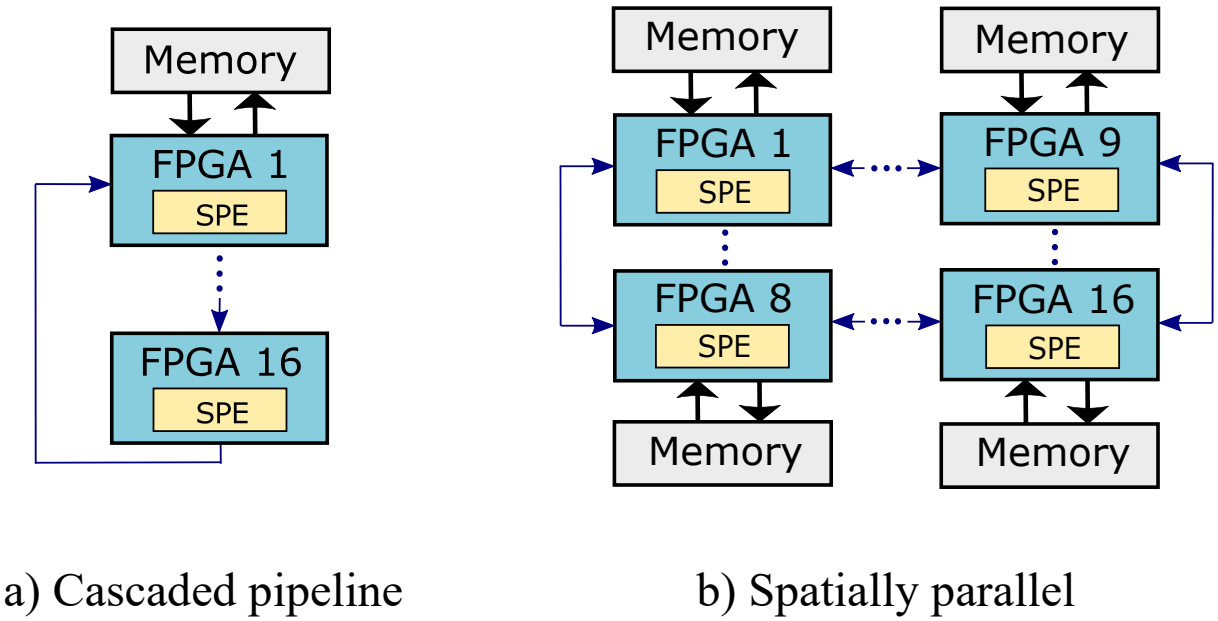


Figure 3.3: Available parallelism for FPGA clusters

implemented. This computing pipeline takes an input stream from memory and outputs a data stream as the computation result for a certain number of time steps. To utilize the FPGAs for parallel computation, there are generally two ways, as shown in Figure 3.3.

Figure 3.3a shows a cascaded pipeline approach of FPGAs, with one *master* FPGA and one or more *slave* FGPAs. The master FPGA has an exclusive access to the external memories and cascades the data stream down to the slave FPGAs. In addition, only the master FPGA is allowed to access its host node for system control. The slave FPGAs are implemented only with stream computing pipelines, in which the last slave device returns the output back to the master FPGA for storage control to memory. On the other hand, Figure 3.3b presents a cluster of spatially-paralleled FPGAs, which all have access to their respective external memories and host devices. Figure 3.3b is more advantageous in terms of the aggregate bandwidth of distributed memories, therefore, allowing multiple data streams to be processed simultaneously. However, since all the FPGAs will be receiving the data streams from their respective memories, there is a need to distribute the data accordingly prior to the start of computation, which may result to an inefficient data pre-processing overhead. Furthermore, a control mechanism should be implemented

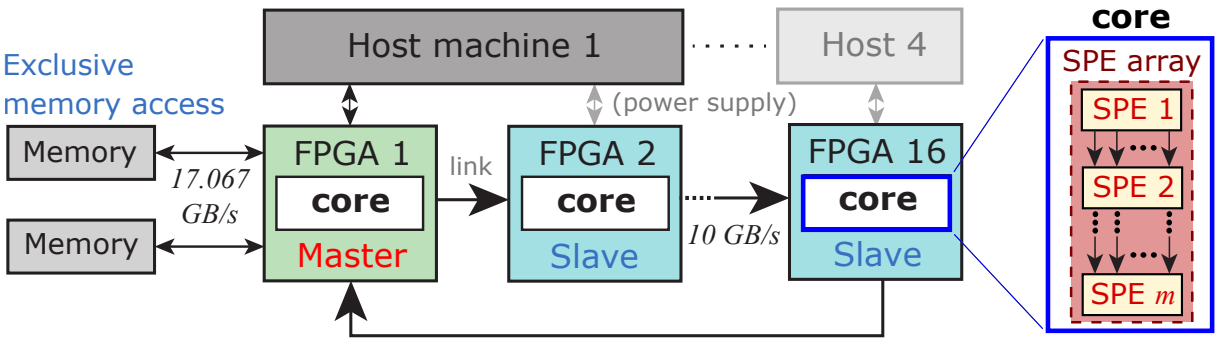


Figure 3.4: FPGA cluster in 1D ring topology showing SPEs in its computing cores

in this cluster to facilitate multi-FPGA operations, like DMA control and communication initialization commands, which could be complicated. Due to these disadvantages, this chapter specifically focuses on adopting the cascaded parallelism of Figure 3.3a due to the simplicity in the localization of computational data to a single memory. Since the master FPGA can exclusively access the centralized memory and its host system, the complicated control mechanism is eliminated and prior to computation, data streams are directly transferred to a single memory.

Through the PCI-Express (PCIe) interface, the pre-computational data streams are transferred from the host to the master FPGA's memory. The master and slave FPGAs are implemented with stream computing pipelines, which accepts the data streams through the master. After the master handles the initial computations, the cascaded slave FPGAs receive the resulting data streams and handle all the consequent computations before returning the final results back for storage to memory. The 1D ring topology is shown in Fig. 3.4, which illustrates the master-slave configuration on 16 clustered FPGAs connected to four host machines, with an option to scale the number of hosts and FPGAs if necessary.

3.2.3 Lossless Bandwidth Compression for Inter-FPGA Communication

FPGA clusters with wide stream computing pipelines often suffer from insufficient communication bandwidth. Likewise, the performance of multi-FPGA designs are often limited by the available bandwidth of its inter-FPGA communication. This is the case, particularly in cascaded FPGAs, since an entire data stream passes through the inter-FPGA path as if it were a part of a deep, huge pipeline. As a consequence, the bandwidth of inter-FPGA communication eventually becomes a bottleneck in stream computing. Since the performance of stream computing with FPGAs depend only on its throughput, it is therefore, almost impossible to improve the performance of multi-FPGA systems because the communication bandwidth limits the entire performance, even if the computing core itself has a high theoretical performance.

To solve this problem, a lossless data compression [4, 86, 88] could be applied to the data stream communicated between the FPGAs, which reduces the required bandwidth of data stream with a slightly additional latency [4,86,88]. The hardware-based lossless compression is particularly designed for effective and efficient numerical floating-point data streams. The compression algorithm exploits continuity of numerical data streams which always have spatial and temporal continuity to reduce an amount of data by employing prediction and subtraction processes. For the lossless compression, two more modules are included in the design, namely, *compression* (CHW) and *decompression* (DHW) hardware.

The input of CHW is connected to the output of SPEs directly, and the output of CHW is connected to the transceiver module, which transmits (TX) the data stream to the next FPGA. CHW receives a data stream from the SPEs which has multiple channels corresponding to variables of numerical computing. CHW compresses the input data stream and also bundles these channels into one data channel for transmission to the next FPGA. To exploit the available bandwidth of inter-FPGA communication, the output width of CHW is fixed, which is the same as the width of the communication channel. DHW receives (RX) a compressed and transmitted data stream from the previous FPGA.

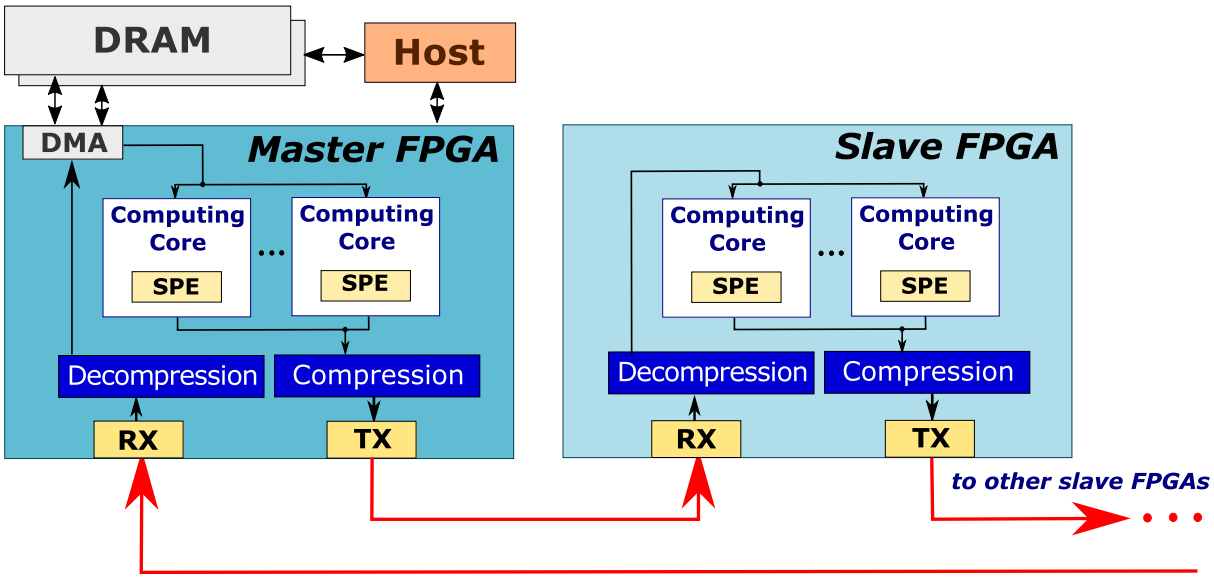


Figure 3.5: FPGA cluster in 1D ring topology with lossless bandwidth compression [4]

DHW then, distributes the received data to the correct channel in the original multi-channel stream, and then, data are decompressed to the original data in each channel. After the decompression, the data stream becomes the input to the next SPEs again.

Figure 3.5 shows the design of the master and slave FPGAs with the lossless compression algorithm hardwares. Every FPGA has both CHW and DHW to reduce the required bandwidth of the data stream. In this design, since the compression ratio of each CHW fluctuates because of lossless compression, the effect of compression also fluctuates. With this, the bandwidth improvement, which is equivalent to the compression ratio at a particular time can be expected. Moreover, the compression and decompression almost have no effect on computational time because CHW and DHW are pipelined and their depths are relatively tiny, whose pipeline depth is around 20. Therefore, this bandwidth compression improves the bandwidth of inter-FPGA communication with small hardware footprint and extremely small delay.

3.2.4 Performance Model

The performance model for stream computing with (n, m) SPE configuration [40, 41] is:

$$P_{\text{theory}}(n,m) = \frac{nmFO_{\text{pipe}}}{1+\frac{mD_{\text{pipe}}}{C_{\text{stream}}}} \frac{\min(B_{\text{mem}}, b_{\text{core}})}{b_{\text{core}}} \quad \text{[GFlops]}, \tag{3.1}$$

where O_{pipe} is the number of operations per unit pipeline, $D_{\text{pipe}}(n)$ is the pipeline depth of a unit pipeline, B_{mem} is available memory bandwidth, and $b_{\text{core}}(n)$ is the required computing core bandwidth. Here, $b_{\text{core}}(n) = nW_{\text{pipe}}F$, where W_{pipe} is the input/output width of a unit pipeline [bytes]. Since having n-parallelized pipelines requires n times wider data, the total number of stream cycles C_{stream} is inversely proportional to n: $C_{\text{stream}}(n) = \lceil N_{\text{grid}}/n \rceil$, where N_{grid} is the number of computational grid points to stream. In this dissertation, Equation (3.1) is extended to estimate the performance in the multi-FPGA platform. The contributing parameters are summarized in Table 3.1.

O_{FPGA} is introduced as the number of operations per FPGA, where $O_{\text{FPGA}}(n,m) = nmO_{\text{pipe}}$. To stream data with C_{stream} cycles, the total number of operations with M cascaded FPGAs is:

$$O_{\text{total}} = MO_{\text{FPGA}}C_{\text{stream}} = nmMO_{\text{pipe}}C_{\text{stream}} \quad \text{[ops]}. \tag{3.2}$$

Cascading FPGAs introduces the communication links into the model. Let B_{link} be the available link bandwidth. When there is insufficient available bandwidth, pipeline stalls will occur. Generally, this happens when either B_{mem} or B_{link} is less than b_{core}. Here, stall ratio r_{stall} is defined as the ratio of stall cycles to total cycles and utilization ratio $(1 - r_{\text{stall}})$ as the ratio of utilized cycles to total cycles:

$$r_{\text{stall}} = \begin{cases} 1 - \frac{B_{\text{link}}}{b_{\text{core}}} & , B_{\text{link}} < min(B_{\text{mem}}, b_{\text{core}}) \\ 1 - \frac{B_{\text{mem}}}{b_{\text{core}}} & , B_{\text{mem}} < min(B_{\text{link}}, b_{\text{core}}) \\ 0 & , \text{otherwise;} \end{cases} \tag{3.3}$$

$$(1 - r_{\text{stall}}) = \frac{\min(B_{\text{link}}, B_{\text{mem}}, b_{\text{core}})}{b_{\text{core}}}. \tag{3.4}$$

While the available external bandwidth B_{mem} is fixed, it should be noted that for utilizing the lossless bandwidth compression, the bandwidth of the compressed links B_{link} can vary with respect to the compression ratio r_{comp}, so $B_{\text{link}} = r_{\text{comp}} W_{\text{link}} F_{\text{link}}$, where W_{link} and F_{link} are the link's data width [bytes] and transmission frequency [Hz], respectively. On another side note, compression ratio is:

$$r_{\text{comp}} = \frac{\text{(size of original data stream)}}{\text{(size of compressed data stream)}} \text{ [4].} \tag{3.5}$$

The entire computation for a single data stream takes $(C_{\text{stream}} + D_{\text{total}})$ cycles, where $D_{\text{total}}(M)$ is the total propagation delay from start to end of the entire computing pipeline. Here, $D_{\text{total}}(M) = M(D_{\text{core}} + D_{\text{link}})$, where core delay $D_{\text{core}}(m) = mD_{\text{pipe}}$ and D_{link} is the inter-FPGA link delay, as introduced in Equation (2.2) on Chapter 2. Since pipeline stalls are anticipated with an insufficient available bandwidth, the total number of cycles for computation is:

$$\begin{aligned} C_{\text{total}} &= \frac{C_{\text{stream}} + M(D_{\text{core}} + D_{\text{link}})}{(1 - r_{\text{stall}})} \\ &= \frac{C_{\text{stream}} + M(mD_{\text{pipe}} + D_{\text{link}})}{(1 - r_{\text{stall}})} \quad \text{[cycles].} \end{aligned} \tag{3.6}$$

If compression is applied, its corresponding latency, D_{comp} is considered in the total propagation delays, where $D_{\text{total}}(M) = M(D_{\text{core}} + D_{\text{link}} + D_{\text{comp}})$. Correspondingly,

$$C_{\text{total}} = \frac{C_{\text{stream}} + M(mD_{\text{pipe}} + D_{\text{link}} + D_{\text{comp}})}{(1 - r_{\text{stall}})} \quad \text{[cycles].} \tag{3.7}$$

Finally, total performance $P_{\text{theory}}(M, n, m)$ is:

$$\begin{aligned} P_{\text{theory}} &= \frac{(\text{Total number of operations})}{(\text{Total computing time})} \quad [\text{GFlops}] \\ &= \frac{O_{\text{total}}}{C_{\text{total}}\left(\frac{1}{F}\right)} \quad = \frac{MFO_{\text{FPGA}}C_{\text{stream}}}{C_{\text{total}}} \\ &= \frac{nmMFO_{\text{pipe}}C_{\text{stream}}(1-r_{\text{stall}})}{C_{\text{stream}} + M(mD_{\text{pipe}} + D_{\text{link}} + D_{\text{comp}})} \\ &= \frac{nmMFO_{\text{pipe}}}{1 + \frac{M(mD_{\text{pipe}}+D_{\text{link}}+D_{\text{comp}})}{C_{\text{stream}}}} \frac{\min(B_{\text{link}}, B_{\text{mem}}, b_{\text{core}})}{b_{\text{core}}}. \end{aligned} \tag{3.8}$$

Based on Equation (3.8), the following scaling factors are identified. First, $nmMFO_{\text{pipe}}$ defines the peak performance with M cascaded FPGAs, where $nmFO_{\text{pipe}}$ is the peak for a single FPGA. On the other hand, performance degradation due to pipeline overhead is indicated by $M(mD_{\text{pipe}} + D_{\text{link}} + D_{\text{comp}})/C_{\text{stream}}$. This suggests that the overhead increases as the data stream size gets larger with respect to the total pipeline depth $M(mD_{\text{pipe}} + D_{\text{link}} + D_{\text{comp}})$. Since M FPGAs would also scale the total propagation delay, therefore, adding more FPGAs will likewise contribute to the pipeline overhead. Finally, $\frac{\min(B_{\text{link}},B_{\text{mem}},b_{\text{core}})}{b_{\text{core}}}$is the effect of insufficient available bandwidth, caused by either the links or by having n-parallelized pipelines in the core.

