2021

Enabling secure multi-party computation with FPGAs in the datacenter

BOSTON UNIVERSITY

COLLEGE OF ENGINEERING

Thesis

ENABLING SECURE MULTI-PARTY COMPUTATION WITH FPGAS IN THE DATACENTER

by

PIERRE-FRANÇOIS W. WOLFE

B.A., Skidmore College, 2015
B.E., Dartmouth College, 2016

Submitted in partial fulfillment of the

requirements for the degree of

Master of Science

2021

Approved by

First Reader

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

Second Reader

Mayank Varia, PhD
Research Associate Professor of Computer Science

Third Reader

Wenchao Li, PhD
Assistant Professor of Electrical and Computer Engineering
Assistant Professor of Systems Engineering

Pitchote a migote. Petit à petit, l'oiseau fait son nid.

Acknowledgments

I wish to first thank my advisor, Prof. Martin Herbordt, for all of the guidance and insights he has provided me throughout my master's program. I recall his enthusiasm in our first conversation over the phone when I was considering various programs to apply to, and I have felt that carry through all our work together on accelerators these past two years. I must express my gratitude to Prof. Mayank Varia for his patience and precision in expanding my knowledge of cryptography. It was particularly important for me to quickly get up to speed with MPC in order to undertake this work. I appreciate Prof. Wenchao Li agreeing to join my committee and providing a fresh perspective with insightful recommendations as I finalized my thesis.

I must acknowledge Rushi Patel who has worked closely with me throughout and whose efforts were critical to successful implementation and testing. I welcomed all of the honest feedback and suggestions Robert Munafo provided as I was planning each step of my work. To all the other members of the CAAD lab, current and graduated, thank you. You made me feel welcome and were happy to discuss any topic with me and answered questions I had.

I wouldn't have been able to undertake this course of study without the support of Dr. Jeffrey Herd, Dr. Bradley Perry, and Dr. Charles Coldwell at MIT Lincoln Laboratory and the opportunity afforded by the Lincoln Scholar Program. Thank you, I look forward to resuming my work with you this summer.

To my family and friends, I appreciate your support and hope to see you all again soon. To my fiancée, Erica Heinz, thank you for all the support through this journey as you are also undertaking the challenges of law school. I am rooting for you as you finish your studies this next year, just as you have for me!

—Pierre-François Wolfe

DISTRIBUTION STATEMENT A. Approved for public release. Distribution is unlimited. This material is based upon work supported by the United States Air Force under Air Force Contract No. FA8702-15-D-0001. Any opinions, findings, conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the United States Air Force.

ENABLING SECURE MULTI-PARTY COMPUTATION WITH FPGAS IN THE DATACENTER

PIERRE-FRANÇOIS W. WOLFE

ABSTRACT

Big data utilizes large amounts of processing resources requiring either greater efficiency or more selectivity. The collection and managing of such large pools of data also introduces more opportunities for compromised security and privacy, necessitating more attentive planning and mitigations. Multi-Party Computation (MPC) is a technique enabling confidential data from multiple sources to be processed securely, only revealing agreed-upon results. Currently, adoption is limited by the challenge of basing a complete system on available software libraries. Many libraries require expertise in cryptography, do not efficiently address the computation overhead of employing MPC, and leave deployment considerations to the user.

In this work we consider the available MPC protocols, changes in computer hardware, and growth of cloud computing. We propose a cloud-deployed MPC as a Service (MPCaaS) to help eliminate the barriers to adoption and enable more organizations and individuals to handle their shared data processing securely. The growing presence of Field Programmable Gate Array (FPGA) hardware in datacenters enables accelerated computing as well as low latency, high bandwidth communication that bolsters the performance of MPC. Developing an abstract service that employs this hardware will democratize access to MPC, rather than restricting it to the small overlapping pools of users knowledgeable about both cryptography and hardware accelerators. A hardware proof of concept we have implemented at BU supports this idea. We deployed an efficient three-party Secret Sharing (SS) protocol supporting both Boolean

and arithmetic shares on FPGA hardware. We compare our hardware design to the original authors' software implementations of Secret Sharing and to research results accelerating MPC protocols based on Garbled Circuits with FPGAs. Our conclusion is that Secret Sharing in the datacenter is competitive and, when implemented on FPGA hardware, is able to use at least 10× fewer computer resources than the original work using CPUs. Finally, we describe the ongoing work and envision research stages that will help us to build a complete MPCaaS system.

Contents

List of Tables

List of Figures

List of Abbreviations

2PC	.....	Two Party Computation
3PC	.....	Three Party Computation
ACID	.....	Atomicity, Consistency, Isolation, and Durability
AES	.....	Advanced Encryption Standard
AI	.....	Arithmetic Intensity
AMBA	.....	Advanced Microcontroller Bus Architecture
AMD	.....	Advanced Micro Devices, Inc
API	.....	Application Programming Interface
ARM	.....	Advanced RISC Machines
ASIC	.....	Application-Specific Integrated Circuit
AWS	.....	Amazon Web Services
AXI	.....	Advanced eXtensible Interface
BASE	.....	Basically Available, Soft state, Eventually consistent
BU	.....	Boston University
BW	.....	Bandwidth
CAAD	.....	Computer Architecture and Automated Design (Lab)
CaaS	.....	Container as a Service
CAP	.....	Consistency, Availability, and Partition tolerance
CCPA	.....	California Consumer Privacy Act
Ceph	.....	Cephalopod (Software Storage Platform)
CPU	.....	Central Processing Unit
CUDA	.....	Compute Unified Device Architecture *(Nvidia trademark, no longer used as an acronym)*
DMA	.....	Direct Memory Access
DRAM	.....	Dynamic RAM
DSP	.....	Digital Signal Processor
EC2	.....	(Amazon) Elastic Compute Cloud
FaaS	.....	Function as a Service
FedRAMP	.....	Federal Risk and Authorization Management Program
FPGA	.....	Field-Programmable Gate Array
FPL	.....	International Conference on Field-Programmable Logic and Applications
GC	.....	Garbled Circuit
GDPR	.....	General Data Protection Regulation
GFS	.....	Google File System

GPGPU	.....	General-Purpose computing on Graphics Processing Unit
GPU	.....	Graphics Processing Unit
HDD	.....	Hard Disk Drive
HDL	.....	Hardware Description Language
HIL	.....	Hardware Isolation Layer
HIPAA	.....	Health Insurance Portability and Accountability Act of 1996
HLS	.....	High-Level Synthesis
HPC	.....	High-Performance Computing
HPEC	.....	High Performance Extreme Computing conference
IaaS	.....	Infrastructure as a Service
IBM	.....	International Business Machines Corporation
IC	.....	Integrated Circuit
IoT	.....	Internet of Things
IP	.....	Intellectual Property
IT	.....	Information-Theoretic
LAN	.....	Local Area Network
MAC	.....	Multiply ACcumulate
MGHPCC	.....	The Massachusetts Green High Performance Computing Center
MIPS	.....	Microprocessor without Interlocked Pipelined Stages
MIT	.....	Massachusetts Institute of Technology
ML	.....	Machine Learning
MLC	.....	Multi-Level Cell
MM	.....	Matrix Multiply
MOC	.....	Mass (Massachusetts) Open Cloud
MOP	.....	MPC OPeration
MOSFET	.....	Metal–Oxide–Semiconductor Field-Effect Transistor
MPC	.....	Multi-Party Computation
MPCaaS	.....	MPC as a Service
MPI	.....	Message Passing Interface
NIST	.....	National Institute of Standards and Technology
NIC	.....	Network Interface Card
NVRAM	.....	Non-Volatile RAM
OCT	.....	Open Cloud Testbed
OCX	.....	Open Cloud eXchange
OpenCL	.....	Open Computing Language
OS	.....	Operating System
PaaS	.....	Platform as a Service
PCI	.....	Peripheral Component Interconnect
PCIe	.....	PCI Express

PHI	.....	Protected Health Information
PII	.....	Personally Identifiable Information
POWER	.....	Performance Optimization With Enhanced RISC
PRF	.....	Pseudo-Random Function
PUF	.....	Physical Unclonable Function
QLC	.....	Quad-Level Cell
RAM	.....	Random-Access Memory
RapidIO	.....	*An interconnect architecture*
RDMA	.....	Remote DMA
RISC	.....	Reduced Instruction Set Computer
RNG	.....	Random Number Generator
R&D	.....	Research and Development
SaaS	.....	Software as a Service
SDK	.....	Software Development Kit
SDN	.....	Software Defined Networking
SFE	.....	Secure Function Evaluation
SHA	.....	Secure Hash Algorithms
SLC	.....	Single-Level Cell
SQL	.....	Structured Query Language
SRAM	.....	Static RAM
SSH	.....	Secure Shell Protocol
SS	.....	Secret Sharing
SSD	.....	Solid State Drive
TCP/IP	.....	Transmission Control Protocol / Internet Protocol
TLC	.....	Triple-Level Cell
VLAN	.....	Virtual LAN
VM	.....	Virtual Machine

Chapter 1

Introduction

1.1 Overview

Many rapid changes are taking place at all levels of computing. This includes a continuous growth in the quantity of data that is collected from the environment, systems, and individuals. The ideal of fungible computing resources, greater efficiency, and convenience offered by Cloud Computing has resulted in the growth of data centers which cluster compute. Hardware progress has moved beyond the predictable performance gains achieved while Moore's Law and Dennard Scaling held true. Subsequently, there has been a greater focus on parallelization with multiple processors as well as using specialized silicon in heterogeneous architectures. All the while, strides have been made in improving the capabilities of communication between systems. New analytic techniques continue to be developed, and existing ones are refined and optimized to make better use of the evolving hardware. With all of the opportunities that come from having more information, connectivity, and processing innovations, there are also risks. More sensitive information is captured and must be kept safe at all times. Keeping attackers at bay requires more careful planning as the highly connected systems that exist present a larger attack surface and more opportunities for errors to have a severe impact. Security must also provide nuanced control of valid user access to data, including both processing and viewing it. Only blocking malicious actors is insufficient and doesn't provide user need-to-know or data privacy guarantees which are critical on everything from company IP to individuals'

PII. Thankfully a wide range of cryptographic tools can be used as part of careful system design to address these challenges.

1.2 Example Scenarios

Sensitive data is everywhere around us: personal, financial, medical, government, and more. However, we regularly hear about costly cybersecurity compromises with some estimates indicating a worldly cost in the trillions (Morgan, 2021). Both companies and individuals lack confidence when it comes to maintaining the privacy and security of their data (Brooks, 2021). Industry, governments, and individuals can all greatly benefit from sharing data and performing joint computations but face risks doing so. Here are a few illustrative scenarios:

HIPAA compliance in the healthcare industry is a good example. Under certain circumstances, data held by a "Covered Entity", a healthcare provider, plan, or clearinghouse, may be shared without explicit written authorization by patients. These scenarios include exchanges between entities for specific goals, such as assessing and improving the effectiveness of treatment protocols, and combating fraud. However, shareable Protected Health Information (PHI) is strictly limited to only information that is immediately relevant to the insight sought and that belongs to patients shared by the entities. Even when permitted under HIPAA, only the minimum amount of PHI needed to complete a task may be shared. The use of tools that can allow for operations on encrypted data, revealing only final outputs, would make it easier to employ data while meeting these directives without fear of over-sharing.

Aside from healthcare, another example is government organizations. These all have strong data confidentiality requirements and have in many cases been limited to using private in-house clusters for processing. Alternately, special accommodations where cloud offerings are separated from the public can sometimes be used but require

extra effort and overhead. This "walled garden" approach can cause fragmentation in hardware, slow update cycles, and high costs when compared to more open cloud offerings. Securing public clouds such that they are acceptable for government use would ease hardware and software system upgrades, include access to new cloud services, and provide better performance. There is recognition of the benefits that the public cloud could offer governments, and some efforts are underway to offer a streamlined but security-conscious path for adoption. The US Federal Government's FedRAMP is one such approach (VanRoekel, 2011). Nonetheless, additional tools and techniques are always needed.

Unfortunately, the choice that is often made is to either not use or share data in the interest of keeping it safe or, alternately, to distribute, compute with, or otherwise employ data even though it is at greater risk of compromise. In some cases, organizations may choose to share data broadly because individuals bear most of the risk. Consumers are becoming more aware of the collection and dissemination of their data and are pushing for more control and accountability as seen in the GDPR and CCPA (Anant et al., 2020). Furthermore, insights about demographics, salaries, or other sensitive information may also pose a risk to companies, particularly if they seem to be lagging competitors.

Beyond what has already been mentioned, there are additional risks associated with data sharing, storage, and processing in the cloud. Potential cloud tenants may not trust certain providers, data partners, co-located peers, or the network being used while still desiring to gain the benefits of a curated, highly available, and scalable environment. We are then left to consider what algorithms and approaches can help to decouple what appears to be a forced trade-off between processing data and keeping it safe while also making it possible to use cloud resources with confidence.

1.3 Motivation & Goals

Thankfully, many options exist for systems to address varying levels of trust and risk while enabling the safe use of shared data. We propose steps toward a complete solution for secure cloud environments which maintains data confidentiality while allowing for joint computation over that protected private data with other co-located parties in the same data center.

The critical cryptography that enables this is Multi-Party Computation (MPC) (also known as SFE) which allows parties to securely share confidential data and compute collective information without ever releasing one's own personal data. As defined in pending legislation within the United States Senate, "the term 'secure multi-party computation' means a computerized system that enables different participating entities in possession of private sets of data to link and aggregate their data sets for the exclusive purpose of performing a finite number of pre-approved computations without transferring or otherwise revealing any private data to each other or anyone else" (Wyden, 2019) In this manner, MPC makes it possible to glean insights from large, private data repositories. These might otherwise be missed when limited to smaller pools of publicly-shareable data or could require parties to duplicate efforts to collect or generate more information.

Cloud usage for confidential data has been a difficult proposition as potential clients can't accept the exposure to other denizens or the controlling parties. Public providers can offer isolated versions of their typical offerings, or organizations can deploy their own clouds. However, such specialized setups may not compete with the scale of computational power available in a public system and cost more by limiting the economies of scales possible in large installations shared between tenants.

1.4 What's wrong with MPC?

Given the features of MPC that can address many of the challenges with computing on data while keeping it secure and private, one is left to question why it is not currently in wider use. There are several difficulties facing MPC. These include the inherent computational overhead necessary when protecting information. More resilient protocols require more resources though they offer stronger guarantees against more advanced adversaries. Though algorithmic improvements have made it possible to conceive of practical implementations of MPC there is still a performance gap that needs to be narrowed. Beyond computation, communication is another limiting factor. Generally speaking, MPC approaches that make efficient use of communication bandwidth experience communication latency in an amplified fashion while those that don't suffer from the latency as much are less efficient in their use of bandwidth. Somewhat cyclically, the systems that require low-latency are discounted because they suffer from latency when implemented on software and run on CPUs resulting in more attention going to the less bandwidth efficient approaches. Those have seen some acceleration efforts but they are still hampered by inefficient use of communication bandwidth. However, the focus on them continues since acceleration work already exists. This circular rut needs to be escaped. If a fast, hardware accelerated MPC implementation, that is relatively easy to use exists, it could bootstrap a positive cycle of improvements and be more accessible to users. MPC currently requires at least some level of experience with cryptography. Providing hardware acceleration can help to make it more viable to use algorithms but if it requires both cryptography and hardware knowledge, the pool of potential users remains very small.

1.5 Proposed Solution

This thesis proposes that Secret Sharing algorithms are best suited for Multi-Party Computation systems built in data center environments. Low-latency communication available to systems within a data center allows Secret Sharing to be selected instead of Garbled Circuits. Given the same communication bandwidth between parties, Secret Sharing maximizes throughput as less data is required for each logical gate processed. FPGAs are the ideal architecture for MPC parties in this scenario because they avoid introducing unnecessary latency, support high communication bandwidth, and can execute custom MPC primitives. Furthermore, they are more efficient than CPUs or GPUs and more flexible than ASICs. By maximizing throughput, cloud providers can build practical systems that offer access to MPC to their clients which will ease and increase adoption.

1.6 Results & Contributions

This thesis demonstrates the performance improvement that can be obtained by operating Secret Sharing protocols by FPGAs in a data center. This research aims to encourage further work towards making this system a reality, improving access to MPC by all types of users. The resulting contributions can be distilled into the following two items:

- Demonstrating the high performance of SS MPC on FPGAs in the data center.
- Providing clear steps to achieve a viable system offering easily usable MPC.

Initial work focused on better understanding the landscape of MPC research and technologies. This led to an interest in Secret Sharing based algorithms which appeared to offer the best overall throughput among protocols when employed in an ideal environment. With the main limiting factor being latency, networked devices in

close proximity appeared optimal. As FPGAs offer great connectivity, low latency, and can be tuned to efficiently perform specific computations, they seemed well suited for the task. In sum, we theorized that we would obtain compelling performance from linked data center FPGAs running SS MPC and set out to verify this hypothesis.

Next, a particular SS MPC algorithm with a simple design that offered high performance was selected (Araki, Furukawa, et al., 2016). An FPGA proof of concept was implemented, tested, and the results compared to the software implementation results from the authors of the original algorithm. We concluded that Secret Sharing MPC in this type of scenario was competitive with Garbled Circuit MPC implementations on FPGA and furthermore, that at least an order of magnitude fewer resources could be used relative to the software on CPU tests performed by the original authors (Patel et al., 2020; Wolfe et al., 2020).

Subsequent work focused on some of the elements necessary to bringing MPC to the data center. This primarily took place under the umbrella of a BU Cloud Computing project course using resources available through the Massachusetts Open Cloud (MOC) and through CloudLab. Prior to the course, the project mentors had created a clean MPC implementation in C, which helped to avoid the performance penalties suffered by some other implementations when relying on many dependencies (Liagouris et al., 2021). Student team members focused on automating deployment to different cloud platforms and performing a preliminary assessment of the differences between environments by improving benchmarking and profiling instrumentation (Rehman et al., 2021). Several Ansible playbooks and detailed documentation were delivered and are currently being used by the mentors in their research. Greater familiarity with cloud resources gained through this experience helped when crafting a detailed vision of the steps necessary to complete a full MPCaaS system.

Work on extending the existing FPGA implementation is ongoing. The long-term

project plan we developed is composed of three main thrusts. The first is enabling MPC in a cloud environment such that the hardware and software used properly provision the resources, isolate the tenants, and make it possible to execute secure MPC operations. This system should seek to account for vulnerabilities and not make assumptions suitable only for experiments. The second is to enable transparent use of FPGAs when MPC operations are called assuming a securely provisioned environment. This will require the selection and modification of some API so that it interfaces with the implemented hardware primitives. The third is to make it possible for an arbitrary application of interest to make transparent use of MPC. While the second thrust should make it possible for a user without FPGA expertise to employ the hardware in MPC, the third one seeks to address a wider pool of possible users and allow someone with little to no knowledge of MPC to solve their problem securely. While a fully general system would be ideal, a wide-open scope is challenging. Picking a focus makes it more manageable. Machine learning (ML) is one field actively being explored with MPC (Du & Atallah, 2001; Knott et al., 2020; Ohrimenko et al., 2016; Zhao et al., 2019). Using MPC primitives deployed on FPGAs to construct operations that serve ML appears to be a fruitful domain for work. With that in mind, current work is focusing on implementing and demonstrating the effectiveness of such hardware designs. Additionally, work seeking to automatically convert ML problems into an MPC format and efforts to adapt existing secure environment provisioning (Mosayyebzadeh et al., 2019) will enable the creation of a production system accessible to a broad range of users.

In summary, we have theorized and then demonstrated the performance potential of Secret Sharing MPC on FPGA hardware on a low-latency network. We developed a Cloud Computing vision for MPCaaS with specific milestones. This architecture would enable easier use of MPC, spurring adoption as a means of addressing some of

the persistent secure data processing needs driven by the growth in big data. While these thrusts notionally build upon each other, some exploratory work is progressing on them in parallel.

1.7 Organization

Following this introduction (Chapter 1), the rest of this thesis is organized as follows: Chapter 2 provides the reader with insights about the development and current status of MPC, processing hardware, network communication, and cloud computing. This information establishes the evolution and current features of the data center context. Big data is shown to be a driver for more efficient processing and greater security. Chapter 3 considers design motivations and details hardware testing including design choices made when implementing MPC primitives on FPGAs. Chapter 4 covers hardware testing. The data is analyzed and compared to results obtained by the original authors of the protocol as well as to other related research. Chapter 5 steps away from the low-level hardware and instead considers system level design for MPCaaS. This includes the efforts related to provisioning and configuring an environment in a data center. Finally, Chapter 6 offers some specific steps towards achieving a complete MPCaaS and provides a final commentary on the research work accomplished.

Chapter 2

Background

2.1 MPC & Secure Computing Algorithms

2.1.1 History

Though MPC has recently gained sufficient attention to be defined in the United States Senate in 2019, the field has been actively researched over the past 40 years (Evans et al., 2018; Shamir, 1979; Yao, 1986; Yao, 1982) with more practical implementations explored since the early 2000s. Research deployments of MPC have been used to protect data in different fields including healthcare (D. W. Archer et al., 2018; Giannopoulos & Mouris, 2018), education (Bogdanov et al., 2016; Feigenbaum et al., 2004), finance (Abidin et al., 2016; Bogetoft et al., 2009; Damgård et al., 2017), and technology (Bonawitz et al., 2017; Ion et al., 2019). Several companies have brought commercial MPC offerings to the market, but many are customized for each client and require expert knowledge to deploy and support (D. W. Archer et al., 2018; Hastings et al., 2019).

In order to preserve data privacy, MPC fundamentally requires additional communication and computation overhead beyond that which the base computation requires. Consequently, to create a truly general-purpose MPC system which is easy to adopt, it is necessary to continue to improve the performance and ease of use while supporting more types of problems. Specialized MPC systems will remain important for performance critical applications and may offer a reasonable and focused starting point for research in the short term. General MPC with reasonable performance

will be needed to increase adoption by being easier to use and less costly to deploy. Furthermore, if a particular task doesn't need cutting edge speed, a general MPC offering may be selected even by users already familiar with MPC simply because of the relative ease to deploy vs. a more sophisticated solution. Existing work has shown that acceleration of general-purpose MPC can translate into viable systems (Y. Huang et al., 2012), which is encouraging for this vision.

To date, most MPC implementations are in software, and thus rely on general-purpose processing hardware and commodity networking equipment. In such cases, Secret Sharing tends to be network latency-bound whereas Garbled Circuits are often compute-bound. Consequently, most of the prior focus in hardware acceleration has been directed toward Garbled Circuits as they appear a more obvious target for acceleration. We'll discuss prior research further later on. First, we'll consider some of the frameworks that currently exist and what they each offer.

2.1.2 Taxonomy & Features

MPC protocols can be designed to accommodate an arbitrary number of compute parties N (where $N > 2$) while tolerating some selected threshold T of those parties being adversarial. A coalition of "bad" parties may be working together in an attempt to break confidentiality in order to learn about other entities' data or in an attempt to tamper with the computation being performed.

While it has been known for 30 years that general-purpose MPC algorithms existed and could be used to securely solve arbitrary problems, most of that work remained theoretical until recently (early 2000s) as inefficiencies in the design of the protocols prevented practical use (Hastings et al., 2019). Specialized protocols that could be feasibly implemented were also researched during this time. While some of those have more reasonable computing hardware requirements, the restricted scope of problems they served still limited adoption. Lately, work on compilers that can automate the

application of MPC to arbitrary problems has made general-purpose MPC feasible to implement and opened an avenue for the development of systems that could see wider adoption. Fairplay was the first notable implementation to show this was possible (Malkhi et al., 2004).

MPC designs generally represent problems as either arithmetic or Boolean circuits using some form of Garbled Circuit or Secret Sharing to secure the data which will be used in a computation. Garbled Circuit protocols are two party compute (2PC) systems. They rely on one party generating an encoded circuit representing the entire problem of interest. This is transmitted to the second party along with any encoded inputs held by the first party. The second party can then evaluate this circuit using both parties encoded inputs and eventually obtain the solution. On the other hand, Secret Sharing-based MPC systems have the compute parties evaluate each gate of the circuit in parallel on their own pieces or *shares* of the data. A small amount of network communication between parties is required for each multiplication or AND gate (none is required for addition or XOR gates). It should be noted that the number of parties only describes the systems being used to evaluate the protocol. While GC designs are essentially 2-party and SS designs can be n-party (commonly 3 or 4 party), there can be any number of clients contributing data and outsourcing the computation to some set of compute parties running the MPC protocol. The client data must just be encrypted and partitioned among the compute parties.

The computation and communication overhead of MPC manifests itself differently for Garbled Circuits and Secret Sharing. The size of the encrypted representation when using GC is such that each gate is a multiple of the number of bits of the chosen security parameter. For example, a single two-input gate would be $4 \times \kappa$ bits because each input could either be 0 or 1 requiring four possible inputs to be generated. If AES-128 were being used where $\kappa = 128$, the amount of data for a single gate of

this sort would be $4 \times 128 = 512$ bits. Even with optimizations (Beaver et al., 1990; Kolesnikov & Schneider, 2008; Naor et al., 1999; Yakoubov, 2017; Zahur et al., 2015), Garbled Circuits have a large communication size (80-128$\times$ the size of the original data); however, they benefit from a small constant number of communication rounds. Consequently, the theoretical max throughput of GC designs over some network is constrained by the bandwidth overhead. The fixed number of communication rounds does give GC the edge in high-latency scenarios despite the overhead, but in an ideal network with low latency, having fewer communication rounds doesn't compensate for the bandwidth cost.

Conversely, Secret Sharing approaches require a low-latency environment because they involve many more rounds of communication, but they consume substantially less bandwidth per computational step. This approach offers a higher potential maximum throughput by avoiding the bandwidth overhead of Garbled Circuits. However, Secret Sharing schemes have a number of communication rounds that increases in proportion to the depth of the circuit being evaluated. This results in the latency also growing with circuit depth. On a high latency network, the time cost is amplified. However, the amount of data that needs to be exchanged per round of communication is small, resulting in low bandwidth requirements. In comparison to GCs, this means that Secret Sharing can achieve a higher throughput and make better use of the resources available when used on a more ideal, lower latency, network.

Beyond the basic choice of protocol, there are numerous other considerations to make before selecting a specific algorithm variant or implementation. Some are as follows:

Different cryptographic schemes offer varying security guaranties. Approaches that promise information-theoretic (IT) security indicate that even with an infinite amount of computational power at their disposal, an adversary would not be able

to compromise the design without certain information. Alternately, computationally secure designs rely on how difficult certain calculations are to perform in order to provide a guarantee about a system being secure from compromise. While some ciphers have a "best by" date based on anticipated computational performance, others have such a high threshold as to be practically secure forever (barring advancements such as quantum computing).

As MPC protocols are concerned with protecting honest parties from adversarial parties, it is important to consider how many possible adversaries of the total N parties it is able to handle. In 2PC cases such as GC, it is necessary to consider the worst case where $n - 1$ parties are corrupt and there is only a single honest entity. However, when there are additional parties, a typical choice is to select supporting an honest majority such that only fewer than $\frac{n}{2}$ of n parties may be adversarial. With an honest majority assumption, all functions can be performed in an IT secure fashion, which is not necessarily the case otherwise (Ben-Or et al., 1988; Chaum et al., 1988; Evans et al., 2018).

Selecting the number of parties for a particular protocol can be influenced by several considerations. While there is no theoretical limit to the number of compute parties that can be supported, in practice, the more parties that exist, the greater potential there is for one or more of them to experience some sort of failure. This is exacerbated when there are large geographic distances or high latencies between parties such as in systems operating over the internet. There is also additional complexity in handling the large number of entities. It is relatively common to see three- and four-party constructions because they avoid excess complexity and, in the three-party case, map very neatly to an honest majority assumption where $\frac{2}{3}$ parties at least are expected to be honest.

The types of adversaries can also vary, and different protocols have varying as-

sumptions about which might be encountered and can be handled. Semi-honest adversaries will follow the rules of a given protocol correctly but may be attempting to uncover information by performing other work on the side based on observations they make. Malicious adversaries can actively be modifying data or diverging from expected steps in a protocol in an effort to prevent the computation, gain information, or influence the result. While it might seem easy to say that a maliciously secure protocol is preferable, there are many scenarios where parties may be sufficiently trusted to follow a protocol or may be obliged to do so. Maliciously secure systems generally require additional computational overhead to provide such guarantees over semi-honest ones, and some situations may tip in favor of the additional performance of a semi-honest approach.

As noted above with regards to selecting the number of parties, the environment of operation (Global internet, single data center, or somewhere in between) will affect the performance of various protocols. Many parties will generally increase failure rate, distance will increase latency, and location of different parties may determine the amount of computational power or network bandwidth available. Some problems will require selecting a protocol that is most suitable; in other cases, crafting an ideal environment for a particular protocol is an option.

Having discussed some of the considerations of MPC algorithms, we will next look at some of the frameworks and tools that implement them.

2.2 Processing & Communication Hardware

As established earlier, MPC necessitates some overhead compared to operations performed in the clear. This manifests itself both in terms of communication and computation. In our proposed solution, we use hardware to improve the performance of MPC in the hope that a reduced overhead will be more attractive to potential

adopters. As we will cover later, changes in hardware and the way information systems are developed and deployed offer some new paths towards more accessible MPC.

2.2.1 History

For many years, using a generic CPU was sufficient for most applications because of the regular performance increases between each generation of chips. This progression is characterized by several trends that are reasonably well known. "Moore's Law," as it became known, came from the 1965 observation by Gordon Moore (Moore, 2006), then Director of Research and Development (R&D) at Fairchild Semiconductor, that the transistor count appeared to be doubling on a roughly yearly basis. He later updated this projection in 1975 to add that he expected that after 1980, the transistor count would double approximately every two years. "Dennard Scaling" describes another important observation alongside "Moore's Law" and is named after Robert Dennard who wrote about it in 1974 while working at IBM (Dennard et al., 1974). Also known as MOSFET scaling, it notes that as transistors grow smaller the power density remains constant (it relates to how much area the transistors are occupying). This essentially means that as transistors grow smaller, they can use less power. Additionally, the clock speed can be increased for those transistors, increasing performance. Taken together, "Moore's Law" and "Dennard Scaling" meant that, for a while, more transistors could fit in the same space year after year, and the individual transistor had greater performance. An estimate of this performance boost is summarized by "Koomey's law" which considers the number of computations that can be performed per joule of energy (Koomey et al., 2011).

While these trends fairly closely described the progress observed over many years, Dennard Scaling came to an end around the early 2000s with Moore's Law further slowing beyond his modified prediction by 2010. When working at small chip scales, the leakage current of transistors increased and resulted in greater heat generation

(Kumar, 2015). Consequently, the substantial frequency increases that were previously expected between generations were no longer possible. While transistor size has continued to decrease (allowing density to increase), the rate has slowed and Moore's law is expected to entirely end around 2025 as transistors reach their physical, atomic limits.

With a limited ability to increase clock speeds, adding more processing cores has been a primary strategy used to keep increasing performance in a post Dennard Scaling world. IBM was the first company to develop a multi-core processor, the POWER 4 ("Power 4 – The First Multi-Core, 1GHz Processor," n.d.), which was revealed in 2001. AMD released the first native dual-core processor in 2005 with Intel releasing their first dual-core in 2006 (where native refers to the cores being located on the same piece of silicon). Since then, core count has continued to increase such that in 2021 AMD now offers several 64 core processors while Intel has a 56-core offering. These are both architectures that make use of multiple chiplets (2×28 cores and 8×8 cores respectively). Though slightly beside the point, the single-die multicore record from Intel is 40 (Xeon Platinum 8380) or 72 (if Xeon Phi is considered). Additionally, Ampere is a company that focuses on ARM based processors and has an 80-core processor available with a 128-core release seemingly planned for later in 2021 (both single-die) (Robinson, 2021).