3.3 Results and Discussion

3.3.1 Implementation

The acceleration platform with master and slave FPGAs is shown in Figure 3.6. It is implemented with eight Terasic DE5A-NET boards [82] on two host machines. Each board includes an Intel Arria 10 10AX115N3F45I2SG FPGA, two DDR3-2133 SDRAMs, a PCIe Gen2 x8 interface, and four high-speed, low-latency quad small form-factor pluggable (QSFP+) transceiver links, each of which has a bandwidth of 40 Gbps. Other necessary peripherals include: two DDR3 controllers, four scatter-gather direct memory access (SGDMA) modules, hardware cycle counters, and dual clock FIFOs (DCFIFOs). Data-width converters (WidthConv) convert the required bit-stream width for the com-

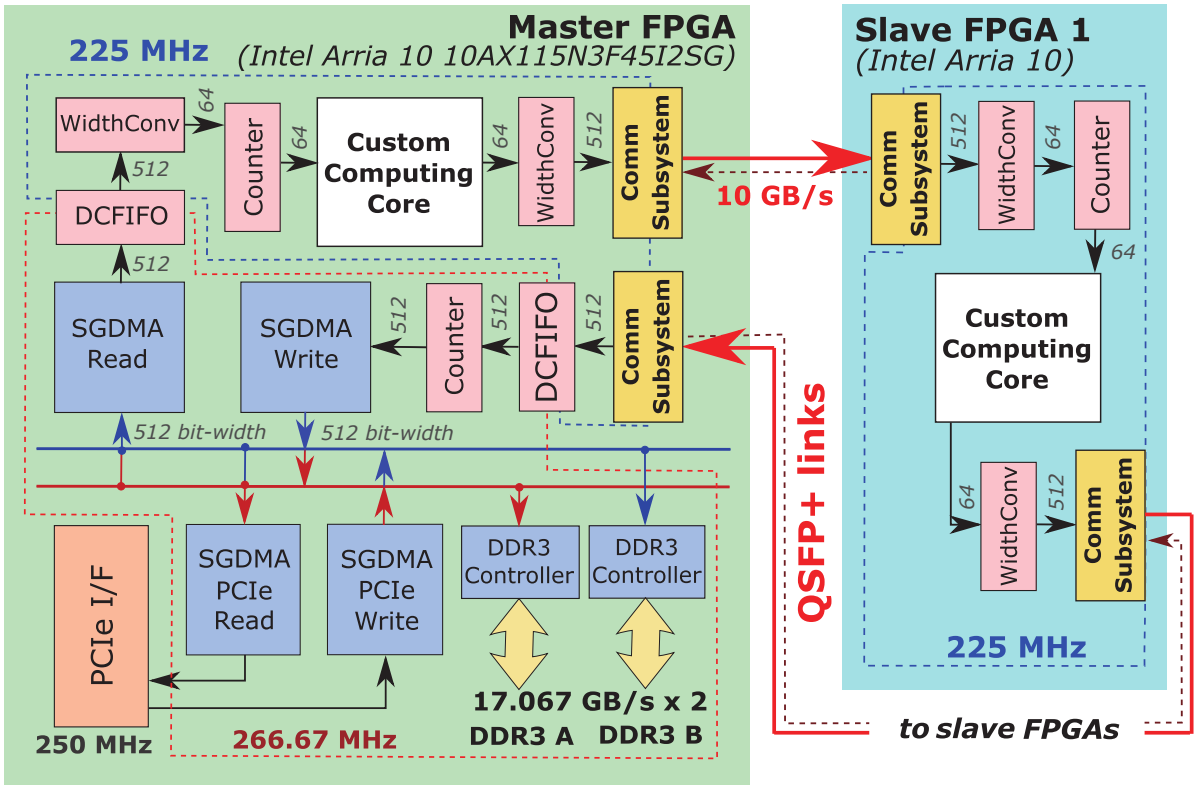


Figure 3.6: Acceleration platform with master-slave FPGAs

putation and communication modules. Different clock domains are also utilized: 250 MHz for PCIe, 266.67 MHz for DDR3 controllers, and an operating range of up to 225 MHz is available for the computing core. Each SDRAM has a peak bandwidth of 17.067 GB/s, while the two bundled 40 Gbps transceiver links claimed to reach 10 GB/s. However, as evaluated on Chapter 2, the sustained link throughput with the communication subsystems averages at 7.92 GB/s (see Figure 2.9). In a single FPGA, two sets of network modules are implemented (FC module and L1/L2 IP core), as introduced in Figure 2.1 on Chapter 2. For a direct network's L1/L2 IP core, Intel's proprietary Serial Lite III (SL3) protocol [90] for lightweight, low-latency, high-speed serial protocol for high-bandwidth applications is utilized.

For evaluation and benchmark purposes, a practical stream computing tsunami simulation [40] with real ocean-depth data is implemented on the proposed stream computing approach on a direct network, indicated by *custom computing core* shown in Figure 3.6. Its different SPE configurations are generated using a domain specific language-based

stream computing compiler (SPGen) [91], where all operations are in IEEE754 single precision floating-point format. Adders and multipliers are implemented using Intel IP cores on the floating-point-DSP blocks, while dividers and square root logic are generated using a floating-point generation tool, FloPoCo [92]. For design space exploration, SPE configuration arrays of $(n, m) = (1, 1), (1, 2), (1, 3), (1, 4), (2, 2), (1, 5),$ and $(1, 6)$ are selected, based on a previous work on tsunami simulation [40]. Using Intel Quartus Prime Pro 18.0, the entire acceleration platform design is generated.

As a result of master FPGA having more peripherals than its slave counterpart, it can fit only up to five SPEs, whereas the slave can accommodate six SPEs, where the limiting factor is the combination of both adaptive logic modules (ALMs) and DSPs. It is also noteworthy that the utilization of DSPs is solely by the computing core only, as shown in Figure 3.7. For $(n, m) = (1, 4)$ and $(2, 2)$, they used the same number of DSPs in their SPEs ($nm = 4$); however, there is a noticeable difference in other areas like ALMs, registers (Regs), and block memories (Kbits) since having n-parallelized pipelines allowed them to share the same stencil buffers in the SPEs [40].

3.3.2 Verification and Evaluation

The scalability and performance of the multi-FPGA platform with direct network using tsunami simulation's SPEs is investigated, in which one SPE with $nm = 1$ has 288 operations. Based on Equation (3.8), the peak performance $nmMFO_{\text{pipe}}$ is obtained with $F = 225$ MHz and $O_{\text{pipe}} = 288$. To obtain the theoretical sustained performance brought by degradation factors, $D_{\text{pipe}}(n) = 3099$ and 1808 for $n = 1$ and 2, respectively; while $D_{\text{link}} = 118$, as obtained in Chapter 2 (see Equation 2.2). For the available bandwidth, $B_{\text{mem}} = 17.067$ GB/s, and sustained $B_{\text{link}} = 7.92$ GB/s. The required bandwidth for tsunami simulation core is $b_{\text{core}}(n) = n \times 32 \times 0.225 = 7.2n$ GB/s, where $W_{\text{pipe}} = 32$ Bytes. In this evaluation, link compression is not applied.

Using actual ocean-depth data requires a sufficiently large N_{grid}, in this case, with 2581×2879 data grid, which is equivalent to $C_{\text{stream}}(1) = 7,430,699$ cycles. To initially

validate the model in Equation (3.8), a relatively smaller N_{grid} with $C_{\text{stream}}(1) = 116,104$ cycles is used, which is roughly 64 times smaller than the N_{grid} for tsunami simulation. Using up to $M = 4$ FPGAs, the peak, theoretical P_{theory}, and sustained performances of different SPE configurations were obtained, as shown in Figure 3.8. Using the hardware counters in Figure 3.6, the stall cycles and total computing cycles were measured for the sustained performance ratings. Figure 3.8 also shows the similarity ratio between theoretical and sustained performances, which is close to 100%, therefore, validating the model in Equation (3.8).

Figure 3.9 shows the performance evaluation of tsunami simulation with actual ocean-depth data using up to $M = 8$ FPGAs. In Figure 3.9, its peak, theoretical, and sustained performances are illustrated for $C_{\text{stream}}(1) = 7,430,699$. For $n = 1$, the available bandwidth is sufficient ($B_{\text{mem}} > B_{\text{link}} > b_{\text{core}}(1)$), making sustained performance close to its peak. In the case of $nm = 4$, two SPE configurations are implemented: $(1, 4)$ and $(2, 2)$, where $n = 2$ caused an increase of bandwidth requirement ($B_{\text{mem}} > b_{\text{core}}(2) > B_{\text{link}}$) when $b_{\text{core}}(2) = 2 \times 32 \times 0.225 = 14.4$ GB/s. This led to pipeline stalls resulting to a lower sustained performance. For $M = 8$ FPGAs, $(n, m) = (1, 5)$ obtained the highest performance with 2.5 TFlops, which is 98% of its peak performance. This shows that the total pipeline depth of $8 \times 5 = 40$ cascaded SPEs is sufficiently enough to accommodate the input C_{stream}, without being affected by the pipeline overhead. In the case where $C_{\text{stream}}(1) = 116,104$ cycles, as shown in Figure 3.8, the pipeline overhead is visibly reflected with the significant difference between the peak and sustained performances as the pipeline depth increases.

Figure 3.10 shows the speedup and parallel efficiency of the largest SPE configurations that can fit the master and slave FPGAs for efficient resource utilization: $(1, 5), (1, 4)$, and $(2, 4)$. The expected speedup is achieved for $n = 1$, when M FPGAs are increased. The differences among the three SPE configurations are significantly observed with more FPGAs due to the pipeline overhead caused by the fixed ocean-depth data grid. $(1, 5)$ has the best speedup but its parallel efficiency is slightly lower than $(1, 4)$'s, due to the pipeline overhead when the problem size of computation is maintained. $(2, 2)$ has the

lowest speedup rate and efficiency because of the insufficient bandwidth caused by having n-parallelized pipelines. This illustrates the performance model's prediction on the factors causing performance degradation, which in this case, is the bottleneck in the inter-FPGA links due to insufficient B_{link}.

With this, expanding the design space with multiple FPGAs supports further performance scaling. The key is finding the balance between the M, n, and m to achieve the best speedup and parallel efficiency rates. In the case of tsunami simulation, the large ocean-depth data grid allowed performance scaling when the pipeline depth is increased over multiple FPGAs. Ideally, n-parallelized pipelines would be the best approach since it would mean lesser computing cycles. However, this requires a larger bandwidth requirement, where B_{link} was not able to satisfy. Based on Figure 3.10, implementing $(n, m) = (1, 5)$ is the best option in terms of area, speedup, and efficiency, even though the latter is slightly lower than $(1, 4)$'s. With the currently utilized data grid, cascading up to $16(1, 5) = 80$ SPEs and $32(1, 5) = 160$ SPEs, have parallel efficiencies of 97% and 94%, respectively, which can still be acceptable rates. However, $M > 32$ FPGAs will bring a rapid rate of decreasing efficiency due to pipeline overhead.

3.3.3 Mitigating Inter-FPGA Communication Bottleneck

In this evaluation, mitigating inter-FPGA bottleneck is investigated by implementing the compression-decompression (CHW and DHW) modules. To complete the evaluation of this approach, another stream computing application, 2D9V Lattice Boltzmann method (LBM) [41] is implemented as the custom application with SPEs on the FPGA, which has a maximum pipeline depth of 16 stages. Since adding wider pipelines inherently increase the system's required bandwidth, the scaling effect of bandwidth compression is also investigated. The compression ratio per communication link is also measured to verify the increased performance. In addition, the strong scalability of the multi-FPGA approach is investigated using LBM-specific parameters on the performance model. In the 2D9V LBM case, $nm = 1$ has 131 operations and its peak performance $nmMFO_{\text{pipe}}$ and

sustained performance P_{theory} is obtained with $F = 175$ MHz, $O_{\text{pipe}} = 131$, $D_{\text{pipe}}(n) = 1550, 830,$ and 470 for $n = 1, 2,$ and 4, respectively, $C_{\text{stream}} = 720 \times 240 = 172800$ cycles, $B_{\text{mem}} = 17.067$ GB/s, $B_{\text{link}} = 7.92$ GB/s, and $W_{\text{core}} = 40$ bytes. The required bandwidth is $b_{\text{core}} = n \times 40 \times 0.175 = 7n$ GB/s.

Figure 3.11 shows the peak and sustained performances for $C_{\text{stream}} = 720 \times 240$, where no bandwidth compression is applied. Since the required bandwidth, $b_{core} = 7n$ GB/s, is less than the available bandwidth, for both B_{mem} and B_{link} when $n = 1$, then the available bandwidth for external memory and inter-FPGA communication is sufficient, causing no pipeline stalls. In the case of $(1, 16)$, the sustained performance is obtained as 318.828 GFlops, which is about 87% of its peak performance. However, when M starts to increase, the difference between sustained and peak performances starts to differ significantly. In the case of $(1, 256)$, the sustained-to-peak performance ratio decreases to just 29%. This is due to the inefficient pipeline overhead caused by the increasing number of FPGAs and a relatively short but fixed data stream. Figure 3.13 shows the same scenario, however, with a larger data stream, $C_{\text{stream}} = 720 \times 240 \times 16 = 2764800$ cycles, where at $(1, 256)$, the sustained performance is 92% of its peak performance. This goes to illustrate that having more FPGAs is more effective and efficient when the size of the data stream is sufficiently large.

In the case of $n = 2$ and 4, where $C_{\text{stream}} = 720 \times 240$ and without bandwidth compression, as shown in Figure 3.11, the parallel n pipelines increased the core bandwidth requirements, $b_{\text{core}} = 14$ and 28 GB/s, respectively. This results to an insufficient communication link leading to pipeline stalls. Stall ratio for $n = 2$ and 4, is 28.57% and 64.29%, respectively, resulting to reduced performance. In $(2, 16)$, the sustained performance is 448.79 GFlops, which is 61% of its peak, while in $(4, 16)$, the sustained performance is 435.99 GFlops, which is 30% of its corresponding peak. It is interesting to note that with $n = 4$, its sustained performance is a bit lower than $n = 2$, considering that it has more parallel pipelines. This intriguing case is believed to be caused by a higher stall ratio of $n = 4$ and the unproportional core delays D_{core} for the different n cores, contributing to the overall pipeline overhead.

To relieve the communication bottleneck, bandwidth compression is applied and the performance estimation is shown in Figure 3.12. In the case of 2D9V LBM application, it is measured that the compression ratios are 2.643 for $n = 1$, 2.392 for $n = 2$, and 1.853 for $n = 4$, where the maximum compression ratio is about 2.875. With bandwidth compression, the sustained performances of $(2, 16)$ and $(4, 16)$ has increased to 626.31 and 739.55 GFlops, which are 85% and 50% of their respective peak performances. In addition, the bandwidth compression also reduced the stalls: 39.05% stall ratio for $n = 4$ and no stalls for $n = 2$. Furthermore, when bandwidth compression is applied to a larger data stream, $C_{\text{stream}} = 720 \times 240 \times 16 = 2764800$ cycles, the sustained performances get closer to their peak performances, as shown in Figure 3.14, proving that pipeline overhead caused by the multiple FPGAs is weakened by the sufficiently long data stream.

3.4 Conclusions

This chapter presents the investigation of a direct network's performance characteristics for a stream computing FPGA cluster. Here, the design and architecture of a deeply pipelined stream computing platform on a ring connection of master and slave FPGAs with point-to-point connections is introduced. Temporal and spatial parallelism in the stream computing pipelines are explored to efficiently utilize the hardware resources. Fine-grained temporal parallelism is achieved by m-cascaded SPEs while spatial parallelism is explored by having n-parallelized pipelines. By cascading the SPEs on multiple FPGAs, a deeper computing pipeline with a vast design space is achieved to support further performance scaling.

Since scalability is finite, a performance model is also presented and validated by implementing a custom practical application on the stream computing prototype platform with eight cascaded FPGAs at 80 Gbps links. With this, a practical and efficient exploration of the vast design space can be achieved with the model. Tsunami simulation using real ocean-depth data set is implemented and evaluated on the master and slave FPGAs. The highest scaled performance on eight FPGAs is achieved with a single pipeline of

five cascaded SPEs $(n, m) = (1, 5)$, where 40 SPEs in a deep pipeline delivered a scaled performance of 2.5 TFlops and a parallel efficiency of 98%. In this chapter, scalability factors are identified. The peak performance is directly affected by the number of FPGAs and the operating frequency. On the other hand, performance degradation dictates the sustained performance, mainly caused by the pipeline overhead and bottleneck from the network.

The implementation of bandwidth-compressed links weakened the effects of communication bottleneck, therefore, allowing increased sustained performances, especially for longer data streams. In the case of $M = 16$ FPGAs, the best performance for a 2D9V LBM application is predicted at $n = 4$, when 739.55 GFlops of sustained performance is achieved through an enhanced communication bandwidth and with a moderate data stream size of 720×240 elements. When a longer data stream is fed to the multi-FPGA pipeline, the closer the sustained performance is to its peak, making the pipeline overhead relatively insignificant.

Table 3.1: Performance parameters

Parameters	Description	Unit
O_{total}	Total number of operations	[ops]
C_{total}	Total computing cycles	[cycles]*
F	Operating frequency	[Hz]
C_{stream}	Number of elements in data stream	-
m	Number of cascaded SPEs per FPGA	-
n	Number of parallel pipelines per FPGA	-
M	Number of cascaded FPGAs	-
O_{pipe}	Number of operations in a unit pipeline	[ops]
O_{FPGA}	Number of operations per FPGA	[ops]
B_{mem}	Available memory bandwidth	[Bytes/s]
B_{link}	Available inter-FPGA link bandwidth	[Bytes/s]
W_{link}	Width of inter-FPGA link	[Bytes]
b_{core}	Required computing core bandwidth	[Bytes/s]
W_{pipe}	Input and output width of a unit pipeline	[Bytes]
r_{stall}	Stall ratio	-
D_{total}	Total propagation delay	[cycles]*
D_{pipe}	Pipeline stages/delay in an unit pipeline	[cycles]*
D_{core}	Pipeline stages/delay in a core	[cycles]*
D_{link}	Inter-FPGA link delay	[cycles]*
D_{comp}	Compression latency/delay	[cycles]*

*All delays are measured in cycles at the same operating frequency F.

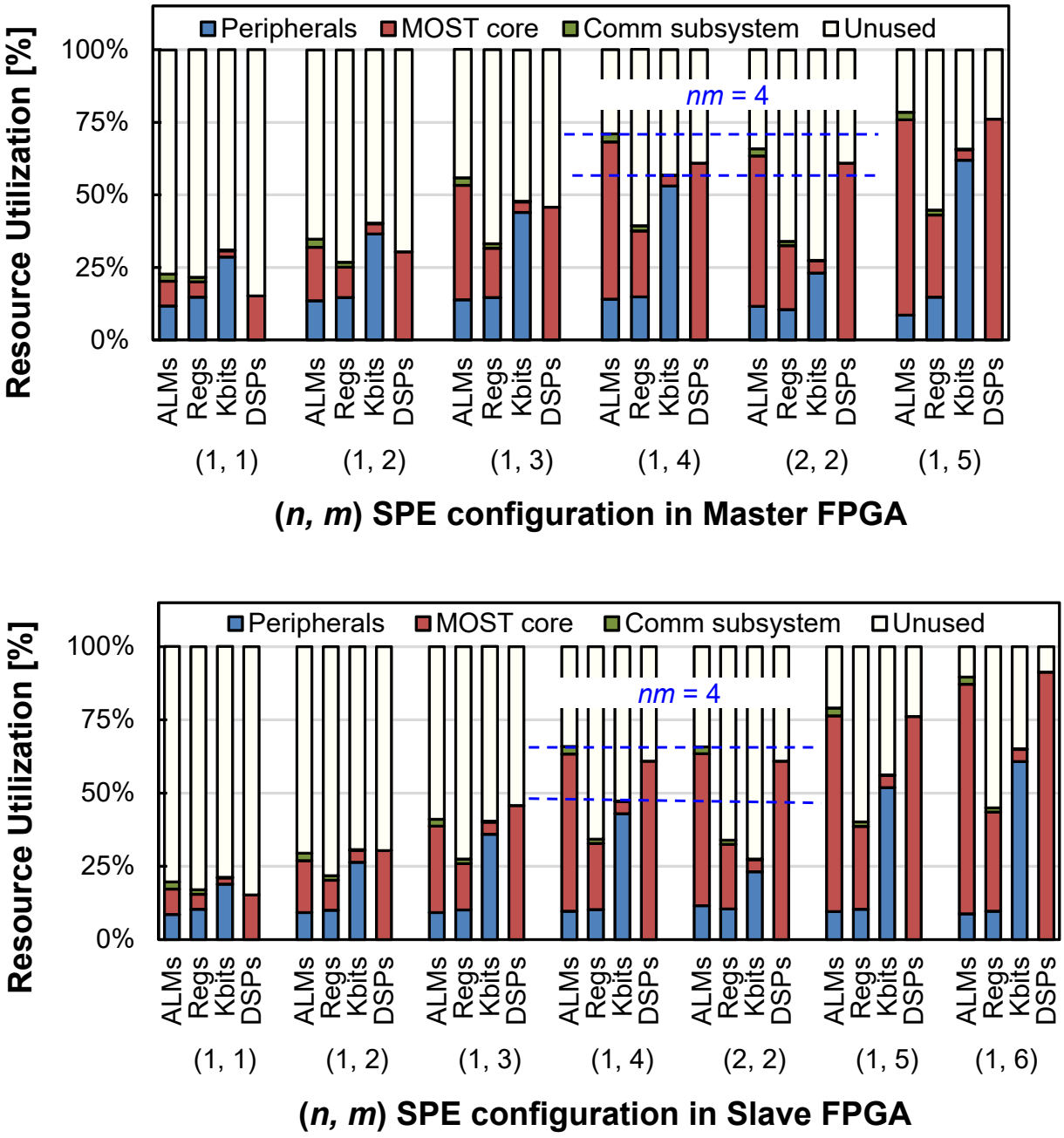


Figure 3.7: Resource utilization with different SPE configurations

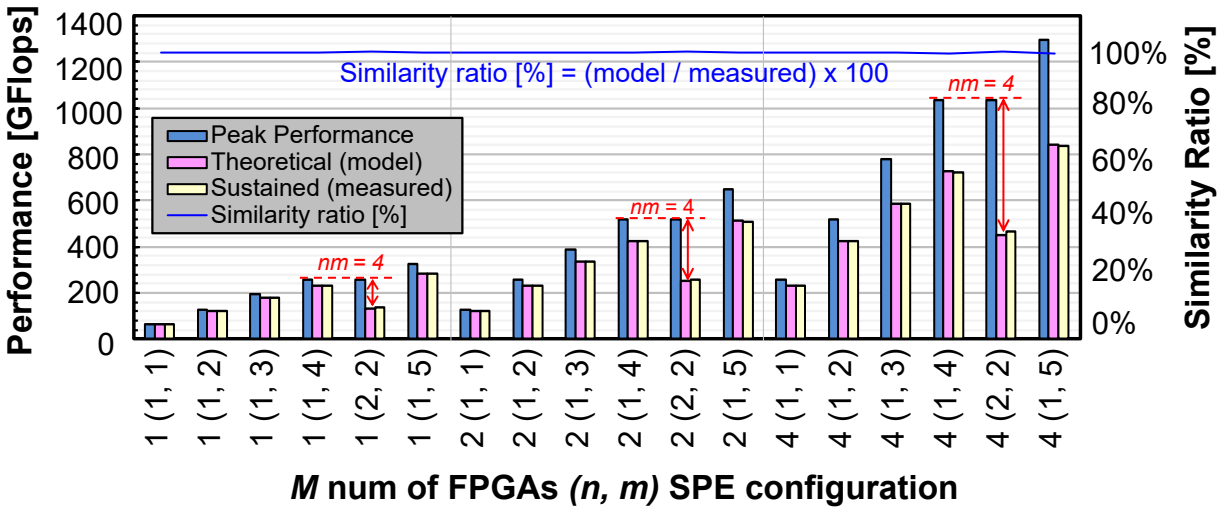


Figure 3.8: Validation of performance model with $C_{\text{stream}} = 116,104$ cycles

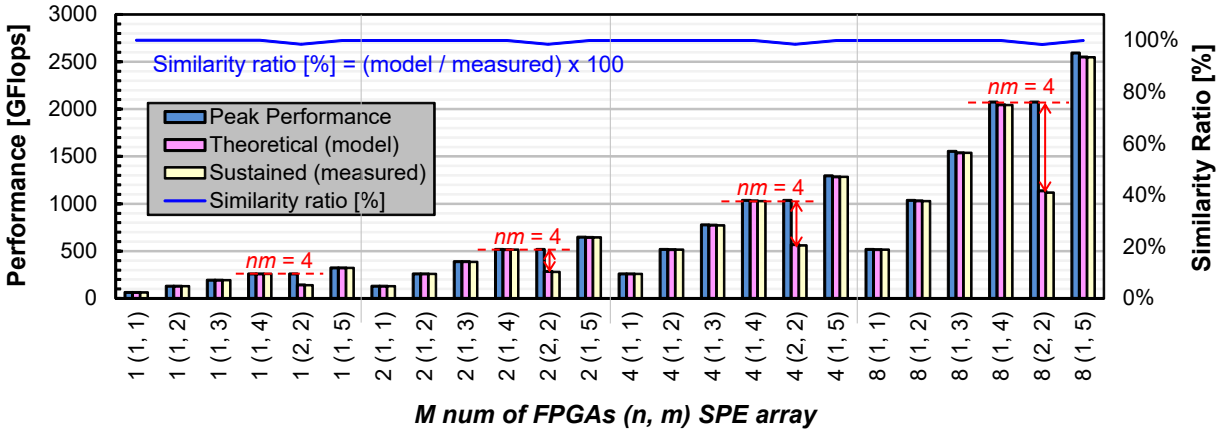


Figure 3.9: Performance evaluation of tsunami simulation

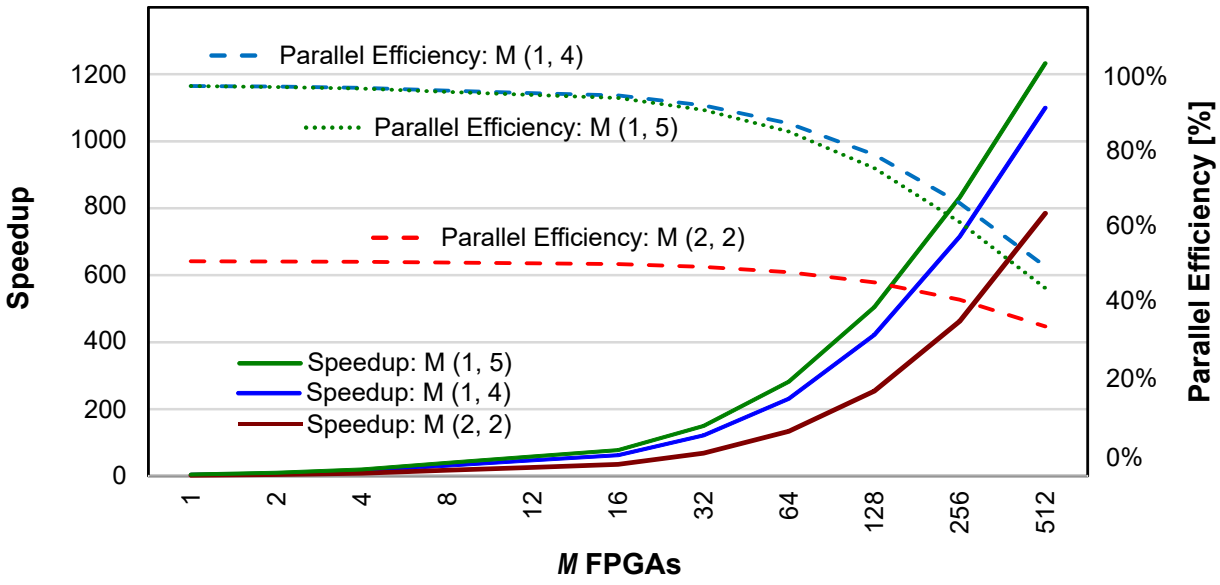


Figure 3.10: Speedup vs. parallel efficiency

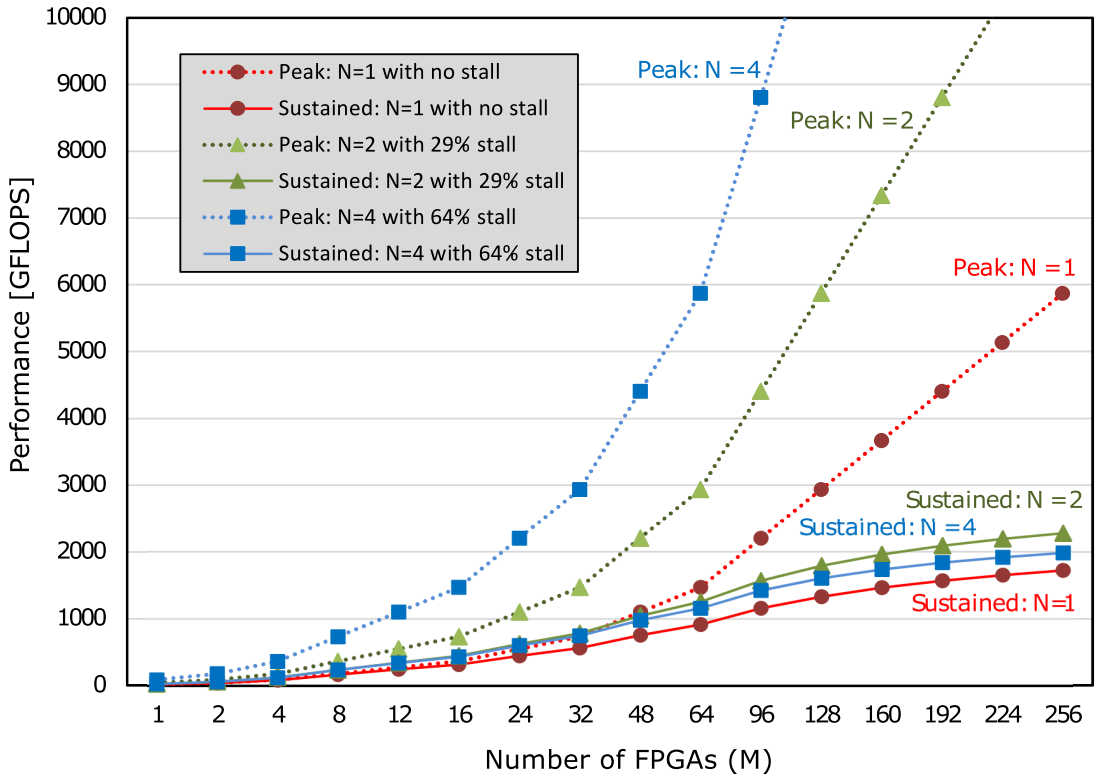


Figure 3.11: Estimated performance of 2D9V LBM without compression $C_{\text{stream}} = 720 \times 240$ elements

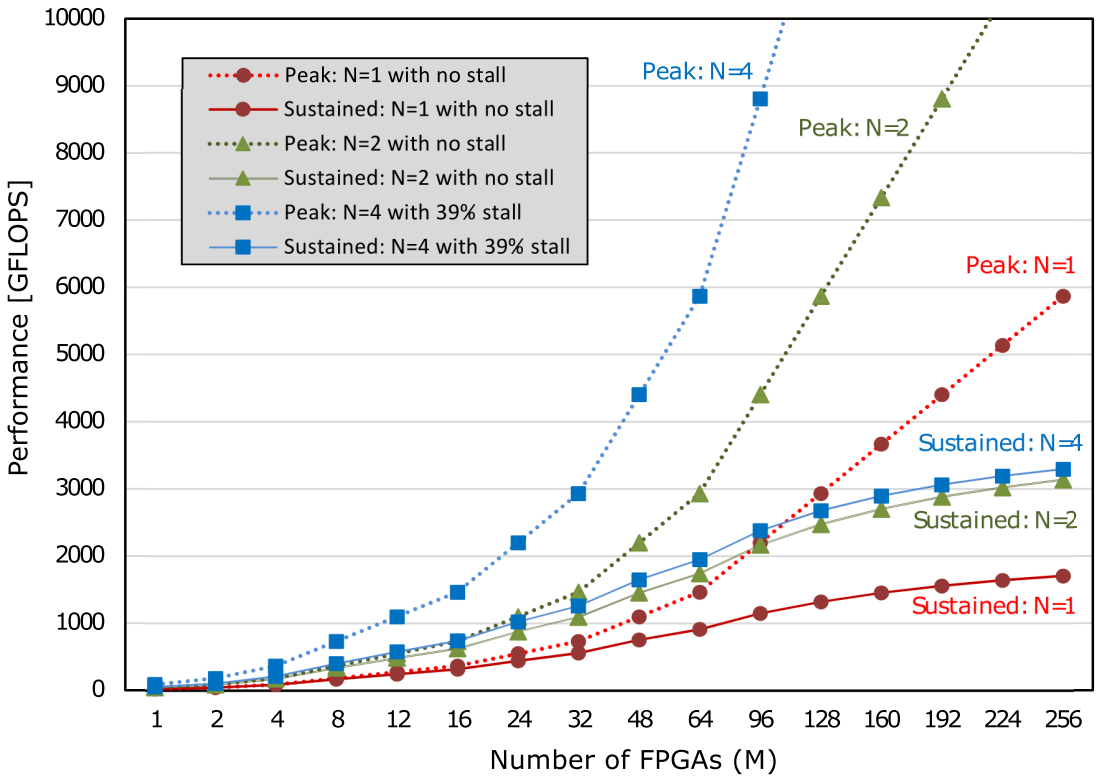


Figure 3.12: Estimated performance of 2D9V LBM with compression $C_{\text{stream}} = 720 \times 240$ elements

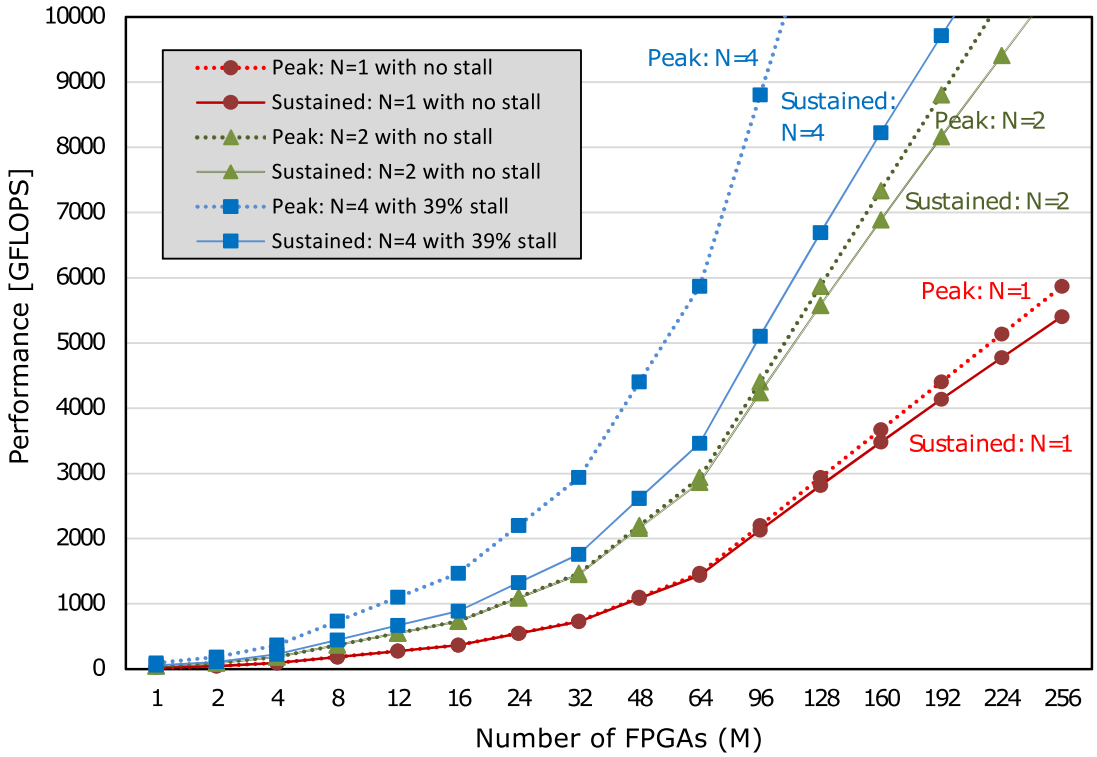


Figure 3.13: Estimated performance of 2D9V LBM without compression $C_{\text{stream}} = 720 \times 240 \times 16$ elements