Crafting architectures with increasing core counts and careful tuning such as dynamically adjusting the speed of certain cores has continued to extend performance. However, fully leveraging the potential of a general architecture has become more challenging because of these many subtleties requiring additional thought to tune software for specific hardware. As a result, there has been an increase in exploring processing architectures tailored for specific tasks which make better use of the available transistor budget. The most specialized are Application-Specific Integrated Circuits

(ASICs) while Field-Programmable Gate Arrays (FPGAs) and Graphics Processing Units (GPUs) are some of the other, more popular, types of architecture. We'll look at each of these in the following section.

2.2.2 Processing Technology

Originally, GPUs were focused on providing specialized, highly parallelized silicon that would be dedicated to managing display output for a computer leaving the central processing unit (CPU) free to perform operations of interest. Researchers found that the highly parallel architecture of GPUs was a good fit for other types of computations. During the 1980s and 1990s some made use of these capabilities though additional effort was required as these devices were not designed with general computations in mind. However, the usefulness of GPUs for other work was noted, and the CUDA language and first GPU architecture designed specifically for general purpose computations were released by Nvidia in 2006 (Harris, 2015). The primary competitor to CUDA, OpenCL, which can be used to target heterogeneous platforms including GPUs, first appeared in 2009. Since then, GPUs have gained wide acceptance and use with machine learning (ML) being a large driver of interest and greatly benefiting from GPUs performance (Raina et al., 2009).

ASICs have existed since the inception of integrated circuits and are designed to serve a particular purpose. With specialization, it is possible to implement the most efficient and greatest performance solution to a problem. The drawbacks to using an ASIC include the time and effort needed to progress from an initial design to a physical chip and the difficulty of fixing or modifying an existing design. For problems whose solutions are mature and unlikely to change or where performance is required at all costs, ASICs are a reasonable choice. Other approaches do offer more flexibility and potentially lower cost from a greater production scale.

First introduced by Xilinx in 1984, FPGAs were touted as a viable alternative

to ASICs for either small production volumes or where the absolute best performance possible wasn't required. Increasing transistor densities and performance per transistor has meant that FPGAs, ASICs, and other technologies can offer greater performance. However, developing devices on smaller and more advanced scales is more complex and costly. The point at which ASICs are worth the additional cost over FPGAs has shifted ever higher, increasingly leading FPGAs to appear in final products. Taking a look at the epochs of FPGA technology, we can trace their shifting role. From their creation until the early 1990s, FPGAs were fairly small, and design automation techniques were not widespread. In more demanding applications, multi-FPGA designs were used. During the 1990s, FPGAs benefited from semiconductor process advancement increasing the number of gates that could be supported in a given area and in connectivity, which made it possible for parts of a design that are physically separated to be linked and logically in closer proximity. Design automation became crucial in effectively working with the larger quantity of resources available on an FPGA. Designs were often able to fit on a single FPGA, and ASICs faced further competition. During the early 2000s (until around 2007) FPGAs experienced the same types of challenges posed to other technologies reliant on the regular improvement to transistor performance and size. As customers more commonly had excess capacity on FPGAs at this time, producers addressed the mismatch in several ways. Smaller, less powerful, but more efficient FPGAs targeted small applications. As the largest FPGAs already had more capacity than most customers needed for individual functions and no longer benefited as much by simply growing larger, vendors started to add more dedicated functions instead. Specialized hardware blocks included integrated CPUs, DSPs, memory, high-speed serial communication, and the tools to use them. IP offerings also made it easier for customers to deploy functions to FPGAs (and to fill up the free fabric – internal resources – that they had). Current

FPGAs often consist of heterogeneous silicon designed to offer a cohesive application platform that can be targeted from the ground-up (Trimberger, 2015).

As will be seen in the remainder of this thesis, three aspects of FPGA technology are particularly relevant to this work: their use as accelerators, their use as communication devices, and their programmability. We discuss these in turn.

First, successful use of FPGAs as accelerators has been demonstrated in general (Gokhale & Graham, 2005; Hauck & DeHon, 2008; M. C. Herbordt et al., 2007; M. Herbordt et al., 2008; Khan et al., 2013) and in many specific applications such as Molecular Dynamics where GPU-like performance has been achieved (Wu et al., 2021; Yang, Geng, Wang, Patel, et al., 2019; Yang, Geng, Wang, Sheng, et al., 2019). There are also several special advantages unique to FPGAs (among off-the-shelf ICs). For example, they can be configured with arbitrary precision (Belanović & Leeser, 2002; de Dinechin & Pasca, 2011; Sun et al., 2008). This has been especially useful in cryptography (Agrawal et al., 2019; Sayilar & Chiou, 2014) and enables efficient hardware implementation as will be seen in Section 3.4.2. One of the likely applications of this work is in ML as both training and inference can be protected using MPC. Ongoing efforts seek to support important operations for ML such as MM and MAC (primarily discussed in Chapter 6). FPGAs have already proven crucial to ML, especially in handling inference with non-uniform models (Geng, Li, et al., 2020; Geng et al., 2021; Geng, Wu, et al., 2020), but also large-scale training (Geng et al., 2018; Wang et al., 2020). As ML is a field of great interest, and both ML and MPC benefit from FPGA deployment, their overlap seems like fruitful avenue for further research.

The second aspect is that, among off-the-shelf ICs, FPGAs uniquely combine accelerator capability with hardware support for communication. This has for decades made them central to high-end communication switches (Bolaria & Byrne, 2009).

Also, and critically to this thesis, in the last five years FPGAs have become widespread in the data center. Of particular relevance is their use in smart NICs and in bump-in-the-wire configurations (Caulfield et al., 2016). General purpose FPGA communication stacks have been developed both for clouds (Putnam, 2014) and clusters (Boku et al., 2019; George et al., 2016; Plessl, 2018; Sheng et al., 2015), e.g., (Sheng, Xiong, et al., 2017; Sheng, Yang, Caulfield, et al., 2017; Sheng et al., 2018). Recently this work has been extended to support using FPGAs for support of MPI (Xiong, Bangalore, et al., 2018; Xiong, Skjellum, et al., 2018) and compute-in-the-network (Haghi, Geng, et al., 2020; Haghi, Guo, et al., 2020; Stern et al., 2018). We discuss communication aspect further next in section 2.2.3.

Finally, much work has been done on FPGA programmability. When using FPGAs for low-level functions, as we do here in this thesis, precise implementation is essential. While this is possible with standard HDLs and allows for fine control, it limits portability. The standard method used to make the device more accessible to non-FPGA programmers is HLS, e.g., using the Xilinx Vitis environment or the Intel Quartus Prime environment built around OpenCL. The difficulty with such an approach is that too much performance can be lost when using such abstraction, although recent work has given direction in solving that problem (Sanaullah & Herbordt, 2018; Sanaullah et al., 2018; Yang et al., 2017).

Moving on from FPGAs and considering future processing developments, quantum computing may eventually mature and become usable commercially. Such a change promises to fundamentally alter how all systems are designed. Until that occurs though, the use of heterogeneous architectures seems to be here to stay.

2.2.3 Communication Technology

Processing capabilities are not alone in seeing progress over time; continued improvements in communication bandwidth have also been important both in serving clients

over long distances as well as maximizing the utilization of networked systems in data centers and particularly in high-performance computing (HPC) scenarios.

Edholm's law was publicly attributed to Phil Edholm of Nortel Networks by John H. Yoakum in 2004 (Cherry, 2004). The bandwidth of communication networks has been increasing at an exponential rate, doubling every 18 months, just as Moore's law dictated for transistor count. This has been attributed to three primary innovations: the advent of MOSFETs and the constant decrease in their size, lasers and their use in communication systems, and information theory (Jindal, 2009).

We can see the increasing capacity of the Ethernet over time when we consider the standards released by the IEEE 802.3 working group (Hajduczenia et al., 2016). In 1998 and 1999 1 Gb Ethernet standards were released for fiber optic and twisted pair respectively. 2002 and 2006 brought 10 Gb Ethernet to fiber and twisted pair. The standards for 40 Gb and 100 Gb Ethernet over a backplane were released in 2010 and for optical fiber in 2015. 2017 brought 300 Gb and 400 Gb Ethernet over fiber. The 2020 Ethernet Alliance roadmap ("2020 Roadmap: Ethernet Alliance," 2020) projects 800 Gb and 1.6 Tb Ethernet standards to be released in the next few years, potentially between 2023 and 2025.

In the data center, these bandwidth increases are one factor that has allowed for a growth in networked systems. The other element at play is the radix of switches. Together, the connectivity and bandwidth determine how many devices can be directly linked and what type of switch hierarchy is necessary as the system scales upwards. Higher bandwidth and radix switches make it possible to keep a flatter network topology and reduce the number of hops necessary between systems. Improvements to switch ASICs, like other silicon, have tracked Moore's law and offered a steady improvement over time. Depending on the types of tasks being executed in a particular data center, different clever network topologies can balance the distance

between nodes, bandwidth, physical cabling constraints, and more.

While the bandwidth per link has been increasing regularly, what is available to a home user when connecting to the internet lags behind. Nielsen's Law is a rule of thumb and that observes that generally a user will see their internet bandwidth grow by 50% each year. Starting from a 300 bps modem in 1983 or thereabouts, the most recent update in 2019 notes 325 Mbps (Nielsen, 2019).

Furthermore, even with the increase in bandwidth over time, latency is constrained by the physics of the speed of light travelling across fiber optics. The relative locations of networked systems will be the main determining factor for latency. Finally, in the MPC cases we are considering, there is a small (single-digit) number of connected systems. This makes it possible to avoid complex network designs, which can be critical to consider in other data center applications with larger numbers of nodes. Instead, as long as a system has sufficiently low latency (data center), the hardware implementation of MPC can perform sufficient operations to fully utilize the available bandwidth.

2.3 MPC with Special Hardware

As noted previously, GC approaches to MPC consume lots of network bandwidth but don't introduce much communication latency because they have a fixed number of communication rounds. Consequently, such approaches can work well even when parties can only communicate over a high-latency network (though bandwidth itself will influence how long the communication will take). While effort has been made to make GC representations more compact and bandwidth efficient, increasing network bandwidth has likely also contributed to reducing the urgency of the network limitations of such protocols. Indeed, prior hardware research has frequently focused on improving the performance of garbling and evaluating, including a fair bit of work on

FPGAs (Fang et al., 2017, 2019; Frederiksen et al., 2014; K. Huang et al., 2019; Hussain & Koushanfar, 2019; Hussain et al., 2018; Järvinen et al., 2010a, 2010b; Leeser et al., 2019; Songhori et al., 2019; Songhori et al., 2016).

With SS, while the network bandwidth is used sparingly, the communication latency is critical to overall performance. As latency increases with the depth of the circuit being evaluated, bigger problems will be more severely impacted. While network bandwidth has increased over time, network latency is dependent on physical distance between parties and avoiding the introduction of any delay in communication. To speed up a problem, the best strategy is to generate and process the flattest circuit possible and parallelize processing. In that manner, more data can be sent in the same communication round. The greater bandwidth efficiency does mean that given the same network a SS approach saturating the connection will have a higher throughput than a GC approach saturating the same connection.

GC and SS can both benefit from the increase in compute offered by accelerators, improved network bandwidth, and the consolidation of these resources into data centers. However, SS can better the utilization of those resources at the risk of additional latency, a trade which is likely to be appealing to data center providers seeking to extract the most work from their resources. For problems that have large circuit depth and that require the lowest latency, GC will still be preferable to SS, but this seems like a minor drawback in the data center for an MPCaaS architecture.

When considering FPGAs for MPC, there are a number of ways in which they might be deployed. As an isolated co-processor each party with an accelerator could offload work to the FPGA, potentially freeing resources for other work although, as a co-processor, performance may still be limited by the speed at which the main processor or system can handle the necessary transactions with other parties. As a bump-in-the-wire, an FPGA can potentially provide the same improvements as

a co-processor but cut out the main processor or system from communication. In a single-node cluster, multiple FPGAs may be located in the same system but belong to different parties and communicate either directly through PCIe or other interconnects like RapidIO. The main question would be with regards to whether such an approach with a shared host can provide sufficient isolation between the parties. Finally, an enclave or silo on a FPGA where multiple parties are placed on the same hardware has the best potential performance by allowing for direct communication between the parties. We see the most open questions about such an approach, however, as guaranteeing sufficient isolation between entities present on the same hardware is very challenging.

The earliest work in accelerating MPC appears to date back to 2010 with an FPGA implementation of GC (Järvinen et al., 2010a, 2010b). The authors used this to assess the potential for implementing GC on a smart-card as well as using the FPGA as a stand-alone GC accelerator. Other work explored garbling entire processors including MIPS (Songhori et al., 2016) and ARM (Songhori et al., 2019) architectures (implemented on FPGAs for testing). Specialized problem acceleration under MPC, including work on ML (Hussain et al., 2018), has also seen some interest.

Some MPC acceleration research has also explored employing GPUs (Frederiksen et al., 2014; Husted et al., 2013; Pu & Liu, 2013; Pu et al., 2011), but that hasn't seen as much attention in the last few years with most efforts focusing on FPGAs, particularly for GC (Fang et al., 2017, 2019; K. Huang et al., 2019; Leeser et al., 2019). These and other efforts (Hussain & Koushanfar, 2019) identify cloud-based FPGAs as desirable for GC evaluation with some performing testing on AWS.

The studies from researchers at Northeastern University are most relevant to the efforts in this work. Their overlay architecture (Fang et al., 2017) and choice of data centers (K. Huang et al., 2019) are similar to our own. Logical blocks which accelerate

the garbling of AND and XOR operations were implemented so that data can be passed to them without requiring the FPGA image to be recreated and reprogrammed. This approach makes it possible for one design to process different MPC circuits.

2.4 Big Data & the Cloud

The term "big data" has been in use since at least the 1990s when it gained popularity. The use of this term has most often been attributed to John Mashey in his presentations while working at Silicon Graphics (Lohr, 2013; Mashey, John R., 1999). Generally, it describes the type of datasets that defy conventional techniques of capture and analysis as a result of their scale. As hardware, software, and algorithmic improvements continue to be made, handling the growing and changing "big data" is a moving target that is never entirely solved.

2.4.1 Data Collection

One of the most obvious challenges in handling "big data" is the question of storage capacity, which can be considered from both data density and marginal cost perspectives. The storage medium influences cost, density, volatility, and read/write speed. All of these factors play into determining which type of storage is best suited to different roles, including long-term storage, short-term storage, and working space in different parts of a system architecture, such as cache, memory, or the file-system.

With density increasing and cost per unit falling, more data can be stored each year. This is the result of both some major new technologies being introduced and refinements to existing ones. The growth of solid-state storage (SSD), typically made with non-volatile NAND Flash memory, has brought huge increases in read/write performance and at a cost that has rapidly approached hard-drive (HDD) prices. While the invention of Flash memory dates back to 1980 (patented shortly after

(Masuoka & Iizuka, 1985)) and has been used over the years, its use alongside or instead of HDDs really took off around 2010.

Additionally, in the last few years persistent memory (NVRAM) has become available (Clarke, 2015). Also known under the 3D XPoint or Optane names, persistent memory is sufficiently rapid to be used as RAM. Though it is not quite as rapid as volatile DRAM, it offers better performance than standard SSDs. In summary, there is a more granular selection of storage technologies allowing for more nuanced trade-offs to be made in system architectures. Going from high to low performance (and low to high capacity) the hierarchy looks something like this: SRAM (CPU Cache), DRAM (Memory), NVRAM (Persistent Memory), SSD (existing different cell counts offering varying performance: SLC, MLC, TLC, QLC), HDD (varying platter counts and more influence performance), and tape. All of these media play a role in making it possible to more efficiently store and process "big data" by balancing capacity, cost, and performance.