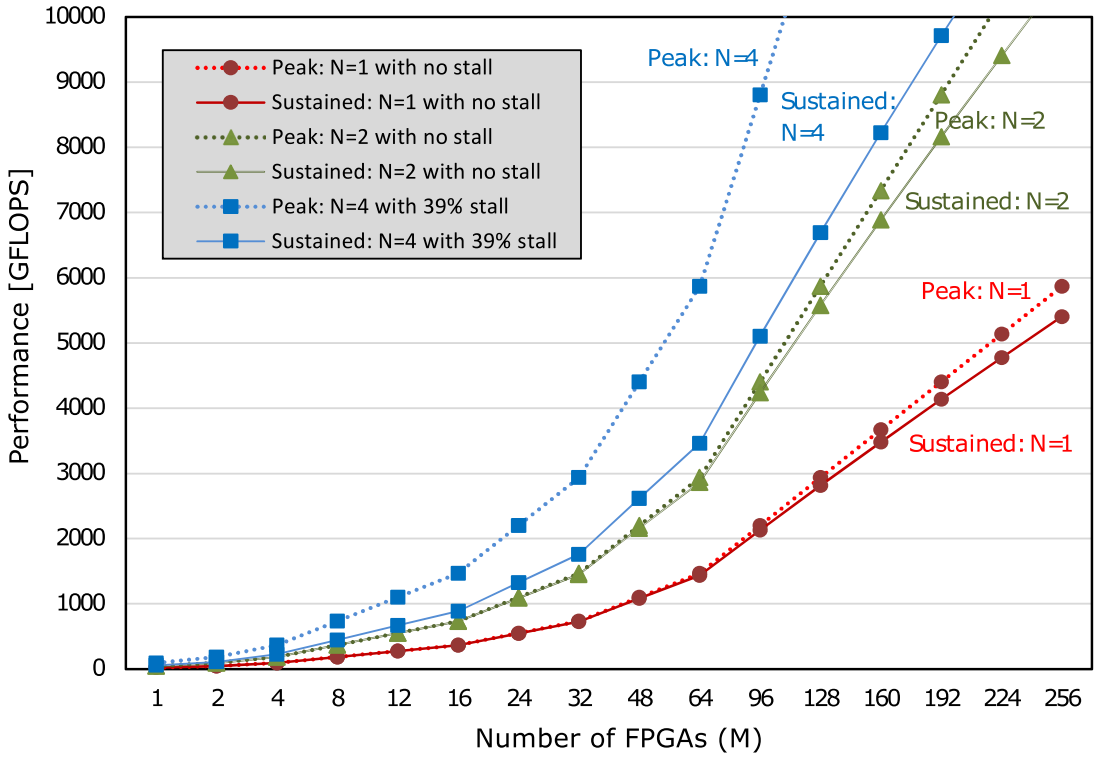


Figure 3.14: Estimated performance of 2D9V LBM with compression $C_{\text{stream}} = 720 \times 240 \times 16$ elements

Chapter 4

Indirect Networks with High-Speed Ethernet Switches

4.1 Introduction

Modern FPGAs have optimized interconnect fabrics, which allow low-latency communication in a dedicated network. Physically connecting FPGAs through their high-speed transceiver links in a direct network, as presented in Chapter 3, is a straightforward method. However, as the size of the cluster grows larger, the possibility of utilizing all FPGAs for a particular application decreases. This also requires multiple hop counts to reach a destination, which is inefficient with a large network diameter. On-chip FPGA routers could route data to its destination with more hops, but this may become inefficient especially with a large network diameter. In addition, a large-scale FPGA cluster could potentially service multiple applications that can be mapped strategically to maximize resource utilization, which requires network flexibility. Just as customized circuits in FPGAs is the key to performance gains, the interconnection network should ideally be scalable and flexible for its target applications.

To address these issues, an indirect network, where FPGAs are connected through switches seems promising. In this chapter, a scalable Ethernet-switched FPGA cluster is presented, where the transceiver links are physically connected to ports of high-speed

Ethernet switches. With offloaded switching or routing functions, this may mean shorter transmission time for a large network diameter due to lower hop counts. As a long-term standard, Ethernet is a good choice of protocol, which supports easier migration to higher data-rates and has adequate support for FPGAs through intellectual property (IP) cores. With its accelerating momentum towards 400+ Gbps within the next coming years [93], using Ethernet for the proposed switched network follows the design principle of independence, which facilitates incremental scaling and forward compatibility. However, its upper layer protocols such as TCP/IP are resource-heavy and expensive in hardware [54].

For the switched network, Layer 2 (L2) Ethernet is used. By supplying source and destination media access control (MAC) addresses in Ethernet frames, an FPGA could send data to a specified receiver FPGA; thus, providing flexibility without changing physical cabling structures. For some applications such as data flow stream computing, it is necessary to establish a connection-oriented *link* with backpressure support over the network. For usability, the custom credit-based network protocol with flow control presented in Chapter 2 is utilized in this switched network.

This chapter aims to know the performance characteristics of the proposed indirect network. By implementing the necessary data transfer hardware on FPGA, the communication performance is modeled, obtained, and compared with the direct networked FPGA cluster presented in Chapter 3, which uses Intel's proprietary Serial Lite III (SL3) protocol [90]. Figure 4.1 shows the connection-oriented inter-FPGA *links* and the necessary network hardware modules. To demonstrate scalability, the communication time of a streamed computing case in a large-scale cluster is estimated, along with its performance by modeling data stream traversal through a network in a ring connection. The following are this chapter's specific contributions:

1. Design of connection-oriented network with Ethernet switches for scalable and flexible FPGA clusters;

2. Investigation of performance characteristics and performance model of connection-oriented switched network;

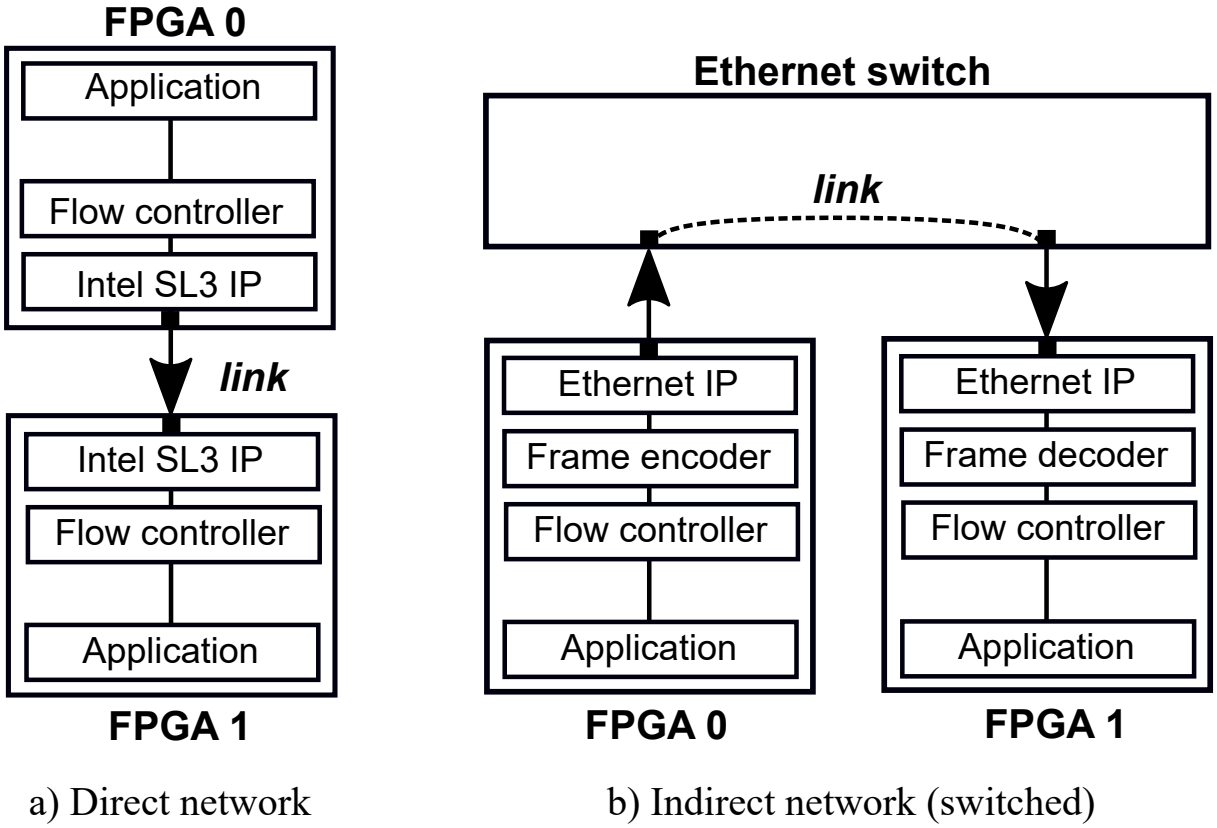


Figure 4.1: Connection-oriented links in dedicated FPGA networks

3. Performance evaluation of stream computing in indirect network; and

4. Demonstration that connection-oriented Ethernet switched network achieves equivalent performance to a point-to-point network for stream computing.

In this chapter, it was observed that the indirect network with 40 Gbps Ethernet (E40G) has obtained an effective network bandwidth of 4.41 GB/s, which is approximately 3% higher than a 40 Gbps SL3 point-to-point FPGA network. This result indicates a good communication performance of applications requiring high-bandwidth and large data transfers, such as stream computing, despite the overhead introduced by variable latency of a switched network. Generally, the scalability and flexibility features of the switched framework provide feasible groundwork for efficient high-level synthesis (HLS) compilers, which target to generate and map customized HPC applications in a large-scale FPGA cluster.

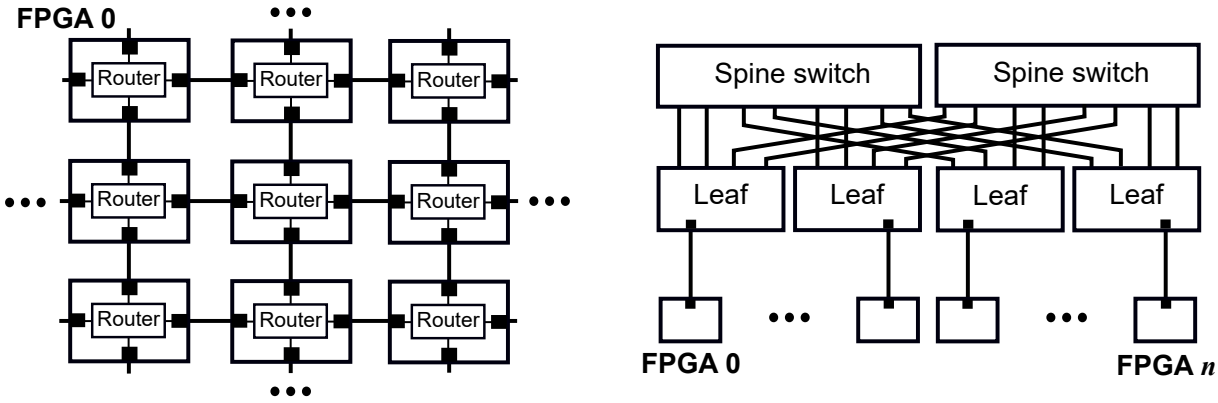


Figure 4.2: FPGA clusters when scaled

4.2 Design and Architecture

This section presents the proposed scalable indirect network framework with its design and architecture, including the custom protocol and model.

4.2.1 Indirect Networks for FPGA Clusters

A direct network based on point-to-point connection is popular for inter-FPGA communication because of its practical and extensive features. Since it allows close physical proximity between FPGAs, high-speed and high-bandwidth data transfers are often implied. A fully-connected network is ideal to keep low-latency transfers but unrealistic when scaled with more FPGAs. To minimize the network diameter, high-radix routers are employed but are usually constrained with the limited number of transceiver links. There is also a high-resource penalty for on-chip routers, which reduces FPGA area for application. Figure 4.2a shows a mesh/torus topology where their routers determine the datapath of a message. In comparison, the absence of a router in Figure 4.1a presented a point-to-point connection with a fixed datapath between two FPGAs.

An indirect or switch-based network enables the FPGA fabric to offload the routing or switching functions to a dedicated switch. Using a switch may introduce some additional latency but with a larger network diameter, there will be lesser hops to reach a destination

compared to a direct network. However, scalability is limited by the number of switch ports. To mitigate this, a multi-stage interconnection network may be constructed by cascading switches such as in a *leaf-spine* architecture [94, 95], (also known as *spine-leaf* or a two-tier fat tree/folded Clos network) shown in Figure 4.2b. In this two-layer network topology, FPGAs are connected to *leaf* switches. These switches are then fully meshed to a series of *spine* switches, which allows scaling with more FPGAs and provides better support for increased east-west traffic flows [96]. Unless two communicating FPGAs are in the same leaf switch, this mesh provides a fixed number of hops to a destination regardless of their physical location in the network, thus minimizing latency while keeping it at a predictable level even when scaled.

4.2.2 Ethernet-based Connection-oriented Links and Protocol

To establish connectivity from one FPGA to another in the switched network, L2 Ethernet is opted, which involves configuring source and destination MAC addresses on Ethernet frames. For some applications like stream computing, establishing this connection-oriented datapath with backpressure is necessary. Even without Layer 3 (L3) routing features, L2 MAC address switching is sufficient for the logical point-to-point connections. However, there is no physical inter-FPGA backpressure channel, which is necessary to propagate receiver availability towards an upstream transmitter. In this chapter, Figure 4.3 presents the necessary hardware modules for a single link in an Ethernet-based switching network, which includes the flow controller (FC), frame encoder and decoder, and Ethernet IP core for L2 and Layer 1 (L1) functions.

4.2.2.1 Ethernet L1 and L2 IP core:

As a standard protocol, there are existing off-the-shelf Ethernet IP cores with different incorporated layers and functionalities available for use. For the proposed indirect network, a low-latency 40/100 Gbps Ethernet IP core with L2 MAC and L1 PHY functions is selected, which follows the IEEE 802.3ba 2010 High Speed Ethernet Standard [97].

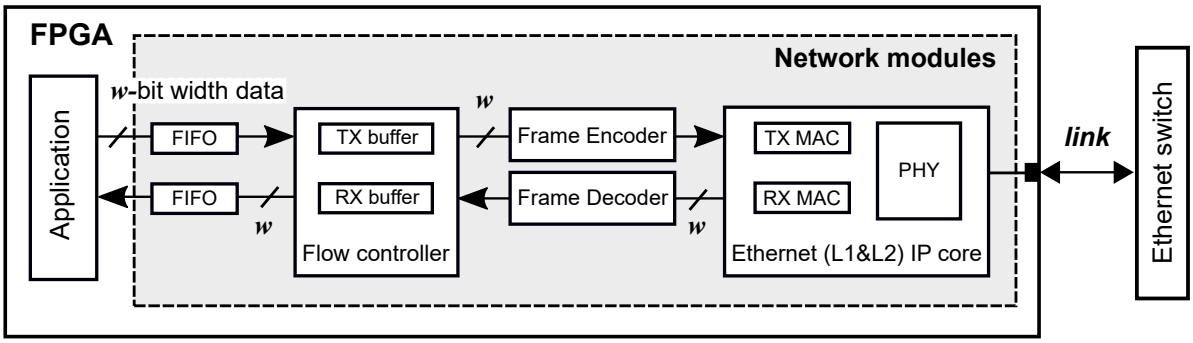


Figure 4.3: Network hardware modules for Ethernet protocol

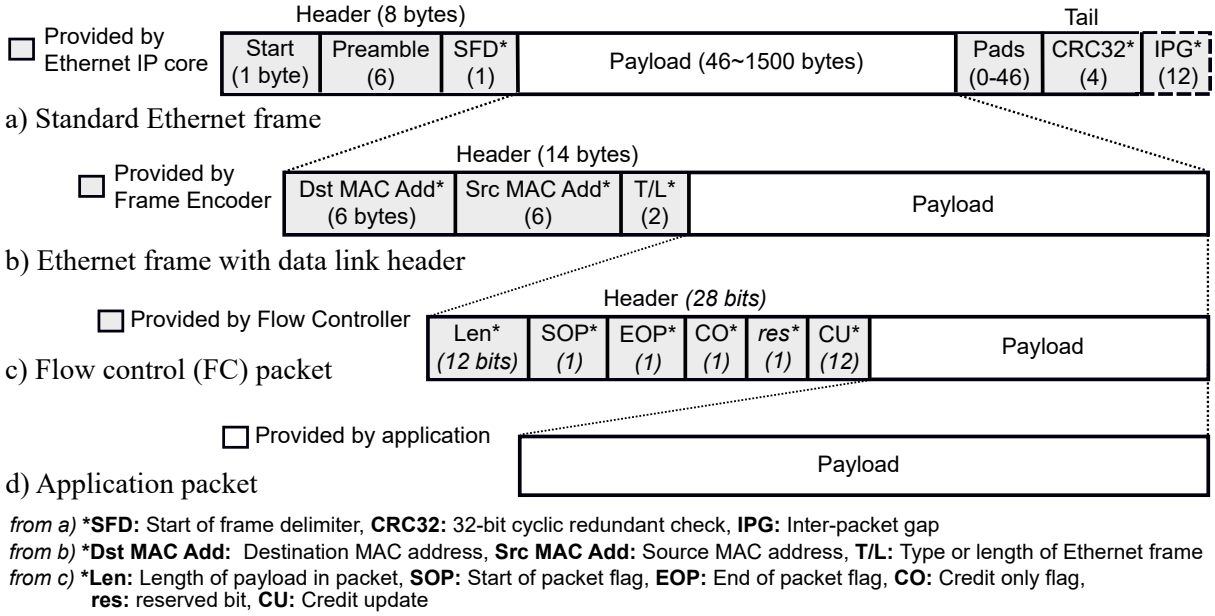


Figure 4.4: Protocol layers

This IP core supports frame encapsulation but without a data link header containing the MAC addresses. It also does not include any upper Ethernet layers, which is sufficient for stream computing requirements. Figure 4.4a shows its standard Ethernet frame output.

In the transmit direction, TX MAC accepts an w-bit width input frame and inserts a header and tail, as shown in Figure 4.4a. This is then passed to the PHY, which encodes it to serialized data for the FPGA transceiver links. In the receive direction, PHY passes deserialized data to RX MAC, which performs checksum calculations, removes the header and tail, and outputs the rest of the frame.

4.2.2.2 Frame Encoder and Decoder:

The frame encoder and decoder handle the flow of data between FC and Ethernet IP core. Essentially, the encoder's main function is to accept data from FC, inserts the data link header into an Ethernet frame, and passes it to the Ethernet IP core. As shown in Figure 4.4b, the encoder inserts the MAC addresses and the type/length (T/L) of the frame. In the receive direction, the decoder strips off the data link header before passing the payload to the FC module.

This module accepts a maximum payload of 1500 bytes, which is the standard maximum transmission unit (MTU) and can be changed as a parameter. A jumbo frame is also supported, as long as the Ethernet switch ports support handling a payload size greater than the standard MTU. However, when the encoder receives data in the form of a packet, which has start of packet (SOP) and end of packet (EOP) signals, the packet is considered a unit payload and is encapsulated directly with a header without other modifications.

4.2.2.3 Flow Controller (FC):

The FC module presented in Chapter 2 is utilized for the proposed switching network. The main purpose of this module is to provide receiver status awareness between two communicating FPGAs through the exchange of *credits*, which provides transmission reliability. It operates autonomously in either half or full-duplex data transfers. In this chapter, Ethernet compatibility is emphasized and supported through frame encapsulations handled by the encoder and Ethernet IP core.

FC receives data from the application, which could be divided into smaller packets composed of *data flits*. In each FC packet, a header is inserted. This is also known as a *control flit*, in which other information are embedded in order to reconstruct the original payload in the receive direction. The protocol is shown in Figure 4.4c.

As discussed in Chapter 2, the credit update (CU) frequency depends on the FC packet size, which is set as a parameter in this module. In order to embed the payload

length in the header, incoming data is placed in a store-and-forward transmitter buffer, FC TX buffer. To minimize induced waiting time for longer payload sizes, CU should be transmitted frequently enough by setting it to every D_{CU} flits. This means that a maximum FC packet sent to the frame encoder is $(D_{\text{CU}} + 1)$ flits including the control flit. This is equivalent to:

$$\text{(Maximum FC packet size)} = \frac{(w\text{-bit width})(D_{\text{CU}} + 1)}{8} \quad \text{[bytes]}, \tag{4.1}$$

which should satisfy the encoder's payload size requirements.

Another important parameter, as presented in Chapter 2, is the depth of the receiver buffer, FC RX buffer. In order to operate at a high rate, FC RX buffer allocation must be sufficiently larger the round-trip time plus CU frequency, D_{CU} [75].

4.2.3 Performance Model

In this section, a model is derived to estimate communication time as performance metric, which is dependent on various factors such as communication patterns and the network topology. To simplify and generalize the model, an FPGA-to-FPGA communication for both direct and indirect networks is derived. Table 4.1 lists the parameters affecting network performance.

For any point-to-point connection, a simple model to describe the total transfer time of a message or payload with m bytes is:

$$\begin{aligned} T_{\text{point-to-point}} &= T_{\text{L}} + \frac{m}{B} \\ T_{\text{point-to-point}} &= t_{\text{N}} + t_{\text{PL}} + \frac{m}{B} \quad \text{[s]}, \end{aligned} \tag{4.2}$$

where T_{L} is the total propagation latency [s] and B is the peak network bandwidth [GB/s], representing latency and streaming factors of a message transfer, respectively. Here, $T_{\text{L}} = t_{\text{N}} + t_{\text{PL}}$, where t_{N} is the node latency [s], also known as start-up latency, which refers to the message handling delays at the sending and receiving nodes, and t_{PL} is the

Table 4.1: Parameters for network performance model

Parameters	Description	Unit
m	Message (payload) size	[bytes]
B	Network link bandwidth	[GB/s]
t_{N}	Node latency (start-up latency)	[s]
t_{PL}	Physical link latency	[s]
l	Number of physical links	-
s	Number of switch hops	-
t_{S}	Average switching latency	[s]
T_{L}	Propagation latency	[s]
T	Total communication time	[s]

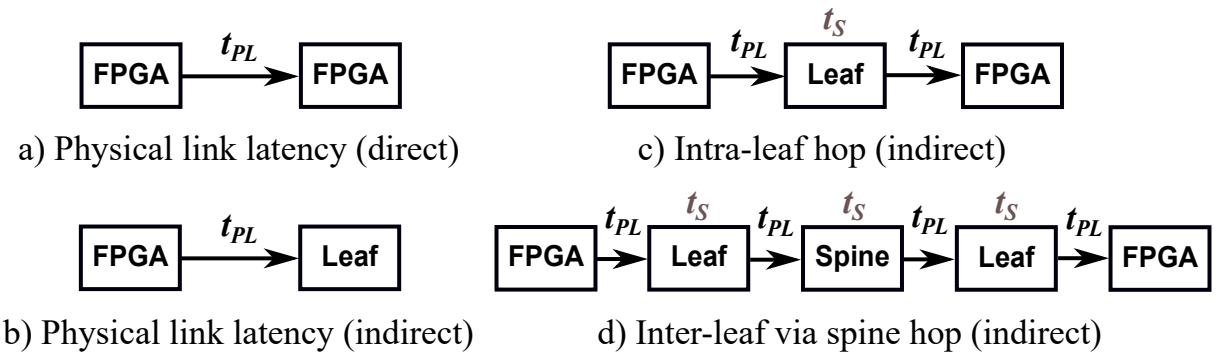


Figure 4.5: Network communication traversal

physical link latency [s], which refers to the time for a node to send and for another node to receive a zero-payload message across a network, as illustrated in Figure 4.5a. Here, an FPGA is defined as a *node*.

For the switched network, the node-to-node communication time is considered by breaking down the network datapath into parts, as shown in Figure 4.5b-d. In an indirect connection, an FPGA is connected to a leaf switch, as shown in Figure 4.5b. Figure 4.5c shows the communication pattern between two FPGAs in a single leaf switch, with its

transfer time as:

$$T_{\text{intra-leaf hop}} = T_{\text{L}} + \frac{m}{B}$$
$$T_{\text{intra-leaf hop}} = t_{\text{N}} + 2t_{\text{PL}} + t_{\text{S}} + \frac{m}{B} \quad [\text{s}], \tag{4.3}$$

since there are two physical links and the transfer included a single leaf switch hop with a switching latency, t_{S}, which is included in T_{L}. Therefore, for an intra-leaf switch data transfer, the communication time is:

$$T_{\text{intra-leaf hop}} = T_{\text{L}} + \frac{m}{B}$$
$$T_{\text{intra-leaf hop}} = t_{\text{N}} + lt_{\text{PL}} + t_{\text{S}} + \frac{m}{B} \quad [\text{s}], \tag{4.4}$$

where l is the number of physical links.

Figure 4.5d presents the FPGA-to-FPGA transfer in separate leaf switches, which involves a spine switch hop. Assuming this is a fully non-blocking full-bandwidth leaf-spine topology with no contention, then its transfer time is:

$$T_{\text{inter-leaf hop}} = T_{\text{L}} + \frac{m}{B}$$
$$T_{\text{inter-leaf hop}} = t_{\text{N}} + 4t_{\text{PL}} + 3t_{\text{S}} + \frac{m}{B} \quad [\text{s}], \tag{4.5}$$

since there are four physical links and three switch hops, assuming the same switching latency for both leaf and spine. To generalize this inter-leaf switch pattern, the communication time is:

$$T_{\text{inter-leaf hop}} = T_{\text{L}} + \frac{m}{B}$$
$$T_{\text{inter-leaf hop}} = t_{\text{N}} + lt_{\text{PL}} + st_{\text{S}} + \frac{m}{B} \quad [\text{s}]. \tag{4.6}$$

Consequently, the effective network bandwidth for both network types is:

$$B_{\text{effective}} = \frac{m}{T} = \frac{m}{T_{\text{L}} + \frac{m}{B}} \quad [\text{GB/s}], \tag{4.7}$$

where T is the total communication time.

4.3 Results and Discussion

In this section, the performance characteristics of the switched network is investigated and compared to a direct network. Resource utilization, latency, and effective network bandwidth of the connection-oriented links are obtained. By applying the measured parameters, the model is used to evaluate scalability.

4.3.1 Implementation

For fundamental evaluation, the network hardware modules are implemented on a Terasic DE5A-NET FPGA board [82], which includes an Intel Arria 10 FPGA. There are four quad small form-factor pluggable (QSFP+) transceiver ports, but only two are utilized for the experiments. For each port, an instance of the network modules is implemented. For the Ethernet IP core, Intel's Low Latency 40 Gbps Ethernet IP core (E40G) [98] is selected to match the tranceiver's 40 Gbps Attachment Unit Interface (XLAUI). As per E40G IP's specification, Avalon Streaming (Avalon-ST) interface [99] is used with a $w = 256$-bit width datapath for the network modules. To complete the indirect network setup, a 16-port Mellanox SN2100 Open Ethernet switch [100] is used, with its ports configured to 40 Gbps in order to match the data rate of Arria 10 FPGA links.

Two transceiver ports with their own direct network modules are prepared on another DE5A-NET board, which includes an FC module connected to a 40 Gbps SL3 IP core [90] per port, as shown in Figure 4.1a. Unlike in Chapter 2 and Chapter 3, the transceiver links in this setup are unbundled. For a fair comparison, 1-meter passive copper QSFP+ transceiver link cables are used for both network types and utilized the same

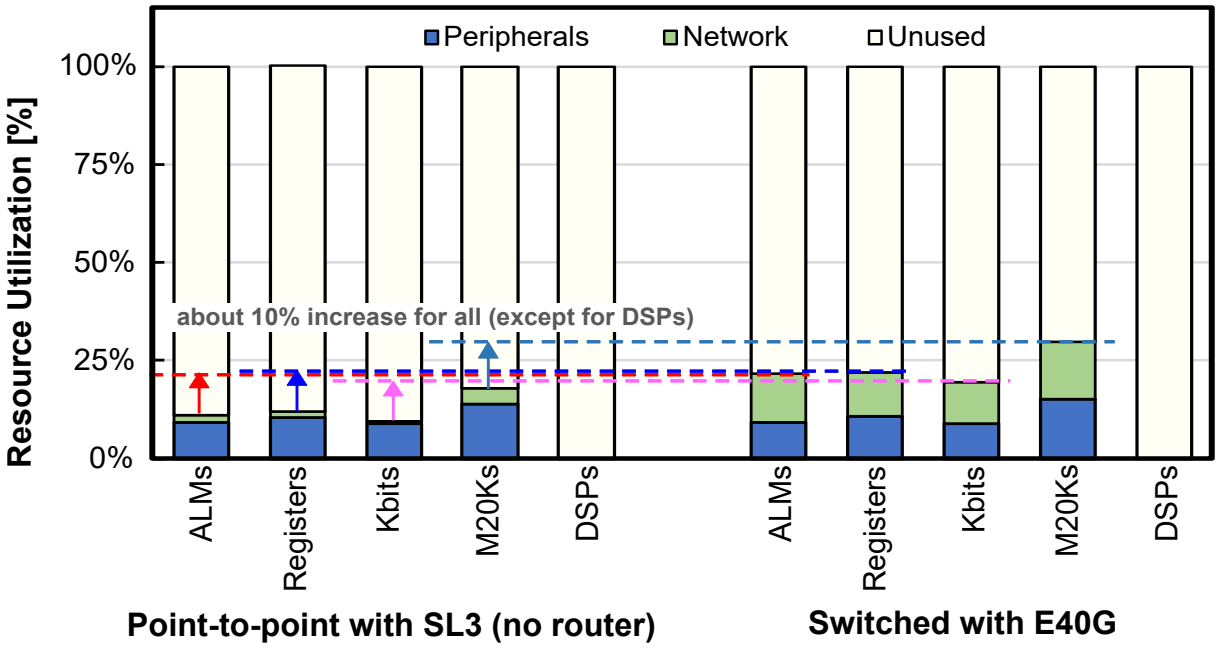


Figure 4.6: Resource utilization of SL3 and E40G Ethernet modules

cross-platform FC module for both SL3 and E40G setup.

For FC buffer allocations, TX buffer has a depth of 32 flits, where the CU frequency is set to send a credit every $D_{\text{CU}} = 32$ flits, as discussed in Chapter 2. Using Equation (4.1), the maximum FC packet size sent to the frame encoder is $(256)(32 + 1)/8 = 1056$ bytes, which satisfies the frame encoder payload size requirements. To fully maximize the network bandwidth, this may be increased to 1500 bytes, with $D_{\text{CU}} = 45$ flits and a TX buffer depth of 64 flits, but this would incur additional logic and an increase in area. Thus, a 32-flit TX buffer allocation for both SL3 and E40G network is retained to maintain equal flow control protocol overhead in this evaluation.

To operate at high data rate, RX buffer depth relies on the link latency, in which SL3's RX buffer depth is at a minimum of 512 flits (see Chapter 2). For the switched network, this is not sufficient due to the additional latency of two or more switch hops; thus, the need to increase E40G's RX buffer allocation to a relatively larger size. For E40G network, FC RX buffer depth is set to 2048 flits, while maintaining 512 flits for SL3.

Figure 4.6 shows the resource utilization of adaptive logic modules (ALMs), registers,

memory logic array blocks (MLAB Kbits), M20K memory blocks, and digital signal processors (DSPs). As shown in green, point-to-point's network modules consume lesser area, while the switched network's consume about 6x, 7x, 18x, and 3x more ALMs, registers, Kbits, and M20Ks, respectively, than the former. This is due to increased logic and memory needed for the frame encoder, decoder, and FC RX buffer allocation. For the E40G switched network, this is around 70-75% of resources, which is a fair amount considering that large application is targeted to be mapped across multiple FPGAs. In addition, it is noteworthy that the SL3 direct network does not include an on-chip router, which when implemented, would imply an increase on its consumption.

4.3.2 Communication Time and Effective Network Bandwidth

To measure parameters for the performance model in Equation (4.6), hardware cycle counters are setup and used for the following cases: (1) point-to-point with SL3, (2) point-to-point with E40G, and (3) a switched network with E40G, as shown in Figure 4.7a. Aside from a switched E40G case (3), a point-to-point connection with E40G case (2) is also considered to obtain the average switching latency, t_{S}.

Table 4.2 shows the measured values for node latency, t_{N} and physical link latency, t_{PL} for a zero-payload equivalent, which in E40G, is encapsulated in a minimum-sized Ethernet frame with 46-byte padded payload. For SL3 case (1), t_{N} only includes FC latency, while for E40G cases (2) and (3), this includes FC, frame encoder, and frame decoder delays; hence, the higher latency of E40G. Due to IP restrictions on SL3 and E40G IP cores, t_{PL} could only be measured by including their protocol overheads; thus, the noticeable difference of their values. Using the measured values of t_{N} and t_{PL}, the RX buffer allocation is also verified to maintain a high data rate transmission.

In order to obtain the effective bandwidth, the total communication time is measured by sending various payload sizes and used it in Equation (4.7). Case (1) shows the highest bandwidth for smaller payload sizes due to its lower communication latency, as illustrated in Figure 4.8. Meanwhile, case (2) shows a lower effective bandwidth than case (1). This

Table 4.2: Measured latency parameters

Network	**Unit**	$t_{\mathbf{N}}$	$t_{\mathbf{PL}}$	$t_{\mathbf{S}}$
(1) Point-to-point with SL3	[us]	0.245	0.354	N/A
(2) Point-to-point with E40G	[us]	0.336	0.496	N/A
(3) Switched with E40G	[us]	0.336	0.496	0.318

is due to the additional protocol overhead of Ethernet and the extra latency of passing through more modules, i.e. frame encoder and decoder, as with case (3). However, the latter shows the lowest effective bandwidth due to a longer communication time via the switch.

For larger payload sizes, it is observed that the effective bandwidth for case (1) is 4.29 GB/s with 86% efficiency. For (2) and (3), both reached a effective bandwidth of 4.41 GB/s at 88% efficiency, which is surprisingly higher than SL3's (approximately 3%). This is caused by SL3 protocol's transmission overheads and lane rate calculations [90], where the required network clock frequency derived was 150.813962 MHz, resulting to 4.83 GB/s peak throughput. For E40G IP core, there is no clock frequency requirement and 154.99442 MHz clock frequency has been utilized, which correspondingly results to a higher peak throughput of 4.96 GB/s. These results demonstrate that even with the switched network's additional overhead, which includes Ethernet protocol and a higher communication latency, it has achieved an equivalent performance to a point-to-point network with sufficiently large payload sizes, since latency no longer dominates transfer time.