2.4.2 Data Processing

Beyond the capacity challenges discussed, information must be organized in some manner when it is stored, ideally in a way that is most suitable to the types of processing tasks it will see. Some important considerations for data management systems include scalability, consistency, and efficiency. Scalability concerns how well a particular database system can be spread across differently sized hardware resources; for enormous amounts of data this can include systems networked across different geographic regions and treated as one entity. Consistency becomes more challenging with the size of a system and relates to whether data that is updated or added in one location will appear the same or different elsewhere in the system. In some cases eventual consistency, where the system guarantees that data will be the same after some amount of time, is adequate. While we won't spend much time on it here, the

trade-off space has been explored with different design philosophies such as ACID and BASE and some guiding concepts such as the CAP theorem (Brewer, 2012). Finally, efficiency concerns the ability of a system to handle different quantities and types of operations. Systems that have real-time requirements need to be sufficiently efficient to handle operations with an output rate that at least matches the input rate.

One type of approach employed for structured data was to move away from SQL-style relational databases to "NoSQL" designs (Kalid et al., 2017). Some examples of development in this field include Google BigTable (F. Chang et al., 2006), Apache Cassandra (Lakshman & Malik, 2010), and Amazon Dynamo (DeCandia et al., 2007). Broadly speaking, some data consistency properties are traded in such systems for greater performance. Alternately, a system like Google Spanner (Corbett et al., 2012) is a distributed SQL database and uses a special consensus algorithm to achieve consistency.

The use of computational frameworks improves the ability of users to work with various data storage architectures. Some of these include Google Pregel (Malewicz et al., 2010), Apache Hive (Thusoo et al., 2009), Google MapReduce (Dean & Ghemawat, 2004), Apache Hadoop (Shvachko et al., 2010), Apache Spark (Zaharia et al., 2010), PowerGraph (now owned by Apple) (Gonzalez et al., 2012), and Apache Storm (Marz, 2014). Frameworks such as these offer support for varying sets of operations, manners of decomposing (e.g., batch, streaming), and ways of organizing and distributing them across resources within individual systems and entire clusters. For example, the MapReduce concept, popularized in big data by Google and used in "MapReduce" and Hadoop, consists of breaking tasks into filtering/sorting steps (mapping) that select relevant data and then applying processing steps on that data (reduction).

An additional layer includes file-system options for storage such as GFS (Ghe-

mawat et al., 2003) and Ceph (Weil et al., 2006). Higher-level scheduling can be managed with systems like Borg (Verma et al., 2015), Mesos (Hindman et al., 2011), and Kubernetes (Burns et al., 2016). Tasks and processes can use Xen (Barham et al., 2003) or containers for virtualization. The use of SDNs and other programmable network management, such as Google's B4 (Jain et al., 2013), have helped make it possible to maximize the utilization of network resources, improve security, simplify management, and gain more insights.

2.4.3 Security & Efficiency

The use of big data raises varied security and privacy questions. Addressing these while maintaining efficiency is critical to obtaining value from the data collected. The taxonomy of big data security and privacy considerations are nicely summarized by NIST (W. Chang et al., 2019). Some of the important ideas to note are that big data sets may not be able to be managed with a single security scheme and may instead need different policies and mechanisms for various subsets. Additionally, the provenance and state of data may differ. Classic approaches to dis-identifying PII may no longer be sufficient, especially when multiple data sources are used together in ways that were not planned for. There are opportunities in different data collection mechanisms for errors or failures that can compromise the information collected, for example from many IoT devices, some of which may not be online at a given moment. Certain types of data that historically were too large for big data analysis, such as high-resolution geospatial data, can now be processed and have perhaps not been prepared or protected appropriately with the cloud in mind. While it's easy to conceptualize the cloud as something omnipresent but amorphous, the physical location of the component data centers has political and regulatory significance. Navigating various countries' laws and norms for hosts, clients, and any data conduits can be complex for both privacy and security because of the large number of stakeholders.

While there are many strategies and tools for managing these different types of risks, it is difficult for many organizations to adequately prepare and adopt safeguards. Doing so requires additional time, effort, and cost upfront and it can be difficult to convince those unfamiliar with the risks to prepare in this fashion, especially if there is no regulatory requirement that must be met or if they exist but lack teeth. Despite the work that is needed on this front, working to make secure tools easier to use (such as MPC) will help simplify addressing these challenges.

2.4.4 Data centers & the Cloud

Simply put, the concept of cloud computing is that instead of using local resources for running a service, an entity can pay to use resources that someone else owns and maintains. One reason for using a cloud provider is that it allows an organization that doesn't specialize in computer hardware, software, or services to gain access to business-critical resources without having to take on the overhead of owning their own equipment or hiring employees to handle it. With well implemented cloud systems, there is also less lead-time in getting up and running. While existing companies can benefit from cloud, allowing them to focus on their core objectives, cloud can also make it far easier for startups to get up and running. They can start with some small number of resources but scale up rapidly if business grows (or down if the model doesn't work out).

Additionally, there is at least the notion that cloud computing is, or at least should be, a fungible commodity. A consumer doesn't necessarily care who is providing the computation and storage they consume, and it should be possible to move their tasks to a different provider if there is a lower cost. However, this isn't necessarily the case with some of the large commercial providers. There are frequently low or no costs for importing data into or moving it around within a particular provider's infrastructure but higher costs to export that data to a local system or to migrate

it to some other cloud provider. Additionally, different cloud providers seek to offer specialized services rather than simply hosting standard tools. While some special tools might have unique features, users that employ them will find it much harder to leave a specific cloud provider when it is easier to just use other tools the provider has designed to be compatible. Furthermore, the additional cost to move data to another provider on top of potentially having to retool is a disincentive. There are some efforts to at least offer ways for more fungible systems to exist such as through the Open Cloud Exchange (OCX) model (Demchenko et al., 2013; Desnoyers et al., 2015).

Data centers can be located near inexpensive power sources such as hydroelectric facilities. Operating at a huge scale, it is possible to have a few workers dedicated to keeping everything running while minimizing the human cost per system. Accepting that systems are "cattle not pets" is also critical. It is easier to make the most cost-effective choice, such as running systems at a warmer temperature and accepting some additional failures because the cost of replacing a system is less than the cost of additional cooling. Keeping to that same idea, using a limited and consistent set of hardware, OSs, and other software helps to reduce the headache of managing different systems and for developers, targeting different platforms. While cloud providers can reduce operational costs, they charge users a huge markup. The capital expenses for a user to run their own system are substantial, and as humans are not fractional, a system must be large enough to justify the workers to operate it. Running a variety of different tasks can also help to ensure there is sufficient utilization as various operations go through different ebbs and surges. At a certain scale, it still makes sense to "roll-your-own."

Centralizing different tasks into a single data center also allows for surges and ebbs in compute demand to be better balanced keeping the overall utilization of the

resources higher and more efficiently serving customer needs. Distributed hardware run by individual companies is likelier to be run at a fraction of capacity and be sized to handle the peak need anticipated. This approach means that there is more idle hardware and less efficient use of resources. For large enough companies, centralizing and running their own cloud can help to achieve the same benefits that dedicated cloud providers see.

2.4.5 Deployment Models

There exists a range of deployment and use models as a result of cloud computing services ("IaaS vs PaaS vs SaaS," n.d.; Wagoner, 2019). Below some of the most common ones are summarized from the lowest level where users have control that most closely resembles managing their own hardware, to the highest level where there is the most abstraction of the underlying hardware and software.

- IaaS: Infrastructure offerings like OpenStack provide mechanisms or abstractions to allocate low-level resources such as cores, memory, and storage space. Clients can then typically run their operating system of choice on the resources they choose and can even create their own network topologies between different hardware allocated. This is the closest offering to owning and operating your own hardware.

- CaaS: Container offerings move up a layer of abstraction and allow for users to run their software in discrete virtualized environments where resources are managed by the provider. Unlike virtual machines which are entirely separate, containers can share a common OS kernel and libraries.

- PaaS: Platform offerings are mostly still beneficial for developers and are scenarios where the provider manages hardware and software and provides some

abstraction upon which code and applications can still be developed. The benefit for the user is that they can focus on their application development without being caught up in managing more details of the systems they are running on.

- FaaS: Function offerings allow users to create or use discrete widgets that are not full featured applications. These can be employed individually or plugged together to obtain the desired functionality without needing to deal with any lower-level management of the system. Additionally, discrete functions can be updated individually and still used in combination together.

- SaaS: Software offerings are the most abstract and are managed and licensed by a third party to the end user such that the details of operating and maintaining the software are hidden. In principle, updates and distribution can be managed by the provider without requiring effort on the part of the user.

2.5 Takeaways

Here are a few takeaways after having considered the history and current status of MPC techniques, computer hardware, advent of big data, and the growth of cloud computing. MPC is a field that has seen a great amount of theoretical work and that has more recently reached a point where it is feasible to implement for more than just research efforts. Hardware has seen some large changes with the paradigm for computing moving in the direction of heterogeneous processing (including accelerators), communication bandwidth increasing, and storage technology diversifying and offering a larger trade-off space between capacity and performance. The increased sources for data collection, capacity for storage, and desire for more insights related to "big data" have resulted in the need for new technological and algorithmic approaches to better use and to better protect the information gathered. The efficiency improvement and client ease of use offered by cloud computing has accel-

erated the clustering of compute power into specific, tightly knit data centers. With more data and greater interconnection between systems, there are many security and privacy risks, but also many opportunities to gain greater knowledge. Research such as the work by Schneider and Zohner (Schneider & Zohner, 2013) demonstrated the performance possibilities for Secret Sharing to compete with Garbled Circuits in a low-latency environment such as a data center. Our work, using FPGAs to accelerate SS MPC in the data center and developing an MPCaaS vision, shows that there is still more that can be done to address the performance overhead and knowledge currently required to use such a system. Moving forward, finding ways to make it easier to use techniques such as MPC will help users and companies to address risks while still reaping the benefits of all the technical innovation that has occurred.

Chapter 3

Design & Implementation

3.1 Design Goals

The primary motivation of this work is to demonstrate the viability of MPCaaS and provide a plan that can achieve that goal. It should be high performance and easy for the average user to employ. Most existing MPC implementations require substantial manual effort and expertise to configure, and they are often network bounded, particularly when executed over long distances. We want users to be able to make use of cloud resources while transparently running their applications of interest on an MPC system.

Because MPC requires multiple computing parties for security and low latency networking for performance, we consider processing hardware owned by different parties and housed within a single data center. Compute parties will each receive input data shares and then communicate throughout evaluation of problems. Other services offered by cloud providers require data input by users. Consequently, clients may already keep some of their data silos in the same data center. If the MPC service is hosted in the same location, it can also benefit from faster access to inputs. Concretely, we imagine a scenario where a small number of FPGAs are connected over high-speed interconnects and have the benefit of drawing data from servers all co-located within the data center. FPGA hardware acceleration has seen increasing adoption in data centers. This includes Machine Learning training and evaluation, image and video processing, network management, and packet analysis. FPGA

hardware properties, as described in Section 2.2.2, and co-location should yield high throughput for MPC protocols based on Secret Sharing, as such designs make the most effective use of available bandwidth. Cloud providers care about throughput because maximizing it means they are able to serve more clients using the same hardware over time. The security and reliability of the system is an important factor when relying on data center resources and needs to be considered before a MPC solution is fully adopted. We will show how our work confirms our ideas for SS MPC on FPGAs in the following sections.

3.2 Design Choices

In this work, the decision was made to focus on a simple and efficient protocol that could be used in a broad number of scenarios and that might be conducive to hardware acceleration in the data center. Hypothesizing that a Secret Sharing solution would best be able to take advantage of the low-latency communication in a data center, we focused on such candidates with the goal of verifying that concept. Eventually, a 3-party protocol tolerating 1 adversarial party who "semi-honestly" follows the protocol was chosen. The small number of computing parties simplifies communication between each other while semi-honest behavior is acceptable in many types of computation. Furthermore, additional existing algorithm work (Araki et al., 2017) addresses how such a system can be extended to provide security against malicious adversaries. Finally, having a small number of compute parties makes it much easier to imagine mapping such an algorithm to several discrete FPGAs owned by different entities and co-located in the same data center. We ruled out GPUs for this selected context because their communication capabilities are constrained relative to FPGAs.

3.3 Protocol & Algorithm Details

Within the category of MPC protocols based on Secret Sharing, we selected a protocol (Araki, Barak, Furukawa, Lindell, et al., 2016; Araki, Furukawa, et al., 2016) for FPGA acceleration due to its simplicity and algorithmic efficiency. The base Araki et al. protocol employs exactly 3 parties, and tolerates 1 adversarial party that is presumed to follow the protocol. Communication occurs in a ring topology, with all parties sending data in the same direction (clockwise or counter-clockwise). MPC schemes based on an honest-majority such as Araki et al. are straightforward to define with three parties, the minimum quantity required. Small numbers of parties are not typically limiting for most MPC applications. It should be noted that the more parties that are added and the greater the separation between them, the more likely it is that an MPC operation will experience greater latency or will fail from connectivity issues. This reinforces the practical benefits of performing MPC in a data center with a limited number of parties.

The workflow involves 3 distinct steps. First, data holders split their data into secure shares that are distributed among the 3 compute parties. The shares are constructed such that any two of the compute parties can decide to reveal data and have sufficient information in a pair of shares to do so. Then, the parties iteratively *compute* over these shares without revealing any secrets. Finally, the compute parties reveal their shares to the output party who can *reconstruct* the final answer. Alternately, they could reconstruct the answer among themselves by exchanging their shares with each other.

We implemented both Boolean and arithmetic types of Secret Sharing circuit representations as described by the authors. The distinction between these two categories is based around the usage of an algebraic ring modulo 2^n. Operations where $n > 1$ bits are used as a single value employ arithmetic gates such as addition or multipli-

cation. We opted to use $n = 128$ for our arithmetic implementation. For the case where $n == 1$, each bit is an independent Boolean value. When considering these, we used 128 independent Boolean gates to match the quantity of data in our $n = 128$ arithmetic gates. In the Boolean case, XOR is equivalent to addition and AND to multiplication. We explore the details of each step below.

3.3.1 Secret Sharing

When following the protocol of Araki et al. input data must be prepared in a particular fashion to be distributed among the three compute parties. The initial value v is processed such that each party $i \in 1, 2, 3$ obtains a share tuple (x_i, a_i). As we will see in the following steps, this will ensure that any two parties have sufficient data to reconstruct the original value using one of the values in their tuple share and the opposite value from the tuple of one of the other parties.

The entity holding the secret value v, either a client or one of the compute parties, selects random values following Eq. 3.1, essentially, values that fit in a binary domain n bits long (e.g., for an $n = 8$-bit field each random value must be 8 bits).

$$x_1, x_2, x_3 \in \mathbb{Z}_{2^n} \tag{3.1}$$

These shares must obey Eq. 3.2 if the shares being generated are arithmetic; otherwise, they are Boolean and must obey Eq. 3.3. It should be noted that the a_i component of the share tuple belonging to party i derives from x_{i-1} which is from the neighboring party. This is essentially a one-time pad and the reason that the x_i from the neighboring party can be used to reveal the original value.

Arithmetic:

$$x_1 + x_2 + x_3 = 0 \text{ then } a_i = x_{i-1} - v \tag{3.2}$$

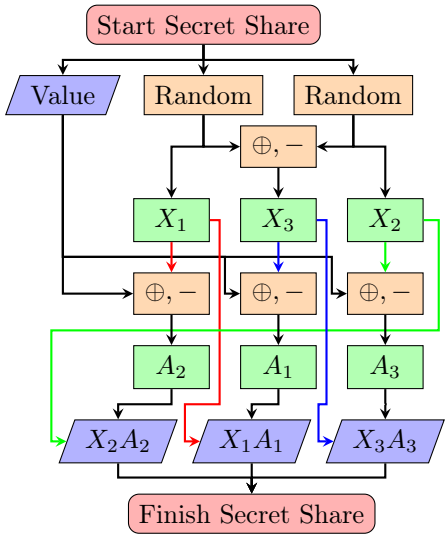


Figure 3·1: Initial Secret Sharing

Boolean:

$$x_1 \oplus x_2 \oplus x_3 = 0 \text{ then } a_i = x_{i-1} \oplus v \tag{3.3}$$

Refer to Figure 3·1 to visualize how a value holder can generate the three shares to be distributed to the compute parties.

3.3.2 Compute Phase

In the *compute phase*, the parties work together to solve a problem in a privacy-preserving manner and generate a result that is also in the secure share format. When solving a circuit in Boolean form, XOR or AND operations are used whereas when an arithmetic circuit is solved, addition and multiplication operations are used. It is easiest to imagine fan-in 2 operations proceeding sequentially with inputs (x_i, a_i) and (y_i, b_i), though we stress that this process is embarrassingly parallel. The shares are respectively for data values "v_1, v_2". Each party i will have a share for each value.

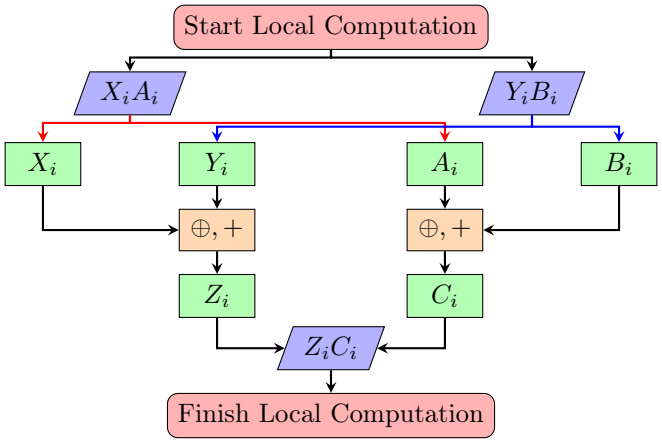


Figure 3·2: Party i's contribution toward computing an XOR gate

Addition/XOR operation

The addition/XOR operations can be performed locally by each party using the shares that they already hold. Communication is not required. Figure 3·2 provides a visual representation of these operations with two example shares (x_i, a_i) and (y_i, b_i).

XOR: Each party can compute the XOR of their individual shares simply by performing a local xor of the individual parts in the pair because a one-time pad is homomorphic under the $\oplus$ operation (see 3.4).

$$z = x \oplus y \text{ and } c = a \oplus b \tag{3.4}$$

Addition: Similar to XOR, individual parties can perform a simple addition of arithmetic shares using their local information without the need for communication (see 3.5).

$$z = x + y \text{ and } c = a + b \tag{3.5}$$

Multiplication/AND operation

The multiplication/AND operations require each party to perform a number of computations and to exchange information with the other parties for the process to

complete. Because communication is required, the depth or serial number of multiplication/AND operations is what determines the latency for solving a particular problem.

AND operation: The first step a party takes is to produce a correlated random value $\alpha \in \{0, 1\}$ which follows Eq. 3.6. That is to say, the random numbers produced by all three parties must XOR to 0; more on that later. The second step requires each party to calculate r_i by combining the halves of the two input share tuples as shown in Eq. 3.7. At this point each party passes their r_i value to one other party member (clockwise as shown here) so that each party now holds r_i and r_{i-1}. Araki et al. show that the values r_i have the property that $r_1 \oplus r_2 \oplus r_3$ equals the result of the AND gate. Hence, the 3 parties collectively know the (sensitive) result of the AND gate, but any 2 parties do not because the remaining r_i value acts as a one-time pad. Finally, each party can compute the output share (z, c); only one half requires the data from a neighbor as shown in Eq. 3.8. The shares built in this fashion maintain the invariant that any 2 of the parties can reconstruct secret values, but any 1 party cannot.

$$\alpha_1 \oplus \alpha_2 \oplus \alpha_3 = 0 \tag{3.6}$$

$$r_i = (x_i \wedge y_i) \oplus (a_i \wedge b_i) \oplus \alpha_i \tag{3.7}$$

$$z = r_i \oplus r_{i-1} \text{ and } c = r_i \tag{3.8}$$

Multiplication operation: Performing multiplication for arithmetic shares is very similar to performing AND for Boolean shares as described above. The first step is also to have each party generate correlated random values. Given the nature of arithmetic shares, these must be chosen such that $\alpha \in \mathbb{Z}_{2^n}$, following Eq. 3.9. The second step requires each party obtain r_i by combining the halves of the two input share tuples as shown in Eq. 3.10. Due to the unique nature of the shares and their

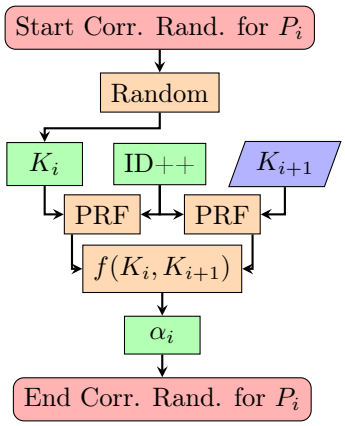
set associativity, we take advantage of the modular multiplicative inverse for our q value. As with the AND gate, communication between parties is necessary at this point with r_i being passed along. Finally, each party can finish calculating their individual share (z, c) as shown in Eq. 3.11.

$$\alpha_1 + \alpha_2 + \alpha_3 = 0 \tag{3.9}$$

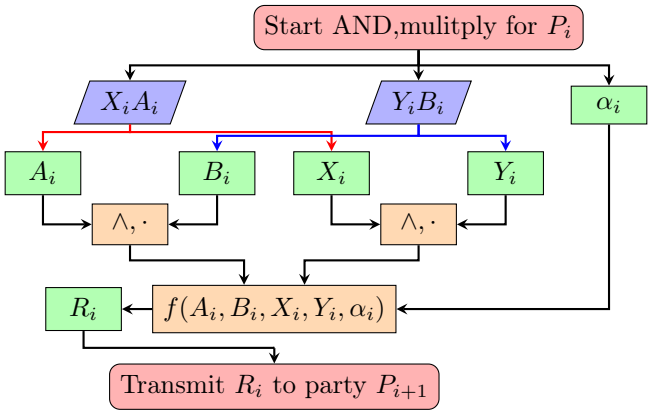
$$r_i = (a_i \cdot b_i - x_i \cdot y_i + \alpha_i) \cdot q \text{ where } q \cdot 3 \equiv 1 \pmod{2^n} \tag{3.10}$$

$$z = r_{i-1} - r_i \text{ and } c = -2r_{i-1} - r_i \tag{3.11}$$

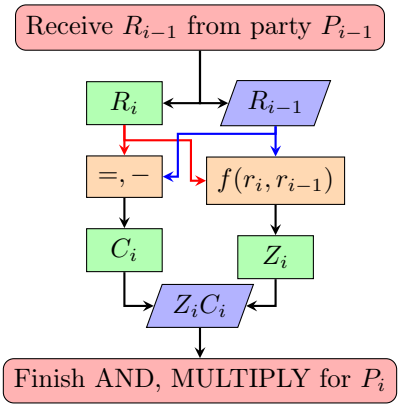
Having now considered how AND and multiplication operations can be executed, let's revisit the correlated random values. These values have nothing to do with the sensitive data, but they make it possible to perform operations on the secret shares while keeping them protected. Now, it is possible for the parties to pick random values, exchange them, and generate a new correlated random number on the fly. However, doing so would require an additional communication step. As a correlated random number is consumed for each AND/multiplication, this would mean that instead of a single communication for exchanging r_i there would be a second communication. SS already is most bottlenecked by communication latency, so avoiding this additional communication is preferable. Instead, each compute party selects their own cryptographic key at random such that $k_i \in \{0,1\}^\kappa$ where κ is the security parameter (i.e., the number of bits). Once chosen, each party shares their key with one other party (to the left) and receives the key selected by the other party. The party can use their key and the other party key along with a publicly known ID "message" as inputs to a PRF to generate correlated randomness. This avoids communication at the cost of some additional computation by each party. We use AES in counter mode as our PRF. By incrementing the counter that is local to each party in lockstep and using the same keys, the same pseudo-random value can be



(a) correlated random value



(b) initial computation and exchange



(c) final computation

Figure 3·3: Party i's contribution toward computing an AND gate

found independently. Furthermore, since using the PRF doesn't require knowing the shares that will be used, it can be computed ahead of time so that a correlated random value is ready to use as soon as the inputs are determined. Figure 3·3a) illustrates this process for one party. Subsequently, Figure 3·3b details the calculation of r_i, and Figure 3·3c shows how to obtain the final share after r_i values are exchanged.

3.3.3 Data reconstruction

In the *reconstruction* phase, we presume that the compute parties have calculated shares (x'_i, a'_i) corresponding to the output value v'. Then, the parties can obtain v' in the clear by revealing their shares (each party needs the share from one of the other parties) and combining them as shown in Eq. 3.13 for Boolean shares or Eq. 3.12 for arithmetic shares.

Arithmetic:

$$v' = z_{i-1} - c_i \tag{3.12}$$

Boolean:

$$v' = z_{i-1} \oplus c_i. \tag{3.13}$$

3.3.4 Extensions

We focus on the Araki et al. Secret Sharing using 3 parties as our starting point. Extensions of this protocol exist to provide more security over malicious individuals or allow for more party members (Araki et al., 2017; Furukawa & Lindell, 2019; Furukawa et al., 2017). As our initial goal was to provide evidence toward the benefits of accelerating MPC in the data center, we focused on the semi-honest case while leaving the option to extend our implementation to support the maliciously secure version of the protocol. Our FPGA implementation of the semi-honest protocol could also be extended with other features such as dynamically switching between share types or MPC protocols in the pursuit of greater performance (Demmler et al., 2015;

Mohassel & Rindal, 2018). Since these initial efforts, there have also been a number of four-party SS MPC protocols (Dalskov et al., 2020; Gordon et al., 2018; Koti et al., 2020) that provide malicious security and may be more interesting to implement than a three-party maliciously secure protocol.

3.4 Implementation Details

3.4.1 Platform Choices

The hardware employed for different tests changed throughout the course of this work. The very initial work in implementing MPC primitives targeted an Intel Arria 10 GX development kit installed in a CAAD research group desktop. This hardware was primarily chosen because of its availability. The design used a softcore, specifically the Nios II, on the FPGA to manage data I/O and provide the external triggering logic to operate the MPC AND primitive implemented in HDL. A custom instruction for the Nios enabled simple software control of the hardware MPC AND operation. Some initial information and insights were gathered from this testing, which will be discussed in the results section. Some limitations led to a change of hardware for subsequent research.

The second implementation makes use of Amazon Web Service (AWS) Elastic Compute Cloud (EC2) FPGAs available through "F1" type instances. This platform offered Xilinx Virtex UltraScale+ VU9P FPGAs accessible via a virtual machine in the EC2 F1 instances. The environment also included a hardware shell for software/hardware co-design between the node CPU (Intel Xeon E5-2686 v4) and FPGA. Software to control the FPGA uses provided DMA functions and PCIe function templates. This furnishes the mechanism for loading data, controlling operations, and retrieving results. The design deployed here is able to generate PCIe packets to pass in data and commands which are translated through the Amazon shell and then

passed to the hardware functions using the AXI bus. We use the general purpose AXI bus supporting a 512-bit data packet to provide a single message containing two secret share vectors (4×128-bits) prior to starting the hardware operations. The HDL design takes each AXI bus message, parses the information, and relays data to a specific implemented module.

One of the motivations for selecting to use AWS was to gain some familiarity and insight with using cloud deployed accelerators. Additionally, several choices of F1 instance exist and seemed attractive. The Amazon AWS F1 instances include options for 1, 2, or 8 FPGAs under the F1.2xlarge, F1.4xlarge, or F1.16xlarge names. Documentation describes two different inter-board communication approaches. Communication between FPGAs in the F1.4xlarge and F1.16xlarge instances is possible at 12 Gbps over PCIe by routing through the host system. Alternatively, using FPGA Direct to enable FPGA access to each other's DRAM over PCIe is possible for lower latency but requires developer implementation of the protocol and transfer engine (Services, 2016) (See FAQs.md). Additionally, FPGAs should have the ability to communicate directly over a 400 Gbps serial ring link (Services, 2016) (see AWS_Shell_ERRATA.md), but, unfortunately, we discovered that support is only tentatively planned in a future release. It might be possible to make use of the hardware in this fashion but would currently require implementing a custom system to handle this communication. In testing the proposed Secret Sharing block, we made use of an AWS F1.2xlarge instance as our initial verification, and proof of concept tests only required a single FPGA. The option remains to rent a larger instance, deploy parties to dedicated FPGAs, and employ PCIe for communication.

3.4.2 Hardware Implementation

Our HDL design work for the MPC primitives started with a focus on the AND operation for Boolean shares while targeting the Arria 10. While the platform was

subsequently shifted to AWS necessitating control logic changes, the existing work on the primitives was retained and further developed. Later work included XOR primitives and supported arithmetic shares in addition to Boolean ones. The arithmetic and Boolean designs share many common elements, as is seen in the similar logic described in section 3.3. With individual primitives, a hardware design can be used to selectively partition resources between XOR/addition gates and AND/multiplication gates. Since all gates are routed individually, connecting gates to form a specific circuit or equation can be done either through the software host process or with an agreed upon hardware change between all party members. If a reasonable mix of gates is programmed onto the FPGA, the need to create and program a new bitstream with a different design can be avoided, saving time. The implemented gates can be arranged in software to process any sequence of MPC operations.

Additionally, multiple gates could be chained together to support higher level functions such as matrix multiply or multiply accumulate (MAC). Previous work done with matrix multiply shows efficient methods of using Garbled Circuits to accelerate the generation of MPC circuitry before passing the garble tables over to the participating party member (K. Huang et al., 2019; Hussain et al., 2018; Leeser et al., 2019). Our approach to matrix multiply using SS would perform all computation jointly across the parties during run-time. One optimization is to perform all of the multiplication on shares locally prior to performing the necessary communication, avoiding multiple latency penalties for each element in an MM operation. An additional option is to take advantage of the fact that the correlated random bits produced by the PRF don't all have to remain together. In the Boolean case they are all independent while in the arithmetic case different quantities can be combined together arbitrarily. This may become more critical when seeking to most efficiently perform operations with data of a fixed precision type.

Within the MPC primitives, we make use of two different OpenCores projects (Castillo, 2004; Hsing, 2012) for Random Number Generator (RNG) and for AES operations (our choice for PRF). These cores provided a convenient, working starting point. In a production design, we would plan to utilize vendor specific RNG hard cores, perhaps dependent on a physically unclonable function (PUF), or alternately make use of a different HDL design if it can be better optimized for the targeted hardware. Considering the security of the OpenCores RNG module was not within the scope of this work but would be necessary in the future when making a selection between different vendor specific tools and designs. The MPC primitives make use of the RNG core during initial key generation, share splitting phase, and the correlated random requirements in both AND and multiply gates. The AES core is used as the PRF that is needed during correlated random value generation.

In our design, prior to performing any MPC computations, each party generates an initial random key using the RNG block. As the RNG block we use produces a 32-bit output, we concatenate four values to obtain a 128-bit long key (as required for AES-128). This key is both stored locally and shared with one other party (such that all keys are passed in the same direction around the ring). A public ID value visible to all the parties is set to 0; in the actual implementation, each party holds its own local copy. This ID ensures the correlation of the random numbers being generated and is incremented for each new value keeping the parties in lockstep. Each party then holds two keys, the ID is consistently set, and the PRF can be used.