Correspondingly, the measured total communication time is also used to validate the performance model by comparing it with our estimated results. By using the obtained parameters such as t_{N} and the effective bandwidth, the transmission time was estimated, as shown in Figure 4.7b-d. Based on the plotted values, the model closely matches the measured time, which can be used to estimate communication performance in larger FPGA clusters.

4.3.3 Performance Estimation of Stream Computing

A stream computing case is considered since it is a promising approach to achieve high throughput data streams from its deep pipelines. A direct network is often the typical choice, thus, its performance in the proposed switching framework is investigated. Two FPGAs in a ring connection were used to perform fundamental evaluation on a switched network, as shown in Figure 4.9a and compared with its equivalent point-to-point ring connection with SL3. The total communication time is obtained and its effective bandwidth is mapped in Figure 4.9b. As anticipated, latency prevails in smaller payload sizes, in which the point-to-point connection has higher effective bandwidth. For larger payload sizes, however, the effective bandwidth of the switched E40G connection saturates at 4.41 GB/s, which still performed better than its direct network counterpart at 4.29 GB/s. This means that an indirect network can achieve equivalent throughput to a direct network when streaming large data sets, which is typical for stream computing applications. Even with the additional communication latency introduced by an indirect network, this becomes negligible when data stream size becomes sufficiently large for its network datapath.

Using Equation (4.2) for SL3 and Equation (4.3) for E40G, the communication time is also estimated by scaling the propagation latency, T_{L} by a factor of two, since this ring connection is equivalent to two point-to-point connections. As shown in Figure 4.9b, the modeled values approximates the measured points, which is expected since the model only accounts for the network communication without interaction.

To evaluate scalability, the communication time of both network connections with a larger cluster setup is estimated. A radix-64 switch ($k = 64$) is assumed, which could accommodate up to $n = 64$ FPGAs. When $n > 64$, the leaf-spine architecture is used to expand the network diameter, where the uplink to downlink ratio is assumed to be balanced (no oversubscription). To build a two-layer, full-bisection bandwidth leaf-spine topology, a total of $n = k \times \frac{k}{2} = 2048$ FPGAs can be connected, with $a = \frac{2n}{k} = 64$ leafs, and $b = \frac{a}{2} = 32$ spines, which are connected in a full bipartite graph with $\frac{k}{a} = 1$ uplink

per leaf to all 32 spines.

In this ring connection, the lowest latency traversal is assumed, where data stream from an FPGA hops to their neighboring FPGA first via intra-leaf hops (see Figure 4.5c), before performing an inter-leaf hop through the spine (see Figure 4.5d). With $n <= 64$, T_{L} is scaled by n, since there are n FPGA-to-FPGA transfers in the ring through a single leaf ($a = 1$). With $n > 64$, the scaling factor for T_{L} is $a(\frac{k}{2} - 1)$, since the FPGAs on the edges of the leaf have to perform an inter-leaf transfer. Consequently, an inter-leaf communication's scaling factor for T_{L} is a, when $a > 1$. By hypothetically assuming the measured parameters in Table 4.2 and the measured effective bandwidth, the total time is estimated, $T = T_{\mathrm{L}} + \frac{m}{B}$, by accumulating the scaled T_{L} values for both intra-leaf and inter-leaf hops, which forms the communication pattern of the ring, while increasing the FPGA cluster size.

Figures 4.9c-e show the transmission time for a large data stream (227 MB), a mid-sized data stream (1 MB), and a small message size (4 KB), respectively. For the large data stream size, a lower transmission time is observed for the E40G switched network up to $n = 1024$ FPGAs, due to its higher effective bandwidth. With $n = 2048$, the data stream size is no longer sufficient with the increased network datapath and the latency factor catches up, making the point-to-point connection with SL3 perform better (see Figure 4.9c). For the mid-sized data stream, as shown in Figure 4.9d, the higher effective bandwidth of the switched network keeps the time difference at a minimum only for a small FPGA cluster (up to $n = 16$ FPGAs). Meanwhile, for small message sizes, as illustrated in Figure 4.9e, the lower latency of a point-to-point connection dominates the total transfer time. This highlights the overhead-inducing component of an indirect network's higher communication latency.

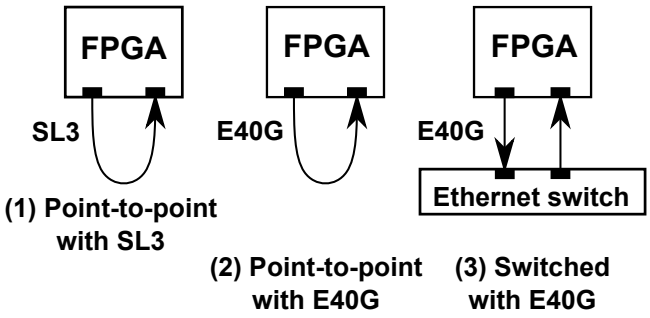
To demonstrate performance scalability, Figures 4.10a-c illustrate the corresponding estimated performance of the ring connection with the same data stream size classifications: large data stream (227 MB), a mid-sized data stream (1 MB), and a small message size (4 KB), respectively. Here, overlapped communication and computation is assumed. Using the stream computing performance model in Equation (3.8) and assuming the

number of operations, O_{total} of tsunami simulation (see Equation (3.2)), performance estimation results are obtained. As expected, streaming a sufficiently large data set scales the performance linearly with more FPGAs on both network types, since they achieve equivalent network throughput. This can be generalized for stream computing applications, since they typically process large data stream sizes, such as an actual ocean depth data set (7,430,699 data elements = 227 MB) used in tsunami simulation (see Figure 4.10a). On the other hand, Figures 4.10b and c show the performance estimation of mid-sized data stream and small message size, respectively. In both cases, higher latency of the indirect network largely affects performance due to both pipeline and communication overheads, when data stream size is insufficient in the ring network datapath.

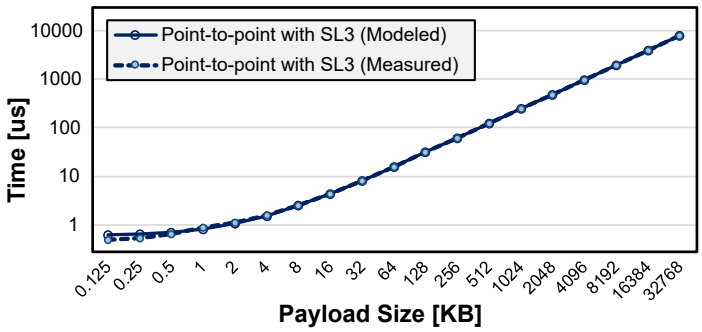
4.4 Conclusions

This chapter presents a design and architecture of an Ethernet-based switched platform for scalable FPGA clusters, where a connection-oriented datapath with backpressure over the network is established. For usability and to setup connectivity, we utilized the credit-based protocol with flow control over Ethernet and implemented the supporting network modules to achieve high-throughput data transfers.

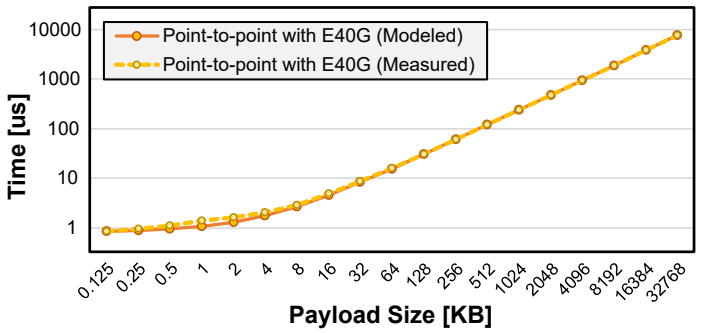
The performance characteristics of the connection-oriented links are investigated and its communication performance is modeled. By obtaining the communication time and effective network bandwidth, the communication time of a streamed computing pattern when scaled to a large-sized cluster is estimated. With the E40G switched network saturating at a higher effective bandwidth for large data streams in comparison with its point-to-point SL3 counterpart, the proposed indirect framework has demonstrated good communication performance and scalability for applications requiring high-bandwidth and large data transfers, despite its longer network propagation latency.



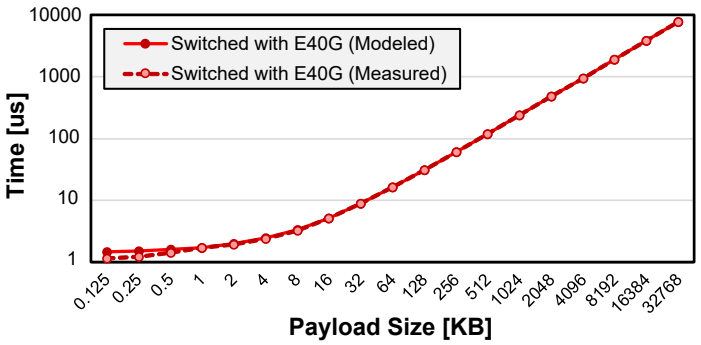
a) Experimental setup for cases (1), (2), and (3)



b) Transmission time for case (1)



c) Transmission time for case (2)



d) Transmission time for case (3)

Figure 4.7: Modeled vs. measured network communication time

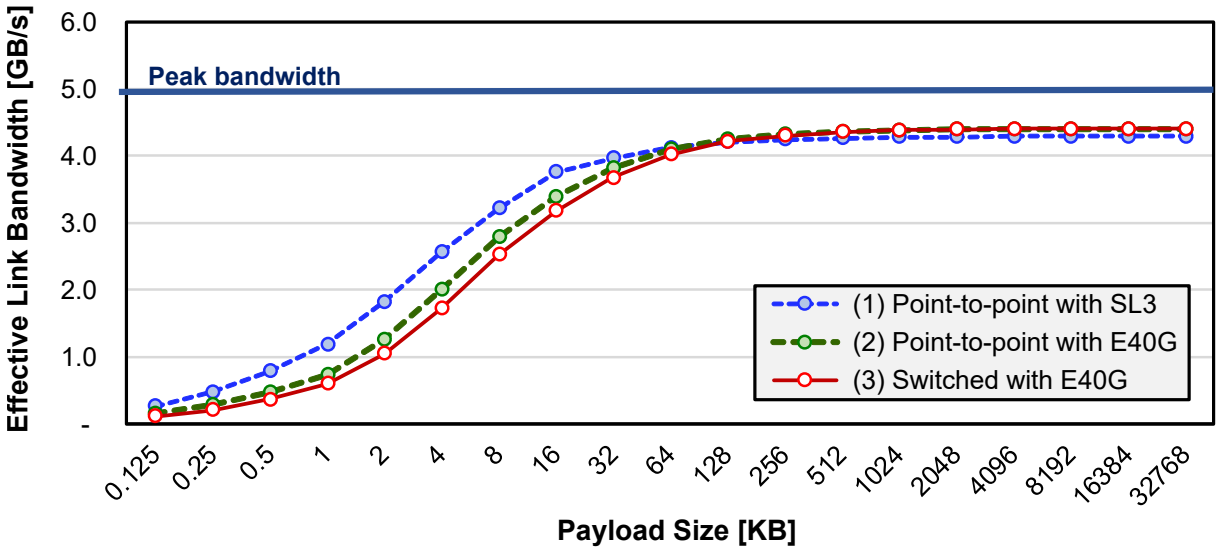
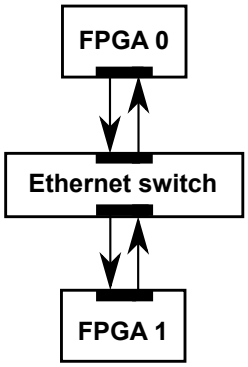
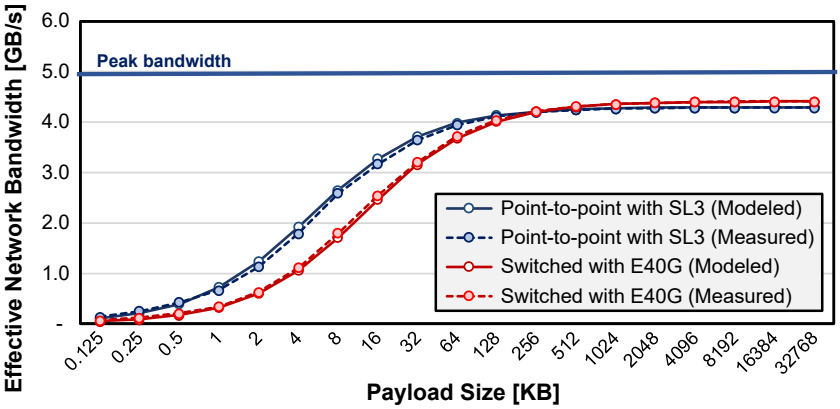


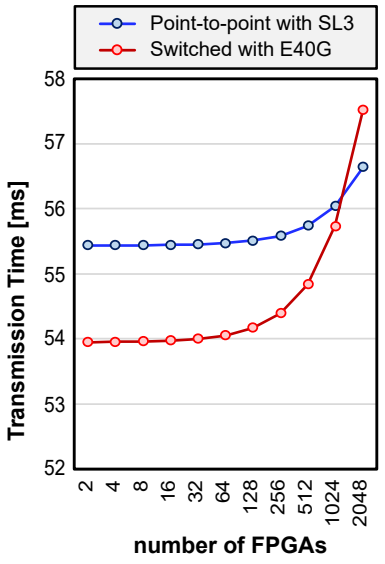
Figure 4.8: Effective network bandwidth of SL3 and E40G for cases (1), (2), and (3)



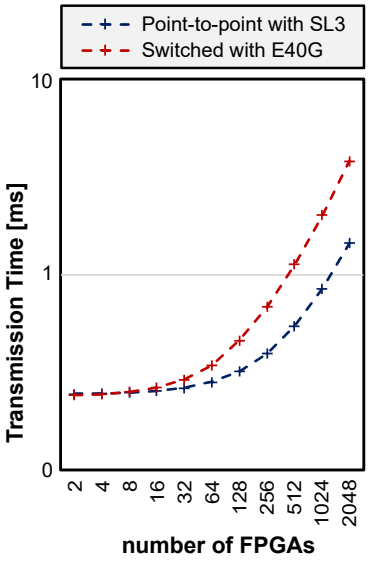
a) Switched network



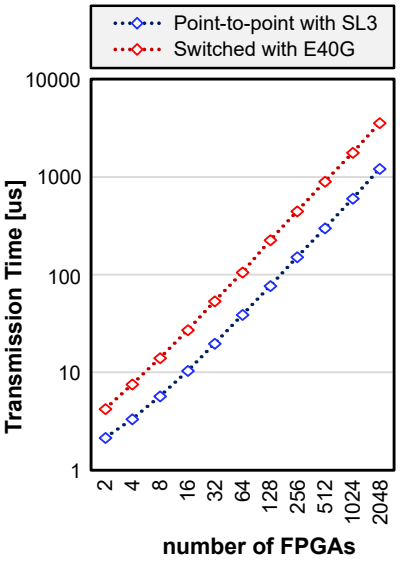
b) Effective network bandwidth with two FPGAs



c) Estimated time for large data stream (227 MB)

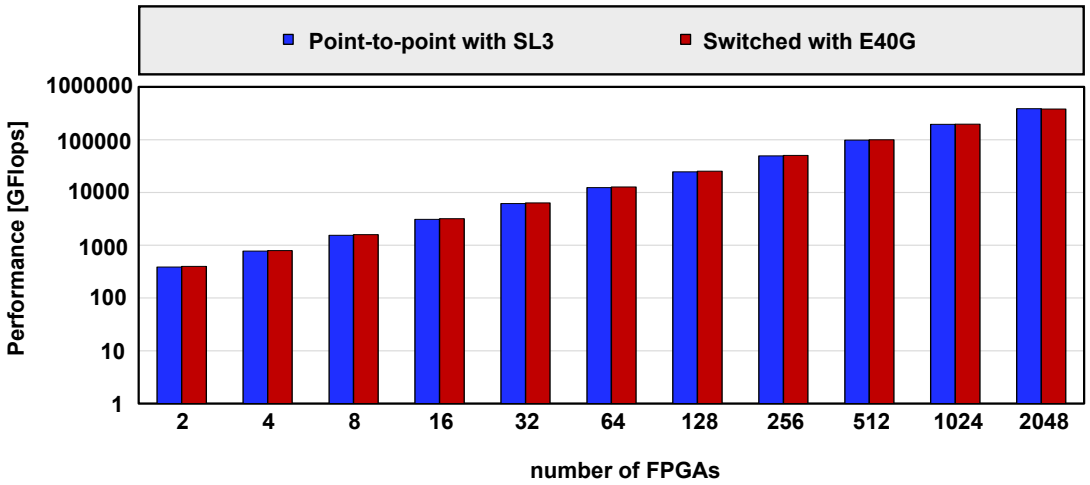


d) Estimated time for mid-sized data stream (1 MB)

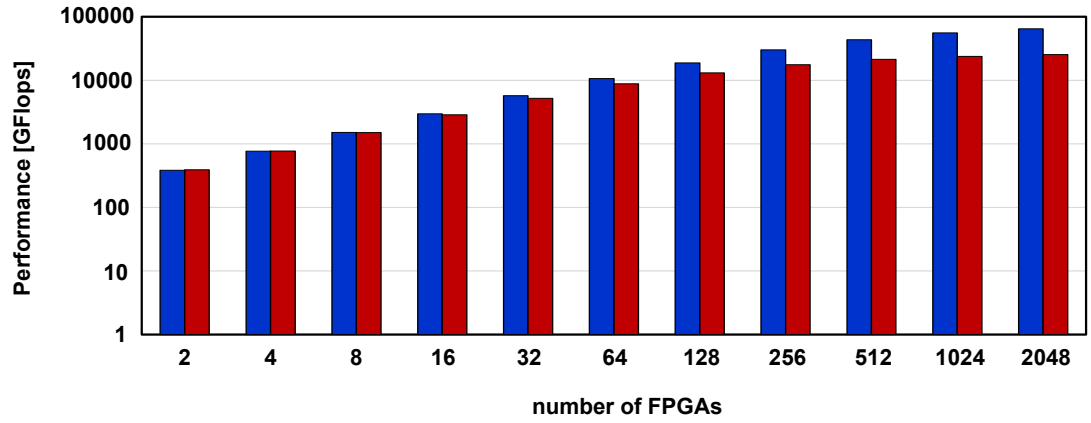


e) Estimated time for small message size (4 KB)

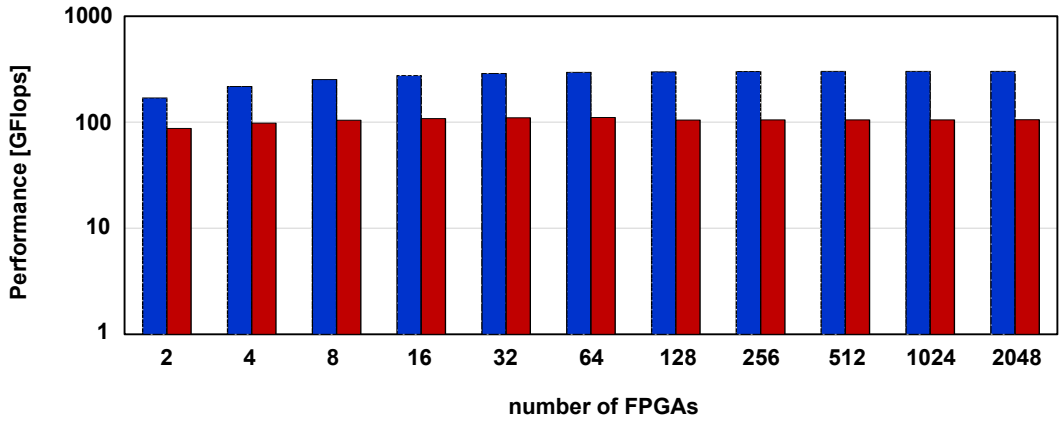
Figure 4.9: Stream computing in a ring connection



a) Estimated performance for large data stream (227 MB*)
Size of tsunami simulation's ocean depth data set



b) Estimated performance for mid-sized data stream (1 MB)



c) Estimated performance for small message size (4 KB)

Figure 4.10: Performance estimation of stream computing in a ring connection

Chapter 5

Conclusions

HPC is among the affected areas with the imminent end of Moore's law and Dennard scaling. Among the several strategies and architectural explorations being done over the years, reconfigurable devices such as FPGAs are dominantly showing their potential through exploiting parallelism with customization and the use of appropriate hardware computing model, such as data flow stream computing. Despite numerous successful studies, FPGAs do not have significant impact on general-purpose HPC systems due to a huge gap between the potential and reality of FPGAs in HPC [27]. In response to a few areas identified in [1] to hasten FPGA adoption in HPC systems, it is the general direction and long-term goal of this research to make FPGAs available to HPC systems as offload engines in a system-wide custom computing infrastructure. Since performance scalability is limited by the available on-chip resources of a single FPGA, it is natural to explore the suitability of tightly-coupled FPGA clusters.

Since FPGAs are relatively new in HPC and are highly customizable in nature, the overall design space is huge, which extends to its interconnection networks. This dissertation particularly focuses on the comparison of direct and indirect network types for FPGA clusters, with specific intent for stream computing requirements. Since the interconnection network is an overhead-inducing component of an FPGA cluster, there is a need to investigate the performance characteristics of both direct and indirect networks in order achieve low-latency and high-throughput communication necessary for high-performance

stream computing.

The main objective of this dissertation is to explore appropriate interconnection networks for stream computing FPGA clusters, in which the following sub-objectives are derived: (1) investigate the suitability and feasibility of direct and indirect networks for stream computing FPGA clusters; (2) design and implement a lightweight and efficient hardware backpressure mechanism for direct and indirect inter-FPGA communication; and (3) investigate and evaluate performance scalability of stream computing on direct and indirect networks. In this dissertation, the first and third sub-objectives were addressed in Chapters 3-4, while Chapter 2 addressed the second sub-objective.

Chapter 2 investigated the requirements for stream computing in FPGA clusters. For most HPC applications, the following demands should be met: a scalable network architecture; efficient, low-latency, and high-bandwidth communication; and with a small footprint on the FPGA fabric. Since stream computing is a good approach to extract high-performance gains in FPGAs, its requirements were also considered, where inter-FPGA backpressure and synchronization must be available. To meet these requirements, a lightweight and efficient hardware backpressure mechanism was designed and implemented. This was achieved by designing a custom credit-based network protocol with flow control, which supports half-duplex and full-duplex communication for both direct and indirect networks. To keep low-latency and high-bandwidth requirements, design parameters were explored and identified as the communication buffers and the credit update frequency, which implies performance and area trade-offs. Using the parameters with least area consumption, the effective network bandwidth was obtained. While the implemented flow controller design in this chapter was on a direct network, the same design principles and mechanism apply for an indirect network, which was investigated further in Chapter 4.

Chapter 3 investigated the suitability of direct networks through point-to-point connections for FPGA clusters. The design and architecture of a deeply pipelined stream computing platform in a 1D torus or ring topology was presented and implemented. Available parallelism for stream computing was also explored to efficiently utilize the hardware

resources. Using the flow control mechanism in Chapter 2 for its network modules, the performance characteristics of its inter-FPGA links were investigated. Through the derivation of a performance model, a practical and efficient design space exploration could be achieved. Performance evaluation was also performed through the implementation of a practical stream computing application. To mitigate the bottleneck-prone communication between FPGAs, lossless bandwidth-compression hardware [4, 86–88] was utilized and investigated. Even with the insufficient link bandwidth caused by wide computing pipelines, reduced stall ratios were obtained, which resulted to improved efficiency.

Chapter 4 explored the feasibility of a scalable and flexible architecture of indirect networks with Ethernet switches. Since stream computing is one of the target applications, connection-oriented links with backpressure support was designed and implemented for standard Ethernet protocol. Through implementation of the necessary network modules, which included the universal flow controller module introduced in Chapter 2, the performance characteristics of the connection-oriented switched network was investigated, and its corresponding performance model was proposed. Performance evaluation was also done by comparing its performance with that of a point-to-point connection's. By obtaining the communication time and effective network bandwidth, a stream computing pattern was estimated on a large-scale FPGA cluster, where a tree topology of switches were considered to increase the network diameter. In this chapter, it was observed and demonstrated that an indirect network can achieve equivalent throughput to a direct network's when streaming large data sets, which is typical for stream computing applications. Even with the additional communication latency introduced by an indirect network, this becomes negligible when the data stream size becomes sufficiently large for its network datapath.

Through prototype implementations, obtaining performance characteristics by empirical measurements, performance modeling, design space explorations, and performance evaluation by estimations and scalability analyses, these different evaluation methods in this dissertation have demonstrated the suitability and feasibility of both direct and indirect networks for stream computing FPGA clusters. For high-performance stream

computing applications, both direct and indirect networks would be good choices for inter-FPGA communication due to their equivalent network throughput, where latency would be deemed insignificant. Generally, since large data sets are being utilized and processed, streaming these sufficiently large data streams scales the performance linearly with more FPGAs for both network types. On the other hand, performance of insufficient data stream sizes on both network types demonstrates communication latency as an overhead-inducing factor, causing degradation of performance. In this case, the indirect network's total transmission time would be higher than a direct network's, which allows latency to dominate and to negatively affect the overall performance.

With respect to the general direction and long-term goal of this research, an indirect network is found to be a sufficient option for general usage in HPC systems, including stream computing applications, due to its scalability and flexibility features. Moreover, a smaller subset of FPGAs in a large-scale indirect network could be allocated for a target application, while its appropriate datapath could also be customized without changing physical connections. This means that an indirect network for large-scale tightly-coupled FPGA clusters is a good infrastructure for offload engines in an HPC environment, such as in supercomputers. As demonstrated in Chapter 4, a switched Ethernet network performs better with larger data streams, compared to a direct network with SL3 protocol, which generally demonstrates good communication performance. This discovery is particularly useful for engineers and hardware designers in selecting an appropriate network protocol and network type for different requirements.

For future work, other communication patterns should be investigated and evaluated on the switched network to further evaluate its network flexibility. In addition, the performance model for indirect network needs to be fine-tuned since it only focused on communication time, without considering computation or interaction delays. Design space exploration should be done with Stratix 10 FPGAs, where their transceiver links support 100 Gbps data rate. This implies improved effective network bandwidth, which suggests an even better performance for both direct and indirect networks.

Another area of future work for the indirect network exploration is to provide a stan-

dard platform for FPGA cluster management, such as mapping of applications and network configurations into the FPGA cluster. As a general direction, the indirect network provides a scalable and flexible infrastructure for high-level synthesis compilers and virtualization management of a large-scale FPGA cluster.

Bibliography

[1] K. D. Underwood, K. S. Hemmert, and C. D. Ulmer, "From silicon to science," *ACM Transactions on Reconfigurable Technology and Systems*, vol. 2, no. 4, pp. 1–15, Sep. 2009.

[2] TOP500, "Top500 supercomputers." [Online]. Available: https://www.top500.org/lists/top500/

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

[4] T. Ueno, K. Sano, and S. Yamamoto, "Bandwidth compression of floating-point numerical data streams for fpga-based high-performance computing," *ACM Transactions on Reconfigurable Technology and Systems*, vol. 10, no. 3, pp. 1–22, May 2017.

[5] G. E. Moore, "Cramming more components onto integrated circuits," *Proceedings of the IEEE*, vol. 86, no. 1, pp. 82–85, 1998.

[6] R. H. Dennard, F. H. Gaensslen, H. N. Yu, V. L. Rideout, E. Bassous, and A. R. Leblanc, "Design of ion-implanted mosfet's with very small physical dimensions," pp. 256–268, 1974.

[7] H. Esmaeilzadeh, E. Blem, R. St. Amant, K. Sankaralingam, and D. Burger, "Dark silicon and the end of multicore scaling," *IEEE Micro*, vol. 32, no. 3, pp. 122–134, 2012.

[8] K. Jim and H.-C. Hoppe, "The technology stacks of high performance computing and big data computing: What they can learn from each other," 2018. [Online]. Available: www.BDVA.eu

[9] M. C. Herbordt, T. VanCourt, Y. Gu, B. Sukhwani, A. Conti, J. Model, and D. DiSabello, "Achieving high performance with fpga-based computing," *Computer*, vol. 40, no. 3, pp. 50–57, Mar. 2007.

[10] D. Lewis, G. Chiu, J. Chromczak, D. Galloway, B. Gamsa, V. Manohararajah, I. Milton, T. Vanderhoek, and J. Van Dyken, "The stratix^{TM} 10 highly pipelined fpga architecture," in *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays - FPGA '16*. New York, New York, USA: ACM, 2016, pp. 159–168.

[11] M. Vestias and H. Neto, "Trends of cpu, gpu and fpga for high-performance computing," in *Proceedings of the 2014 24th International Conference on Field Programmable Logic and Applications (FPL)*. Munich, Germany: IEEE, Sep. 2014, pp. 1–6.

[12] M. Langhammer and B. Pasca, "Floating-point dsp block architecture for fpgas," in *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays - FPGA '15*. New York, New York, USA: ACM, 2015, pp. 117–125.

[13] M. Parker, "Understanding peak floating-point performance claims," Intel, Tech. Rep., 2016.

[14] Altera, "Achieving one teraflops with 28-nm fpgas," *Altera White Paper*, Sep. 2010.

[15] A. Davidson, "A new fpga architecture and leading-edge finfet process technology promise to meet next-generation system requirements," *Intel FPGA White Paper*, 2015.

[16] E. Nurvitadhi, S. Subhaschandra, G. Boudoukh, G. Venkatesh, J. Sim, D. Marr, R. Huang, J. Ong Gee Hock, Y. T. Liew, K. Srivatsan, and D. Moss, "Can fpgas beat gpus in accelerating next-generation deep neural networks?" in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays - FPGA '17*. New York, New York, USA: ACM Press, 2017, pp. 5–14.

[17] A. DeHon, J. Adams, M. DeLorimier, N. Kapre, Y. Matsuda, H. Naeimi, M. Vanier, and M. Wrighton, "Design patterns for reconfigurable computing," in *Proceedings of the 12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'04)*, 2004.

[18] M. C. Herbordt, Y. Gu, T. Vancourt, J. Model, B. Sukhwani, and M. Chiu, "Computing models for fpga-based accelerators." *IEEE Computing in Science and Engineering*, vol. 10, no. 6, pp. 35–45, Oct. 2008.

[19] L. Gan, W. Luk, W. Xue, X. Huang, Y. Zhang, G. Yang, H. Fu, X. Huang, G. Yang, H. Fu, and C. Yang, "Solving the global atmospheric equations through heterogeneous reconfigurable platforms," *ACM Transactions on Reconfigurable Technology and Systems*, vol. 8, no. 2, 2015.

[20] O. Lindtjorn, R. G. Clapp, O. Pell, O. Mencer, M. J. Flynn, and H. Fu, "Beyond traditional microprocessors for geoscience high-performance computing applications," *IEEE Micro*, vol. 31, no. 2, pp. 41–49, Mar. 2011.

[21] M. Chiu and M. C. Herbordt, "Molecular dynamics simulations on high performance recon-figurable computing systems," *ACM Transactions on Reconfigurable Technology and Systems*, vol. 3, no. 4, 2010.

[22] A. Mahram and M. C. Herbordt, "Ncbi blastp on high-performance reconfigurable computing systems," *ACM Trans. Reconfig. Technol. Syst. 7, 4, Article*, vol. 7, no. 4, 2015.

[23] A. Ebrahimi and M. Zandsalimy, "Evaluation of fpga hardware as a new approach for accelerating the numerical solution of cfd problems," *IEEE Access*, vol. 5, pp. 9717–9727, 2017.

[24] M. Awad, "Fpga supercomputing platforms: A survey," *FPL 09: 19th International Conference on Field Programmable Logic and Applications*, pp. 564–568, Aug. 2009.

[25] J. Fowers, K. Ovtcharov, M. Papamichael, T. Massengill, M. Liu, D. Lo, S. Alkalay, M. Haselman, L. Adams, M. Ghandi, S. Heil, P. Patel, A. Sapek, G. Weisz, L. Woods, S. Lanka, S. K. Reinhardt, A. M. Caulfield, E. S. Chung, and D. Burger, "A configurable cloud-scale dnn processor for real-time ai," in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, Jun. 2018, pp. 1–14.

[26] Q. Xiong, A. Skjellum, and M. C. Herbordt, "Accelerating mpi message matching through fpga offload," in *2018 28th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, Aug. 2018, pp. 191–1914.

[27] C. Plessl, "Keynote 2 - fpga-accelerated high-performance computing – close to breakthrough or pipedream?" in *2017 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*. IEEE, 2017.

[28] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, "Optimizing fpga-based accelerator design for deep convolutional neural networks," in *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays - FPGA '15*. New York, New York, USA: ACM Press, 2015, pp. 161–170.

[29] R. DiCecco, G. Lacey, J. Vasiljevic, P. Chow, G. Taylor, and S. Areibi, "Caffeinated fpgas: Fpga framework for convolutional neural networks," in *2016 International Conference on Field-Programmable Technology (FPT)*. Xi'an, China: IEEE, 2016.

[30] E. Wang, J. J. Davis, R. Zhao, H.-C. H.-c. Ng, X. Niu, W. Luk, P. Y. K. Cheung, G. A. Constantinides, P. Y. K Cheung, G. A. Constantinides, H.-C. H.-c. Ng, X. Niu,

E. Wang, J. J. Davis, P. Y. K Cheung, G. A. Constantinides, R. Zhao, H.-C. H.-c. Ng, and W. Luk, "Deep neural network approximation for custom hardware: Where we've been, where we're going," *ACM Comput. Surv. 1, 1, Article*, vol. 1, no. 1, Jan. 2019.

[31] S. Neuendorffer and K. Vissers, "Streaming systems in fpgas," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 5114 LNCS, 2008, pp. 147–156.

[32] R. Sierra, F. Mangani, C. Carreras, and G. Caffarena, "High-performance decoding of variable-length memory data packets for fpga stream processing," in *29th International Conference on Field Programmable Logic and Applications (FPL)*, 2019, 2019, pp. 307–313.

[33] M. Koraei, O. Fatemi, and M. Jahre, "Dcmi: A scalable strategy for accelerating iterative stencil loops on fpgas," *ACM Transactions on Architecture and Code Optimization*, vol. 16, no. 4, Oct. 2019.

[34] R. Stephens, "A survey of stream processing," *Acta Informatica*, vol. 34, no. 7, pp. 491–541, 1997.

[35] I. Buck, T. Foley, D. Horn, J. Sugerman, K. Fatahalian, M. Houston, and P. Hanrahan, "Brook for gpus: Stream computing on graphics hardware," in *SIGGRAPH04: Special Interest Group on Computer Graphics and Interactive Techniques*, 2004.

[36] F. Plavec, "Stream computing on fpgas," Ph.D. dissertation, University of Toronto, 2010.

[37] M. Lin, S. Cheng, and J. Wawrzynek, "Cascading deep pipelines to achieve high throughput in numerical reduction operations," in *2010 International Conference on Reconfigurable Computing Cascading.* Quintana Roo, Mexico: IEEE, 2010.