When initializing the AES-128 core, 21 clock cycles are needed to obtain the first output. As long as new desired values are fed in on each clock cycle, the system is fully pipelined, and each additional clock cycle after initialization will produce another output. When initially implementing the MPC AND/multiplication operation, the two keys held by a party are alternated every cycle and the ID is incremented every

other cycle. Consequently, after initialization, the single AES core will take two cycles and output the pseudo-random values for both keys associated with the first ID value. The next two cycles will produce the values associated with the next ID, etc. This construction allows for one less AES core to be used, but does come at the cost of an additional clock cycle. In our results, we will see that this does not prevent very impressive results but for a system that is fully pipelined, using two AES cores allows for a new value every cycle. The function $f(K_i, K_{i+1})$ (see Figure 3·3a) used to combine the two PRF outputs into a correlated random value is pre-determined when the party selects to use Boolean (Eq. 3.15) or arithmetic operations (Eq. 3.14). Arithmetic:

$$\alpha_1 + \alpha_2 + \alpha_3 = 0 \text{ and } \alpha_i = PRF(K_i) - PRF(K_{i+1}) \tag{3.14}$$

Boolean:

$$\alpha_1 \oplus \alpha_2 \oplus \alpha_3 = 0 \text{ and } \alpha_i = PRF(K_i) \oplus PRF(K_{i+1}) \tag{3.15}$$

The MPC AND module itself consists of a few bitwise operations that produce the intermediate r_i values (Figure 3·3b). Most latency occurs in the transmission of the R_i values since the final step (Figure 3·3c) depends on them.

Based on the minimal data dependencies for the AND/multiply operation, in principle, a fully pipelined HDL implementation could consume and produce a new pair of input shares each cycle. Using AES block size as we are, that would allow each module to output 128 Boolean gates (or some combination of arithmetic gates adding to 128) each clock cycle. Such a design could saturate a 10 Gbps network connection if operated at 78.13 MHz or greater. While the implementation tested does not quite reach one output per clock cycle, this thought experiment does give an idea of the ability of a SS MPC design to maximize the utilization of available bandwidth. As we will see as we consider test results, this is also achievable while leaving FPGA fabric free for other purposes.

Chapter 4

Testing & Results

4.1 FPGA Testing

The following sections will describe various stages of development and testing MPC SS primitives on an FPGA. This includes the following: a first test considering a single party and Boolean AND module to verify function and resource utilization, another which tested multiple parties and AND modules and which duplicated these sets of three to again consider resource utilization, a final set of tests again using three parties but which improved the MPC primitive performance and extended support to both arithmetic and Boolean shares.

Given our focus on maximizing throughput, our multi-party design seeks to obtain an idea of the upper bound for maximum utilization of a single FPGA exclusively performing MPC. While we eventually plan to deploy parties across multiple FPGAs connected in a ring, our test environment currently consists of placing three parties on a single FPGA as this avoids limiting our communication bandwidth, allowing us to find the maximum possible performance. We do note that while multiple compute parties sharing a single FPGA is attractive for offering the best possible real-world performance, we only use such a configuration for the performance testing as previously stated. There are many more security challenges that require research to verify that the parties can be adequately isolated in such a configuration. More practically, our tests can help to determine how best to allocate resources and size the network for a multi-FPGA MPC deployment.

The design includes the necessary routing and control logic to perform calculations between all parties from start to end without software intervention. Again, because this was performed as a test on a single FPGA, there is also a single host computer that is used to load the data and operations onto three parties. In a final implementation, each FPGA would have its own host system which would handle data and operation loading.

With regards to the data we collect and present, a de facto metric of performance of MPC within the cryptography community is the rate of computing the Advanced Encryption Standard (AES). This exists as a Boolean circuit comprising thousands of AND and XOR gates. Because the performance of MPC depends largely on the number of AND and XOR gates and only minimally on its topology, knowing MPC's performance on AES provides some insight into performance on other computations. We consider our tests in the context of this and other metrics; more on that in Section 4.3.

4.1.1 Test with Arria 10

As mentioned in Section 3.4.1, initial testing targeted an Intel Arria 10 FPGA and used the Nios II softcore to operate test software for loading data and triggering the MPC AND hardware. Using a basic implementation of the Nios II Custom Instruction required a multi-cycle approach because of the limited data width that could be passed with each individual command. There are circumstances where a softcore is viable. In our case, to avoid the multi-cycle latency penalties, we would perhaps have used local storage, which could be accessed with a wider data bus, or other techniques to avoid having to load individual data values and an operation command in sequence.

Regardless, we did decide to eventually move to a different platform, but our work with the Arria 10 did provide some initial insights. With a single MPC AND design synthesized in Quartus, targeting the Arria 10 (10AX115S2F45I1SG) on our

development board, we found the following: The most constrained resource for a single AND was the 704.5 Kb of M20K block memory consumed post-synthesis, making it possible to estimate the utilization to be ∼1.32% based on the total 53.260 Mb of M20K available on the Arria 10. Maximizing use of the fabric, this would permit ∼76 instances, or perhaps more realistically using 70% of the fabric might allow for ∼54 instances of the MPC AND. At this point in development, the MPC AND required 6 clock cycles between operations, which meant that when employing 6 groups of 8 MPC AND instances (48 total), it was possible for 8 transactions to begin each cycle. Even with this imperfect arrangement, a quick calculation shows that 8 of our AND operations (128 bits wide) per cycle at 200 MHz are sufficient to saturate a ∼205 Gbps connection, far more than the 10 Gbps link tested by Araki et al. with their software implementation of the protocol. While only a single party with one core was actually implemented and tested on the development board, the resource results and our calculations encouraged us to continue this work.

4.1.2 Initial Test with AWS

With these synthesis results from Quartus, but also seeking to avoid some of the challenges of the Nios II and to find a more fitting cloud target, we looked to Amazon Web Services (AWS). On AWS, our evaluation of the implemented MPC SS modules considers total FPGA resource utilization and total throughput with all gates running in parallel. Our implementation on this platform still requires an initial 21 cycles of pre-computation to prepare party keys and the first correlated random values. Repeating the same configuration as on the Arria 10, for a single 1-party block post-routing, the Virtex Ultrascale+ in the F1 instance utilizes ∼3.20% of its resources. It should be noted that while the Intel Quartus implementation only appeared to use ∼1.32%, that was a post-synthesis value rather than post-routing as prepared in Xilinx Vivado. Subsequently, for verification of multi-party operation, a 3×party

Table 4.1: AWS Implementation Result Analysis

AND Cores	Bits	Gbps	AES (millions op.)/sec
1	128	2.67	0.490
3	384	8.00	1.47
12	1536	32.0	5.89
24	3072	64.0	11.8
48	6144	128	23.5
60	7680	160	29.4

design with 1 AND per party was placed on the F1 instance FPGA. As mentioned earlier in the chapter, this was simpler to test than a three-FPGA design, avoided communication limitations, and provided useful information.

As each 1 party block contains more control logic than just a single MPC AND, estimates about the maximum amount of AND blocks that can fit will be conservative. Duplicating these groups of the 3×partis produced the results summarized in Table 4.1. As with the Arria, the number of AND modules and the 6-cycle delay were used to determine the number of bits per clock cycle that can be processed with the FPGA. The default AWS F1 clock rate of 125 MHz was used. Additionally, the equivalent number of AES/sec is reported for easier comparison with the results from Araki et al. and was estimated by dividing the number of AND/sec by the 5440 AND/AES in the reference AES circuit model as mentioned in Section 4.1. A plot of the FPGA utilization in Figure 4·1 shows a fairly linear relationship between number of AND modules and utilization. Both the initial Arria and AWS results were presented in the first publication covering this research (Wolfe et al., 2020).

4.1.3 Secondary Work with AWS

While the initial work demonstrated the potential of SS on FPGA hardware, the implementation only addressed Boolean shares and not arithmetic. The next research step focused on supporting arithmetic shares and verifying that they could be implemented with the same performance as the existing Boolean ones (Patel et al., 2020).

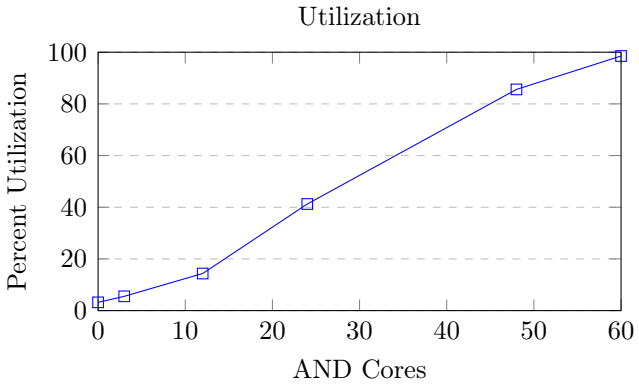


Figure 4·1: AWS FPGA fabric total utilization

In summary, with modification, the AND and XOR primitives were able to selectively handle multiplication and addition operations with essentially the same resource consumption and performance as the Boolean only implementation shown in Table 4.1 and Figure 4·1, that is to say, with testing on the same AWS F1 instance clocked at 125 MHz with the AND/multiplication gate taking a new input every 6 cycles and the XOR/addition gate accepting one on every cycle.

Some additional work was able to reduce the AND/multiplication gate from 6 cycles to 4 cycles. This design was able to operate at 125 MHz but couldn't be pushed much further. As the eventual goal was to create a fully-pipelined primitive that could accept a new input on each clock cycle, this wasn't tested further, and more effort went into reworking the design to achieve this goal. This fully-pipelined system has since been implemented and is currently being tested with the higher-level goal of implementing a MAC operation that could be useful for tasks like ML.

4.2 Reference Results

Briefly, we consider the secure operation metric used by the original authors (Araki, Barak, Furukawa, Lindell, et al., 2016; Araki, Furukawa, et al., 2016) when presenting their test results. The use the software implementation of their Secret Sharing

Table 4.2: Araki et al. Result Analysis

Araki et al. Results		Verification		
Cores	AES/sec	Gbps/serv.	Gbps/serv. w/over.	Error
1	100103 ± 1632	0.572	0.559	2.19%
5	530408 ± 7219	2.99	2.96	0.85%
10	975237 ± 3049	5.47	5.45	0.35%
16	1242310 ± 4154	6.95	6.94	0.10%
20	1324117 ± 3721	7.38	7.40	0.28%

protocol to perform MPC protected AES operations (Araki et al., 2017). To be clear, the authors used the logic of the AES-128 operation as mapped to a Boolean circuit representation in Bristol Fashion Key Expanded AES (D. Archer et al., n.d.), which requires 5440 secure AND operations. This AES is the problem of interest and is distinct from the PRF that is used as part of the secure MPC primitive implemented in HDL (AES-128 is used for the hardware primitive because of the good performance it offers).

The test described by Araki et al. is embarrassingly parallel, simultaneously running 12800 independent secure AES computations per core in each node. The total AES operations Araki et al. performed can be used to verify the number of bits communicated. Runtime is obtained from the AES/sec rate and total number of AES operations reported. When we include a reasonable overhead for TCP/IP communication between the nodes of 2.74% (Iveson, 2013), the verified network rates we calculate closely match the reported results with less than 2.5% error (Table 4.2). Having confirmed the operations they used while testing, our FPGA design can be more closely compared to the original software system by dividing the number of AND operations possible by the 5440 AND/AES conversion factor.

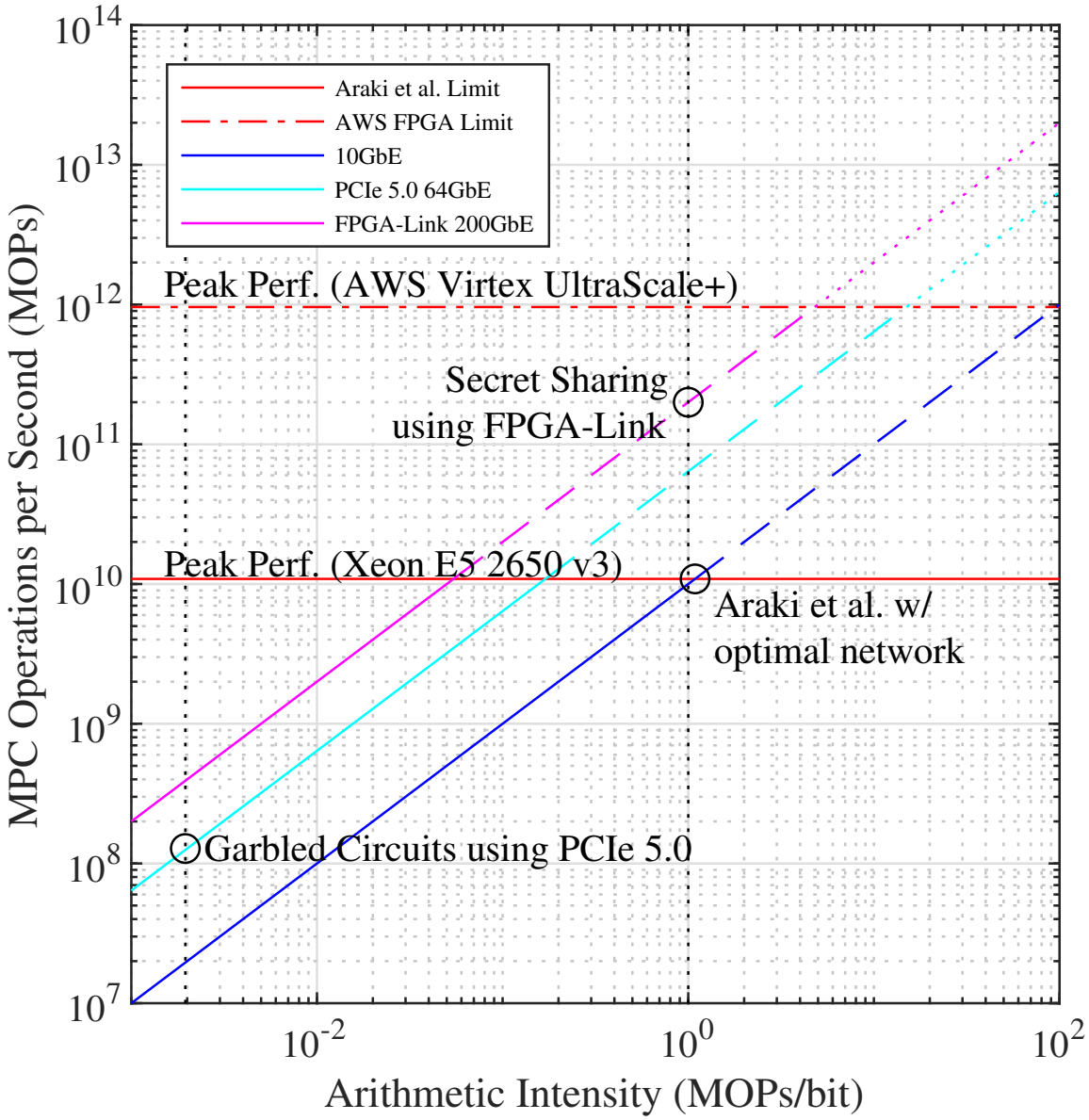


Figure 4·2: Roofline model comparing Secret Sharing performance against data center bandwidth. We denote limitations in the Araki et al. design and our FPGA Secret Sharing implementation, and FPGA implementations based on Garbled Circuits.

4.3 Analysis

The original authors (Araki, Barak, Furukawa, Lindell, et al., 2016; Araki, Furukawa, et al., 2016) tested their SS protocol implemented in software and deployed across three nodes with general purpose processors. In their configuration, each node had two Xeon E5-2686 v4 processors, each offering a total of 20 cores per party. The systems were linked in a ring with 10 Gbps network connections. Nearly saturating the connection (7.38 Gbps of 10 Gbps) required roughly 50% of each node's processor time. As noted in their analysis, one of the main limitations preventing full network saturation was due to the multiple cores causing queuing congestion at the shared NIC. If we imagine that this queuing issue were solved, we can try to estimate what the theoretical best output the processors are capable of. Using their reported number of MPC AES/sec and network communication for a single core, we can scale from the 73.3% CPU usage to 100% and estimate that 1 core might be approximately capable of about ∼130 thousand MPC AES/sec. If this were achieved, one core could saturate a ∼0.780 Gbps network connection. Multiplying for 20 cores on each node, we can find an estimated ideal maximum of ∼2.7 million MPC AES/sec resulting in traffic of about ∼15.6 Gbps. This is also slightly more generous than the approximation obtained by multiplying 7.38 Gbps obtained at ∼50% to get 14.76 Gbps.

If we consider the test results described in Section 4.1.2 and Section 4.1.3, we can see that, even with a 6-cycle delay, the AND/multiply block as tested on AWS F1 only required 3 AND cores per party to exceed the 7.38 Gbps reported by the protocol authors in their test. Three MPC AND modules per party also only consume ∼5% of the fabric available on each party's FPGA, a 10× improvement vs the 50% CPU utilization reported. This extends further if we consider attempting to fully employ all of the FPGA fabric. If we do so, each board (and party) could implement a maximum of 60 AND cores, which could saturate 160 Gbps links while performing 29.4 million

MPC AES/sec. While impressive, it is also important to recall that, as described in Section 3.4.2, an optimized, fully-pipelined MPC AND/multiply gate (processing 128 bits at a time as ours does) can saturate 10 Gbps at just 78.13 MHz or greater. There still remains more performance to be coaxed from the hardware design. With a fully-pipelined module operating at the same 125 MHz frequency used for the current tests, 200 Gbps links between parties could be saturated while consuming less than 24% of the FPGA fabric. With a slightly higher clock speed, this could be lowered further. All of the remaining fabric leaves room for additional functions, perhaps tying MPC primitives together in hardware.

To help visualize the peak performance that can be obtained with different platforms and networks when using MPC, we created a roofline plot Figure 4·2. Rather than continuing to use the MPC AES/sec unit that was necessary to compare to the original work, the roofline's Y-axis is measured in number of MPC operations per second (MOPs). The X-axis, arithmetic intensity (AI), is designated as MOPs/bit of data transferred over the network. We only use the AND/multiply gate for this plot as XOR/addition doesn't require communication. Furthermore, we assume a fully-pipelined design where one MOP per cycle is processed. Conveniently, as each SS gate requires one bit of communication (when correlated random numbers are generated locally to each party), an AI of 1 is possible.

With this in mind we can consider the horizontal lines. Looking at the Xeon, we can see that the Araki et al. design as tested on the CPU nodes should be able to reach a theoretical maximum of 10^{10} MOPs as long as there is no network bound (or overhead). If we instead consider an FPGA fully populated with MPC AND/multiply gates, we can see that, when unbounded, it should be possible to outperform the CPU by $\sim$2 orders of magnitude.

Having considered these compute capabilities, we can see that the Araki et al. test

system should provide an almost perfect balance between network and MOPs when utilizing their 10 Gbps interface (again, assuming that overhead can be avoided or eliminated). However, if a larger bandwidth network is available, then their application will be limited by computational performance and thus will not be able to fully utilize the bandwidth available. Switching our focus back to the FPGA, we can see that at an AI of 1, the 10 GbE, 64 GbE, and even 200 GbE network connections are still not sufficient to keep up with the compute available. Consequently, with those network speeds, the theoretical peak of 10^{12} MOPs can't be reached.

This isn't a cause for alarm however as there are many faster links arriving to the data center and to FPGA hardware. Additionally, there are likely to be other functions of interest that would be placed on an FPGA used for MPC rather than purely AND/multiplication gates. It is quite possible to imagine a case where enough of an FPGA is used to make full use of direct network links for MPC while other functions deployed would communicate with the host nodes via PCIe.

To provide context with respect to GC MPC, we imagine that each garbling table can be created in a single clock cycle. In a traditional GC MPC gate a table contains $4\times$ hashes (SHA-1 is commonly used though not all the bits are necessarily utilized). For our roofline, we consider the AI of Garbled Circuits to be $1/(4 \times 128) = 0.002$, which, when plotted, shows that Garbled Circuits are heavily network limited due to the low arithmetic intensity and high bandwidth requirements. In summary, there are preferable scaling properties when using SS as opposed to GC in an ideal environment.

Between the results of our FPGA tests and of our analysis, we find that SS MPC is competitive with GC approaches. There is also strong support for the benefits of deploying SS MPC on FPGAs when low-latency, high-bandwidth communication is available, such as in the data center.

Chapter 5

Cloud Deployment & Platform Analysis

5.1 Cloud Computing Course

This chapter takes a step away from the low-level implementation details. Instead, observations derive primarily from research conducted as part of the project-based Cloud Computing course taught by Prof. Oran Krieger and Prof. Ata Turk. While efforts described previously focused on exploring the types of MPC protocol and the scenarios in which they could best benefit from different types of hardware acceleration, the system level design and mechanism for deployments are equally important to eventually creating a complete MPCaaS system. The mentors for the project group, Prof. John Liagouris and Prof. Vasiliki Kalavri, had recently cleanly implemented an MPC framework they called *Secrecy* (Liagouris et al., 2021). By creating a fresh implementation, they were able to keep the number of outside dependencies to a minimum, and the underlying data transactions could be grouped into tables of values, both of which help boost performance. The only dependencies included using Libsodium for random value generation and MPI (specifically OpenMPI) for communication between parties.

The goals for the student group were to prepare tools to more easily deploy the code-base into different cloud environments on the MOC and on CloudLab and to make improve the ability to run tests and collect data. With the ability to test in various environments, the focus on collected data was in examining the primary bottleneck of the selected MPC protocol in the time available. With *Secrecy* making

use of a 3PC SS protocol, the inter-party MPC communication primitive was the focus of most initial testing.

While the main objective of automation was to improve the ability to rapidly test in different environments during the course, the tools explored and developed were also meant to be used in ongoing work. Furthermore, automation would help to better understand the nuances of configuring and deploying MPC in different fashions in a data center, something which is critical to understand when developing production MPCaaS.

The three main environments considered consisted of VMs deployed using OpenStack on the MOC, bare-metal nodes provisioned on CloudLab (Duplyakin et al., 2019), and containers deployed using OpenShift on the MOC. The MOC is hosted at the MGHPCC in Massachusetts while CloudLab consists of resources across several data centers with some primary ones located in Utah, Wisconsin, and South Carolina. The MPC communication primitive in *Secrecy* was tested using several different approaches with MPI, specifically both blocking and non-blocking (sync and async) as well as batched data in various quantities and serialized element-by-element transactions. Instrumentation of tests for the communication primitive was improved in several fashions. Instead of relying on the original "gettimeofday()" call, the new instrumentation employs "clock_gettime()" using the "CLOCK_MONOTONIC" source (absolute elapsed wall-clock time from an arbitrary point in time). This change makes it possible to more accurately measure each portion of the operation. Additionally, the test was parameterized to allow for automatic iteration over a range of values with a specific step size. The results were all placed in a single CSV file for convenient analysis. Finally, some experimentation with the Score-P utilities resulted in initial documentation on some simple benchmarking and profiling that can be enabled when compiling with OpenMPI. Some simple tests produced reasonable results, and this

offers another avenue for students and researchers who are working on this in the future. It may particularly come in useful for collecting lower-level information about inter-party communication.

5.2 Platform Comparison

The bare-metal systems tested through CloudLab were provisioned by means of a web interface for Emulabs (White et al., 2003). Python and geni-lib (from GENI (Berman et al., 2014)) were used to describe the bare-metal system nodes and links desired for various tests. These descriptions automatically generate RSpec files used by the system to provision the hardware for the experiment as the RSpec files themselves are more challenging to edit by hand. Using this approach, existing OS images could also be pre-selected and loaded onto each of the systems, leaving only final package configuration before testing.

To prepare the VMs on the MOC OpenStack instance, the web interface was used for selecting an OS image, resources, and defining the network connections. No configuration scripts or files were needed to reach the point of a loaded OS on each VM, leaving final package installation and configuration.

On both CloudLab and MOC OpenStack, the first few tests were prepared by means of manual package installation and commands. This was replaced by a series of shell scripts. Eventually, in order to better automate all of those steps and achieve a consistent result, Ansible playbooks were developed to ensure full system preparation of running tests. Playbooks were also used to initiate the MPC transaction tests and retrieve data for examination.

Ansible is an agentless tool that only requires the configuring/managing system and client systems to have Python installed. Connections between managing and client systems are temporary and typically conducted over SSH. With few dependen-

cies, this approach is lightweight and is consequently relatively easy to use. When carefully crafted, the automation actions described in a playbook make it possible to reach a consistent system state. Not only does Ansible make it possible to write playbooks that reach specific states, but it is also possible to craft them so that a playbook is idempotent. Then, a playbook can be utilized to verify that the desired state is reached as multiple executions will not cause any already correct state to be modified. The containers deployed on MOC OpenShift are also possible to automate using Ansible but differed sufficiently from the bare-metal systems and VMs that extended testing and development of a playbook for them was not possible during the group project timeline.

5.2.1 Bare Metal, Virtual Machines, Containers

Ideally, the team sought to deploy bare-metal, VMs, and containers onto CloudLab in order to make a direct comparison of the relative overhead cost of the approaches when run with the same underlying resources. This was overly ambitious given the limited time but remains a direction for ongoing research. Instead, some preliminary results were captured when running the same tests across the MOC and CloudLab. The results that the project mentors found when testing on the MOC (Liagouris et al., 2021) were closely matched by the VMs configured by the student team, which provided some assurance that there were not any major unexpected changes or variations. Essentially, batching inter-party data and allowing OpenMPI to determine if and how to fragment that into multiple packets produced a faster transaction time than serializing the operations into a sequence of individual MPI transactions (forcing small, immediate packets).

Additionally, the tests performed on the bare-metal nodes on CloudLab were faster than the same tests conducted on VMs on the MOC OpenStack. This result isn't surprising given some amount of additional overhead expected for the VMs, but with-

out a test on the same hardware, it isn't possible to indicate exactly how much was lost as a result of the virtualization. Additionally, the bare-metal nodes using an x86 architecture had slightly better performance than the equivalent ARM nodes available, but it isn't possible to attribute this to either architecture and would require additional testing to properly examine.

While full testing was not possible, the automation efforts were eye opening and did provide some insight and ideas for what might work best when designing a complete MPCaaS.

5.2.2 Scalability & Automated Provisioning

Moving forward, a set of bare-metal nodes seems ideal for an initial deployment of MPC parties. Using such an approach makes it much easier to logically separate entities and makes it possible for each compute party to establish trusted control over the system they would use to evaluate an MPC problem. This does not address all risks; there must still be careful consideration of how to manage system firmware and provide the necessary integrity guarantees between users. There are, however, existing researchers working on this problem (Mosayyebzadeh et al., 2019).

Additionally, there are performance advantages in reducing the number of layers of abstraction layered on top of hardware. While this does mean that a server could not be shared between multiple different compute parties, this may not be too much of a draw-back. If FPGAs are used for accelerated MPC, then the host system is critical for each of the compute parties to configure and operate a connected FPGA in a trusted fashion. While there would be less of a demand for the host processor to perform MPC, there are still a number of essential steps that require it including to coordinate with users to ingest data shares, coordinate execution of the desired MPC operations, distribute the final result, and perform eventual system cleanup. Careful sizing of the processor may help to complete all of these tasks while cutting idle time.

Beyond the tools available through CloudLab, there are a number of other ways (Ellis, 2020) to provision bare-metal clouds including OpenStack Ironic. It is possible to envision one of these bare-metal systems being used by a data center to provision hardware for several different compute parties with one host processor and one FPGA each. A common set of Ansible or other setup scripts can enable each party to inspect the operations on their own to establish trust and then to execute them on the hardware they have been provisioned. This allows them to reach the same configuration state as their peers. Using such an approach will also make deployment and setup more natural for system administrators.

5.3 Key Ideas

While a cloud deployment was already envisioned while working on the low-level hardware implementation, the vision lacked some clarity prior to the hands-on work experience from the cloud project. Having worked with multiple different deployment environments has made bare-metal nodes appear a better choice for MPCaaS, at least initially. In addition to offering at least some additional performance by avoiding the overhead of virtualization, the tools and infrastructure for provisioning and configuring such systems were relatively easy to use. It was also relatively easy to configure access for different users for each of the nodes and to imagine how one might isolate the nodes and hand off control to independent parties. Working with Ansible also showed that it and similar tools could provide an easy mechanism for having multiple different MPC parties operate the same protocol while still having the freedom to inspect all the configuration steps on their own based on their level of trust. It is also encouraging to have had several conversations with students who are continuing the work on *Secrecy* and are at the very least referring to the tools and documentation from this work and potentially modifying and using them.

Chapter 6

Remaining Work & Conclusion

6.1 Steps to MPCaaS

At this point we have considered the goals of MPC and the different types of protocols that exist under that name. The history of technological advances highlights the vast improvement and specialization in processing architectures, increases in communication performance, and greater capacity of storage. The opportunity and ability to collect, process, and benefit from "big data" both resulted from, and further spurred, some of these innovations. In particular, the clustering of compute into larger data centers and the cloud model of providing access to different resources goes hand-in-hand with "big data."

With all of these changes have come greater challenges to security and privacy. While the problem has been recognized, it remains challenging to handle adequately. Luckily, at this point MPC is feasible to implement and promises to be another innovation that could help to address the issue. By making use of the algorithmic, architectural, and operational opportunities in computing, MPCaaS can become an easy-to-use reality for many users who are not experts in cryptography or computer hardware. We have a plan consisting of three steps, described in the following subsections, that can be followed to achieve this vision.

6.1.1 MPC in the Cloud

In order to enable MPC in the cloud, we need to select which kind of protocol to use. Performing well across geographically separated parties is an important consideration, especially with high-latency connections, but is not our main focus. As shown earlier, we seek to support cases where hardware is co-located and can be assigned to different users in order to support multiple computations one after the other. Determining what protocol best fits this scenario is crucial, and our work leads us to believe that SS fits this model best. Also critical is choosing an appropriate cloud deployment model that can provide the necessary security guarantees. While actually getting a working MPC system into the cloud does not require the greatest performance initially, keeping acceleration in mind when planning is important as a production system will need to minimize the gap between MPC and non-MPC services.

Bare metal clouds may provide the system isolation needed between parties. Users will likely need some mechanism to gain a trusted foothold in a shared data center. Incorporating existing work, e.g., on elements such as Hardware Isolation Layer (HIL) (Mosayyebzadeh et al., 2019) and isolated networks through VLANs, will be crucial to deliver a system that does not require a user to handle all those considerations on their own. Offering the deployment and management tools and selection of appropriate MPC protocol would help system architects and administrators adopt such a system. An administrative portal for easy configuration would be ideal.

6.1.2 Transparent Hardware

The next stage of work focuses on the interface between an MPC software library and the FPGA hardware implementation of MPC primitives. The end goal is for users, such as data scientists, with sufficient MPC knowledge to use the API offered by an MPC library to benefit from hardware accelerations without needing to also

be knowledgeable about hardware design or HDLs. In order to deliver this vision, an MPC library that offers both sufficient features and a relatively simple interface needs to be identified. Alternately, a new library could be developed. In the interest of being able to offer some working system, it may also make sense to reduce the scope of supported applications to focus more development effort on a limited set of operations to map to hardware. One idea is to concentrate on ML applications rather than being fully general. There has already been research seeking to support ML on MPC (Du & Atallah, 2001; Knott et al., 2020; Ohrimenko et al., 2016; Zhao et al., 2019). One good parallel for this vision is the manner in which PyTorch (Paszke et al., 2019) and other tools are able to make the use of GPUs transparent to their users while still accelerating training and inference. Researchers at Facebook also appear to be moving in this direction as evidenced by the work on CrypTen, which seeks to provide a software tool that enables PyTorch-style use of MPC with ML (Knott et al., 2020).