[38] K. Dohi, K. Okina, R. Soejima, Y. Shibata, and K. Oguri, "Performance modeling of stencil computing on a stream-based fpga accelerator for efficient design space ex-

ploration," *PAPER Special Section on Reconfigurable Systems, IEICE Transactions*, vol. E98-D, no. 2, 2015.

[39] K. Sano, S. Abiko, and T. Ueno, "Fpga-based stream computing for high-performance n-body simulation using floating-point dsp blocks," *Proceedings of the 8th International Symposium on Highly Efficient Accelerators and Reconfigurable Technologies - HEART2017*, no. June, pp. 1–6, 2017.

[40] K. Nagasu, K. Sano, F. Kono, and N. Nakasato, "Fpga-based tsunami simulation: Performance comparison with gpus, and roofline model for scalability analysis," *Journal of Parallel and Distributed Computing*, vol. 106, no. August, pp. 153–169, 2016.

[41] K. Sano and S. Yamamoto, "Fpga-based scalable and power-efficient fluid simulation using floating-point dsp blocks," *IEEE Transactions on Parallel and Distributed Systems*, vol. PP, no. 99, 2017.

[42] H. R. Zohouri, A. Podobas, and S. Matsuoka, "Combined spatial and temporal blocking for high-performance stencil computation on fpgas using opencl," in *Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays - FPGA '18*. New York, New York, USA: ACM Press, 2018, pp. 153–162.

[43] A. Putnam, A. M. Caulfield, E. S. Chung, D. Chiou, K. Constantinides, J. Demme, H. Esmaeilzadeh, J. Fowers, G. P. Gopal, J. Gray, M. Haselman, S. Hauck, S. Heil, A. Hormati, J.-Y. Kim, S. Lanka, J. Larus, E. Peterson, S. Pope, A. Smith, J. Thong, P. Y. Xiao, D. Burger, A. M. Caulfield, A. Smith, J. Thong, P. Y. Xiao, D. Burger, E. S. Chung, D. Chiou, K. Constantinides, J. Demme, H. Esmaeilzadeh, J. Fowers, G. P. Gopal, J. F. Gopi, P. Gopal, J. Gray, M. Haselman, S. Hauck, S. Heil, A. Hormati, J.-Y. Kim, S. Lanka, J. Larus, E. Peterson, S. Pope, A. Smith, J. Thong, P. Yi, X. Doug, and B. Microsoft, "A reconfigurable fabric for accelerating large-scale datacenter services," in *ISCA '14 Proceeding of the 41st annual international*

symposium on Computer architecuture. Minneapolis, MN, USA: IEEE, 2014, pp. 13–24.

[44] A. M. Caulfield, E. S. Chung, A. Putnam, H. Angepat, J. Fowers, M. Haselman, S. Heil, M. Humphrey, P. Kaur, J.-Y. Kim, D. Lo, T. Massengill, K. Ovtcharov, M. Papamichael, L. Woods, S. Lanka, D. Chiou, and D. Burger, "A cloud-scale acceleration architecture," in *MICRO-49 The 49th Annual IEEE/ACM International Symposium on Microarchitecture*, 2016.

[45] AWS, "Amazon ec2 f1 instances." [Online]. Available: https://aws.amazon.com/ec2/instance-types/f1/

[46] F. Chen, Y. Shan, Y. Zhang, Y. Wang, H. Franke, X. Chang, and K. Wang, "Enabling fpgas in the cloud," in *CF '14: Proceedings of the 11th ACM Conference on Computing Frontiers*. Association for Computing Machinery, 2014.

[47] S. Byma, J. G. Steffan, H. Bannazadeh, A. Leon-Garcia, and P. Chow, "Fpgas in the cloud: Booting virtualized hardware accelerators with openstack," in *Proceedings - 2014 IEEE 22nd International Symposium on Field-Programmable Custom Computing Machines, FCCM 2014*. Institute of Electrical and Electronics Engineers Inc., Jul. 2014, pp. 109–116.

[48] S. A. Fahmy, K. Vipin, and S. Shreejith, "Virtualized fpga accelerators for efficient cloud computing," in *Proceedings - IEEE 7th International Conference on Cloud Computing Technology and Science, CloudCom 2015*. Institute of Electrical and Electronics Engineers Inc., 2015, pp. 430–435.

[49] J. Weerasinghe, F. Abel, C. Hagleitner, and A. Herkersdorf, "Enabling fpgas in hyperscale data centers," in *Proceedings of the 2015 IEEE 12th Intl Conf on Ubiquitous Intelligence and Computing and 2015 IEEE 12th Intl Conf on Autonomic and Trusted Computing and 2015 IEEE 15th Intl Conf on Scalable Computing and Communications and Its Associated Workshops (UIC-AT*. IEEE, Aug. 2015, pp. 1078–1086.

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

[51] J. Sheng, C. Yang, A. Sanaullah, M. Papamichael, A. Caulfield, and M. C. Herbordt, "Hpc on fpga clouds: 3d ffts and implications for molecular dynamics," in *2017 27th International Conference on Field Programmable Logic and Applications, FPL 2017*, 2017, pp. 5–8.

[52] J. Sheng, C. Yang, and M. C. Herbordt, "High performance communication on reconfigurable clusters," in *2018 28th International Conference on Field Programmable Logic and Applications (FPL)*, Dublin, Ireland, 2018.

[53] N. Tarafdar, T. Lin, E. Fukuda, H. Bannazadeh, A. Leon-Garcia, and P. Chow, "Enabling flexible network fpga clusters in a heterogeneous cloud data center," New York, New York, USA, pp. 237–246, 2017.

[54] A. T. Markettos, P. J. Fox, S. W. Moore, A. W. Moore, A. Theodore Markettos, P. J. Fox, S. W. Moore, and A. W. Moore, "Interconnect for commodity fpga clusters: Standardized or customized?" *Conference Digest - 24th International Conference on Field Programmable Logic and Applications, FPL 2014*, pp. 1–8, Sep. 2014.

[55] R. S. Correa, D. De, and G. Électrique, "Implementation of ultra-low latency and high-speed communication channels for an fpga-based hpc cluster," Ph.D. dissertation, Université de Montréal, 2017.

[56] L. M. Ni, "Issues in designing truly scalable interconnection networks," in *Proceedings of the International Conference on Parallel Processing Workshops*. Institute of Electrical and Electronics Engineers Inc., 1996, pp. 74–83.

[57] O. Mencer, K. H. Tsoi, S. Craimer, T. Todman, W. Luk, M. Y. Wong, and P. H. W. Leong, "Cube: A 512-fpga cluster," in *Proceedings of the 2009 5th Southern Conference on Programmable Logic (SPL)*. IEEE, Apr. 2009, pp. 51–57.

[58] C. Chang, J. Wawrzynek, and R. W. Brodersen, "Bee2: A high-end reconfigurable computing system," *IEEE Design and Test of Computers*, vol. 22, no. 2, pp. 114–125, Apr. 2005.

[59] M. Porrmann, J. Hagemeyer, J. Romoth, M. Strugholtz, and C. Pohl, "Raptor-a scalable platform for rapid prototyping and fpga-based cluster computing," *Advances in Parallel Computing*, vol. 19, pp. 592–599, 2010.

[60] S. W. Moore, P. J. Fox, S. J. Marsh, A. T. Markettos, and A. Mujumdar, "Bluehive - a field-programable custom computing machine for extreme-scale real-time neural network simulation," in *2012 IEEE 20th International Symposium on Field-Programmable Custom Computing Machines*. IEEE, Apr. 2012, pp. 133–140.

[61] R. Baxter, S. Booth, M. Bull, G. Cawood, J. Perry, M. Parsons, A. Simpson, A. Trew, A. McCormick, G. Smart, R. Smart, A. Cantle, R. Chamberlain, and G. Genest, "Maxwell - a 64 fpga supercomputer," in *Second NASA/ESA Conference on Adaptive Hardware and Systems (AHS 2007)*. IEEE, Aug. 2007, pp. 287–294.

[62] T. Bunker and S. Swanson, "Latency-optimized networks for clustering fpgas," pp. 129–136, Apr. 2013.

[63] M. Nüssle, B. Geib, H. Fröning, and U. Brüning, "An fpga-based custom high performance interconnection network," in *ReConFig'09 - 2009 International Conference on ReConFigurable Computing and FPGAs*, 2009, pp. 113–118.

[64] H. Fröning, M. Nüssle, H. Litz, and U. Brüning, "A case for fpga based accelerated communication," in *9th International Conference on Networks, ICN 2010*, 2010, pp. 28–33.

[65] R. Ammendola, A. Biagioni, O. Frezza, F. Lo Cicero, A. Lonardo, P. Paolucci, D. Rossetti, A. Salamon, F. Simula, L. Tosoratto, and P. Vicini, "A 34 gbps data transmission system with fpgas embedded transceivers and qsfp+ modules," in *IEEE Nuclear Science Symposium Conference Record*, 2012, pp. 872–876.

[66] N. Zilberman, Y. Audzevich, G. A. Covington, and A. W. Moore, "Netfpga sume: Toward 100 gbps as research commodity," *IEEE Micro*, vol. 34, no. 5, pp. 32–41, Sep. 2014.

[67] I. Kuon, R. Tessier, and J. Rose, "Fpga architecture: Survey and challenges," *Electronic Design Automation*, vol. 2, no. 2, pp. 135–253, 2008.

[68] A. Azarian and J. M. P. Cardoso, "Coarse/fine-grained approaches for pipelining computing stages in fpga-based multicore architectures," in *Proceedings of the European Conference on Parallel Processing: Euro-Par 2014: Parallel Processing Workshops*. Springer, 2014, pp. 266–278.

[69] H. Ziegler, Byoungro So, M. Hall, and P. Diniz, "Coarse-grain pipelining on multiple fpga architectures," in *Proceedings of the 10th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*. Napa, CA, USA, USA: IEEE, 2002, pp. 77–86.

[70] S. Murtaza, A. G. Hoekstra, and P. M. A. Sloot, "Cellular automata simulations on a fpga cluster," *The International Journal of High Performance Computing Applications*, vol. 25, no. 2, pp. 193–204, May 2011.

[71] P. A. Skordos, "Initial and boundary conditions for the lattice boltzmann method," *Physical Review E*, vol. 48, no. 6, pp. 4823–4842, Dec. 1993.

[72] Y. Kono, K. Sano, and S. Yamamoto, "Scalability analysis of tightly-coupled fpga-cluster for lattice boltzman computation," in *Proceedings of the 22nd International Conference on Field Programmable Logic and Applications (FPL 2012)*. IEEE, Aug. 2012, pp. 120–127.

[73] K. Sano, Y. Kono, H. Suzuki, R. Chiba, R. Ito, T. Ueno, K. Koizumi, and S. Yamamoto, "Efficient custom computing of fully-streamed lattice boltzmann method on tightly-coupled fpga cluster," *ACM SIGARCH Computer Architecture News*, vol. 41, no. 5, pp. 47–52, Dec. 2013.

[74] S.-W. Jun, M. Liu, S. Xu, and Arvind, "A transport-layer network for distributed fpga platforms," in *2015 25th International Conference on Field Programmable Logic and Applications (FPL)*. London, UK: IEEE, Sep. 2015, pp. 1–4.

[75] H. Kung and R. Morris, "Credit-based flow control for atm networks," *IEEE Network*, vol. 9, no. 2, pp. 40–48, 1995.

[76] R. Sass, W. V. Kritikos, A. G. Schmidt, S. Beeravolu, P. Beeraka, K. Datta, D. Andrews, R. S. Miller, and D. Stanzione, "Reconfigurable computing cluster (rcc) project: Investigating the feasibility of fpga-based petascale computing," in *Proceedings 2007 IEEE Symposium on Field-Programme Custom Computing Machines, FCCM 2007*. IEEE Computer Society, 2007, pp. 127–138.

[77] M. Nüssle, H. Fröning, S. Kapferer, and U. Brüning, "Accelerate communication, not computation!" in *High-Performance Computing Using FPGAs*. New York, NY: Springer New York, 2013, pp. 507–542.

[78] H. Kung and Koling Chang, "Receiver-oriented adaptive buffer allocation in credit-based flow control for atm networks," in *Proceedings of INFOCOM'95*. IEEE Comput. Soc. Press, 1995, pp. 239–252.

[79] R. Jain, "Congestion control and traffic management in atm networks: Recent advances and a survey," *Computer Networks and ISDN Systems*, vol. 28, no. 13, pp. 1723–1738, Oct. 1996.

[80] S. Kamolphiwong, A. Karbowiak, and H. Mehrpour, "Flow control in atm networks: a survey," *Elsevier Computer Communications*, vol. 21, pp. 951–968, 1998.

[81] BERTEN DSP, "Gpu vs fpga performance comparison," *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays - FPGA '17*, pp. 2–5, May 2016.

[82] "Terasic inc. web." [Online]. Available: http://www.terasic.com.tw/en/

[83] K. Sano, Y. Hatsuda, and S. Yamamoto, "Scalable streaming-array of simple soft-processors for stencil computations with constant memory-bandwidth," in *Proceedings on the 2011 IEEE 19th Annual International Symposium on Field-Programmable Custom Computing Machines*. IEEE, May 2011, pp. 234–241.

[84] H. M. Waidyasooriya, M. Hariyama, H. Muthumala, W. And, M. Hariyama, H. M. Waidyasooriya, and M. Hariyama, "Multi-fpga accelerator architecture for stencil computation exploiting spacial and temporal scalability," *IEEE Access*, vol. 7, pp. 53 188–53 201, 2019.

[85] K. Sano, Y. Hatsuda, and S. Yamamoto, "Multi-fpga accelerator for scalable stencil computation with constant memory bandwidth," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 3, pp. 695–705, Feb. 2014.

[86] T. Ueno, Y. Kono, K. Sano, and S. Yamamoto, "Parameterized design and evaluation of bandwidth compressor for floating-point data streams in fpga-based custom computing," in *Reconfigurable Computing: Architectures, Tools and Applications. ARC 2013. Lecture Notes in Computer Science*, P. Brisk, J. de Figueiredo Coutinho, and P. Diniz, Eds., vol. 7806 LNCS. Los Angeles, CA, USA: Springer-Verlag Berlin Heidelberg, 2013, pp. 90–102.

[87] K. Sano, K. Katahira, and S. Yamamoto, "Segment-parallel predictor for fpga-based hardware compressor and decompressor of floating-point data streams to enhance memory i/o bandwidth," *Data Compression Conference Proceedings*, pp. 416–425, 2010.

[88] T. Ueno, Y. Kono, K. Sano, and S. Yamamoto, "Fpga-based implementation of compact compressor and decompressor of floating-point data-stream for bandwidth reduction," in *Proceedings of the International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA '12)*, 2012, pp. 119–126.

[89] P. Lindstrom and M. Isenburg, "Fast and efficient compression of floating-point data," *IEEE Transactions on Visualization and Computer Graphics*, vol. 12, no. 5, pp. 1245–1250, Nov. 2006.

[90] Intel, "Seriallite iii ip solution." [Online]. Available: https://www.altera.com/solutions/technology/transceiver/protocols/pro-seriallite-3.html

[91] K. Sano, "Dsl-based design space exploration for temporal and spatial parallelism of custom stream computing," in *Proceedings of the Second International Workshop on FPGAs for Software Programmers (FSP 2015)*, Aug. 2015, pp. 29–34.

[92] "Flopoco project web." [Online]. Available: http://flopoco.gforge.inria.fr/

[93] Ethernet Alliance, "2019 roadmap - ethernet alliance," 2019. [Online]. Available: https://ethernetalliance.org/technology/2019-roadmap/

[94] M. Alizadeh and T. Edsall, "On the data path performance of leaf-spine datacenter fabrics," in *Proceedings - IEEE 21st Annual Symposium on High-Performance Interconnects, HOTI 2013*. IEEE Computer Society, 2013, pp. 71–74.

[95] S. Walklin, "Leaf-spine architecture for otn switching," in *2017 International Conference on Computing, Networking and Communications, ICNC 2017*. Institute of Electrical and Electronics Engineers Inc., Mar. 2017, pp. 95–99.

[96] Cisco Systems, "Cisco data center spine-and-leaf architecture: Design overview (white paper)," Cisco Systems, Inc., Tech. Rep., 2016.

[97] IEEE, "Ieee 802.3ba-2010 standard for information technology." [Online]. Available: https://standards.ieee.org/standard/802{_}3ba-2010.html

[98] Intel, "Low latency 40-gbps ethernet ip core user guide." [Online]. Available: https://www.intel.com/content/www/us/en/programmable/products/intellectual-property/ip/interface-protocols/m-alt-40gb-ethernet.html

[99] Intel Altera, "Avalon interface specifications." [Online]. Available: https://www.intel.com/content/www/us/en/programmable/documentation/nik1412467993397.html

[100] Mellanox Technologies Ltd., "Sn2100 open ethernet switch," 2019. [Online]. Available: https://www.mellanox.com/ethernet/switches.php

Acknowledgments

In the course of this research, the help and cooperation of numerous individuals have proven to be invaluable. I am sincerely and heartily grateful to my advisors, Professor Kentaro Sano and Professor Hiroyuki Takizawa, for the support and guidance they have showed me throughout the entire process of this undertaking. I am sure this would have not been possible without their help. To Professor Masanori Hariyama and Associate Professor Ryusuke Egawa for taking the time to review my work, their comments and suggestions are very well appreciated. I am also grateful to Ministry of Education, Culture, Sports, Science, and Technology of Japan (MEXT), for the opportunity and financial support. To the staff members, team members, students, and former students of Takizawa-Egawa Laboratory in Tohoku University and Processor Research Team (Sano Team) in RIKEN R-CCS, I thank them for being supportive co-researchers and friends. To my precious family, my good friends, and my loving husband in the Philippines, without their support that they sent me through well wishes, thoughts, and prayers, I would not have seen through this. Lastly, to God Almighty I am grateful for the gift of life and the countless blessings.

Antoniette Pangilinan MONDIGO

January 2020