6.1.3 Transparent MPC Functions & Program Conversion

The final stage is at the highest notional level. Here the goal is to take a working MPC platform that employs hardware acceleration and make it usable even to those who do not have experience with MPC. This tool should offer a mechanism to convert programs into an MPC format automatically, which can then be mapped and deployed on cloud-based FPGA hardware. If the choice is made to support a limited set of applications, such as ML, then it might be possible to select an input format, such as a PyTorch model, familiar to the end-user. This would limit the types of MPC interactions that need to be supported and would make it possible to map key communication stages from a clear protocol to an encrypted one. As ML use has been growing, providing the ability to train and infer under MPC could open up more pools of data for use by scientists.

6.2 Current & Future Work

Ongoing work has resulted in improving the AND/multiplication hardware implementation such that it is fully pipelined. This has included doubling the number of internal AES cores that are used as PRF increasing resource use. The ability to start a new operation on each cycle is worth this extra cost. Furthermore, there remains a number of additional optimizations that can be used to reduce the size of this block. The most interesting use of this fully pipelined operation is in seeking to enable the efficient execution of MM and MAC operations, which are critical for ML and other applications. We plan to perform additional testing with the improved MPC primitives and refine them. Additionally, ever since work started on implementing the Araki et al. protocol, we have had the goal of eventually supporting a maliciously secure version of the protocol. While simply using the extension of the semi-honest protocol (Araki et al., 2017) originally seemed like the best course of action, several four-party maliciously secure protocols have been discussed in recent literature and may offer more efficient approaches to achieve this security objective.

We are also currently working with a few testbeds that offer multiple, directly-linked FPGA resources. As AWS F1 does not appear likely to support serial ring communication in the near future, we hope some of these other efforts will allow us to test a design that is closer to a production MPCaaS. Depending on the rate of progress, there may also be a chance to start considering some of the automation techniques used when testing *Secrecy*.

More broadly, we aim to follow the steps described for achieving MPCaaS. This may include further work on the MOC if the hardware versions for MPC on independent nodes with FPGAs do not progress rapidly enough. Then, those lessons learned and hardware implementation (potentially deployed to the OCT) could directly lead to the second step. The final stage could likely be explored at a high level, perhaps

looking at conversions of ML to MPC problem descriptions. CrypTen is a potentially useful project from which we can draw inspiration.

6.3 Conclusions

In conclusion, in this thesis we describe, implement, and analyze the MPC protocol described by Araki et al. (Araki et al., 2017) in hardware. We demonstrate the viability of Secret Sharing MPC in a low latency environment and test the design on an FPGA in the cloud, highlighting greater potential scalability of the design compared to alternatives (both algorithmic and architectural). With these insights, we are actively improving our design to increase the performance further while targeting the steps we have laid out for achieving a complete MPC cloud service. This is an exciting time for MPC techniques, hardware acceleration, and the cloud. Hopefully this work can encourage further efforts accelerating MPC and can lead to some new widespread and easy-to-use security and privacy tools.

References

Abidin, A., Aly, A., Cleemput, S., & Mustafa, M. A. (2016). An MPC-Based Privacy-Preserving Protocol for a Local Electricity Trading Market. *15th International Conference on Cryptology and Network Security (CANS)*, *10052*, 615–625. https://doi.org/10.1007/978-3-319-48965-0_40

Agrawal, R., Bu, L., Ehret, A., & Kinsy, M. (2019). Open-source FPGA implementation of post-quantum cryptographic hardware primitives. *2019 29th International Conference on Field Programmable Logic and Applications (FPL)*, 211–217. https://doi.org/10.1109/FPL.2019.00040

Anant, V., Donchak, L., Kaplan, J., & Soller, H. (2020). *The consumer-data opportunity and the privacy imperative*. https://www.mckinsey.com/business-functions/risk/our-insights/the-consumer-data-opportunity-and-the-privacy-imperative

Araki, T., Barak, A., Furukawa, J., Lindell, Y., Nof, A., & Ohara, K. (2016). DEMO: High-Throughput Secure Three-Party Computation of Kerberos Ticket Generation. *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, 1841–1843. https://doi.org/10.1145/2976749.2989035

Araki, T., Barak, A., Furukawa, J., Lichter, T., Lindell, Y., Nof, A., Ohara, K., Watzman, A., & Weinstein, O. (2017). Optimized Honest-Majority MPC for Malicious Adversaries - Breaking the 1 Billion-Gate per Second Barrier. *Proceedings - IEEE Symposium on Security and Privacy*, 843–862. https://doi.org/10.1109/SP.2017.15

Araki, T., Furukawa, J., Lindell, Y., Nof, A., & Ohara, K. (2016). High-Throughput Semi-Honest Secure Three-Party Computation with an Honest Majority. *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, 805–817. https://doi.org/10.1145/2976749.2978331

Archer, D., Abril, V. A., Lu, S., Maene, P., Mertens, N., Sijacic, D., & Nigel, S. (n.d.). *'Bristol Fashion' MPC Circuits*. https://homes.esat.kuleuven.be/~nsmart/MPC/

Archer, D. W., Bogdanov, D., Lindell, Y., Kamm, L., Nielsen, K., Pagter, J. I., Smart, N. P., & Wright, R. N. (2018). From Keys to Databases - Real-World Applications of Secure Multi-Party Computation. *The Computer Journal*, *61*(12), 1749–1771. https://doi.org/10.1093/comjnl/bxy090

Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., & Warfield, A. (2003). Xen and the Art of Virtualization. *SIGOPS Operating Systems Review*, *37*(5), 164–177. https://doi.org/10.1145/1165389.945462

Beaver, D., Micali, S., & Rogaway, P. (1990). The round complexity of secure protocols. *Proceedings of the Twenty-Second Annual ACM Symposium on Theory of Computing*, 503–513. https://doi.org/10.1145/100216.100287

Belanović, P., & Leeser, M. (2002). A library of parameterized floating-point modules and their use. In M. Glesner, P. Zipf, & M. Renovell (Eds.), *Field-programmable logic and applications* (pp. 657–666). Springer Berlin Heidelberg. https://doi.org/10.1007/3-540-46117-5_68

Ben-Or, M., Goldwasser, S., & Wigderson, A. (1988). Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation. *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, 1–10. https://doi.org/10.1145/62212.62213

Berman, M., Chase, J. S., Landweber, L., Nakao, A., Ott, M., Raychaudhuri, D., Ricci, R., & Seskar, I. (2014). GENI: A federated testbed for innovative network experiments. *Computer Networks*, *61*, 5–23. https://doi.org/10.1016/j.bjp.2013.12.037

Bogdanov, D., Kamm, L., Kubo, B., Rebane, R., Sokk, V., & Talviste, R. (2016). Students and taxes: A privacy-preserving study using secure computation. *Proceedings on Privacy Enhancing Technology (PoPETs)*, *2016*(3), 117–135. https://doi.org/10.1515/popets-2016-0019

Bogetoft, P., Christensen, D. L., Damgård, I., Geisler, M., Jakobsen, T., Krøigaard, M., Nielsen, J. D., Nielsen, J. B., Nielsen, K., Pagter, J., Schwartzbach, M., & Toft, T. (2009). Secure multiparty computation goes live. In R. Dingledine & P. Golle (Eds.), *Financial cryptography and data security* (pp. 325–343). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-03549-4_20

Boku, T., Kobayashi, R., Fujita, N., Amano, H., Sano, K., Hanawa, T., & Yamaguchi, Y. (2019). Cygnus: GPU meets FPGA for HPC. *International Conference on Supercomputing*. https://www.r-ccs.riken.jp/labs/lpnctrt/assets/img/lspanc2020jan_boku_light.pdf

Bolaria, J., & Byrne, J. (2009). *A Guide to FPGAs for Communications*. The Linley Group.

Bonawitz, K., Ivanov, V., Kreuter, B., Marcedone, A., McMahan, H. B., Patel, S., Ramage, D., Segal, A., & Seth, K. (2017). Practical secure aggregation for privacy-preserving machine learning. *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 1175–1191. https://doi.org/10.1145/3133956.3133982

Brewer, E. (2012). CAP twelve years later: How the "rules" have changed. *Computer*, *45*(2), 23–29. https://doi.org/10.1109/MC.2012.37

Brooks, C. (2021, March 2). *Alarming Cybersecurity Stats: What You Need To Know For 2021.* https://www.forbes.com/sites/chuckbrooks/2021/03/02/alarming-cybersecurity-stats-------what-you-need-to-know-for-2021/?sh=35cd237858d3

Burns, B., Grant, B., Oppenheimer, D., Brewer, E., & Wilkes, J. (2016). Borg, Omega, and Kubernetes: Lessons learned from three container-management systems over a decade. *Queue*, *14*(1), 70–93. https://doi.org/10.1145/2898442.2898444

Castillo, J. (2004). systemc_rng. https://opencores.org/projects/systemc_rng

Caulfield, A. M., Chung, E. S., Putnam, A., Angepat, H., Fowers, J., Haselman, M., Heil, S., Humphrey, M., Kaur, P., Kim, J. .-Y., Lo, D., Massengill, T., Ovtcharov, K., Papamichael, M., Woods, L., Lanka, S., Chiou, D., & Burger, D. (2016). A cloud-scale acceleration architecture. *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 1–13. https://doi.org/10.1109/MICRO.2016.7783710

Chang, F., Dean, J., Ghemawat, S., Hsieh, W. C., Wallach, D. A., Burrows, M., Chandra, T., Fikes, A., & Gruber, R. E. (2006). Bigtable: A distributed storage system for structured data. *7th USENIX Symposium on Operating Systems Design and Implementation (OSDI 06)*, 205–218. https://www.usenix.org/conference/osdi-06/bigtable-distributed-storage-system-structured-data

Chang, W., Roy, A., & Underwood, M. (2019, October 21). *NIST Big Data Interoperability Framework: Volume 4, Security and Privacy.* https://doi.org/10.6028/NIST.SP.1500-4r2

Chaum, D., Crépeau, C., & Damgard, I. (1988). Multiparty Unconditionally Secure Protocols. *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, 11–19. https://doi.org/10.1145/62212.62214

Cherry, S. (2004). Edholm's law of bandwidth. *IEEE Spectrum*, *41*(7), 58–60. https://doi.org/10.1109/MSPEC.2004.1309810

Clarke, P. (2015). *Intel, Micron Launch "Bulk-Switching" ReRAM.* https://www.eetimes.com/intel-micron-launch-bulk-switching-reram/

Corbett, J. C., Dean, J., Epstein, M., Fikes, A., Frost, C., Furman, J., Ghemawat, S., Gubarev, A., Heiser, C., Hochschild, P., Hsieh, W., Kanthak, S., Kogan, E., Li, H., Lloyd, A., Melnik, S., Mwaura, D., Nagle, D., Quinlan, S., ... Wang, R. (2012). Spanner: Google's Globally-Distributed Database. *10th USENIX Symposium on Operating Systems Design and Implementation (OSDI 12)*, 261–264. https://www.usenix.org/conference/osdi12/technical-sessions/presentation/corbett

Dalskov, A., Escudero, D., & Keller, M. (2020). *Fantastic Four: Honest-Majority Four-Party Secure Computation With Malicious Security*. https://eprint.iacr.org/2020/1330

Damgård, I., Damgård, K., Nielsen, K., Nordholt, P. S., & Toft, T. (2017). Confidential benchmarking based on multiparty computation. In J. Grossklags & B. Preneel (Eds.), *Financial cryptography and data security* (pp. 169–187). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-662-54970-4_10

de Dinechin, F., & Pasca, B. (2011). Designing custom arithmetic data paths with FloPoCo. *IEEE Design Test of Computers*, *28*(4), 18–27. https://doi.org/10.1109/MDT.2011.44

Dean, J., & Ghemawat, S. (2004). Mapreduce: Simplified data processing on large clusters. *6th Symposium on Operating Systems Design & Implementation (OSDI 04)*, 137–150. https://www.usenix.org/conference/osdi-04/mapreduce-simplified-data-processing-large-clusters

DeCandia, G., Hastorun, D., Jampani, M., Kakulapati, G., Lakshman, A., Pilchin, A., Sivasubramanian, S., Vosshall, P., & Vogels, W. (2007). Dynamo: Amazon's Highly Available Key-Value Store. *SIGOPS Operating Systems Review*, *41*(6), 205–220. https://doi.org/10.1145/1323293.1294281

Demchenko, Y., v. d. Ham, J., Ngo, C., Matselyukh, T., Filiposka, S., d. Laat, C., & Escalona, E. (2013). Open Cloud eXchange (OCX): Architecture and Functional Components. *2013 IEEE 5th International Conference on Cloud Computing Technology and Science*, *2*, 81–87. https://doi.org/10.1109/CloudCom.2013.108

Demmler, D., Schneider, T., & Zohner, M. (2015). ABY - A Framework for Efficient Mixed-Protocol Secure Two-Party Computation. *2015 Network and Distributed System Society (NDSS) Symposium. The Internet Society.* https://doi.org/10.14722/ndss.2015.23113

Dennard, R. H., Gaensslen, F. H., Yu, H., Rideout, V. L., Bassous, E., & LeBlanc, A. R. (1974). Design of ion-implanted MOSFET's with very small physical dimensions. *IEEE Journal of Solid-State Circuits*, *9*(5), 256–268. https://doi.org/10.1109/JSSC.1974.1050511

Desnoyers, P., Hennessey, J., Holden, B., Krieger, O., Rudolph, L., & Young, A. (2015). Using Open Stack for an Open Cloud Exchange(OCX). *2015 IEEE International Conference on Cloud Engineering*, 48–53. https://doi.org/10.1109/IC2E.2015.40

Du, W., & Atallah, M. J. (2001). Secure multi-party computation problems and their applications: A review and open problems. *Proceedings of the 2001 Workshop on New Security Paradigms*, 13–22. https://doi.org/10.1145/508171.508174

Duplyakin, D., Ricci, R., Maricq, A., Wong, G., Duerig, J., Eide, E., Stoller, L., Hibler, M., Johnson, D., Webb, K., Akella, A., Wang, K., Ricart, G., Landweber, L., Elliott, C., Zink, M., Cecchet, E., Kar, S., & Mishra, P. (2019). The Design and Operation of CloudLab. *Proceedings of the USENIX Annual Technical Conference (ATC)*, 1–14. https://www.flux.utah.edu/paper/duplyakin-atc19

Ellis, A. (2020). awesome-baremetal. https : / / github . com / alexellis / awesome - baremetal.git

2020 Roadmap: Ethernet Alliance. (2020). https://ethernetalliance.org/technology/2020-roadmap/

Evans, D., Kolesnikov, V., & Rosulek, M. (2018). *A pragmatic introduction to secure multi-party computation*. NOW Publishers. https://doi.org/10.1561/3300000019

Fang, X., Ioannidis, S., & Leeser, M. (2017). Secure function evaluation using an FPGA overlay architecture. *FPGA 2017 - Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 257–266. https://doi.org/10.1145/3020078.3021746

Fang, X., Ioannidis, S., & Leeser, M. (2019). SIFO: Secure computational infrastructure using FPGA overlays. *International Journal of Reconfigurable Computing, 2019*. https://doi.org/10.1155/2019/1439763

Feigenbaum, J., Pinkas, B., Ryger, R., & Saint-Jean, F. (2004). Secure computation of surveys. *EU Workshop on Secure Multiparty Protocols*, 2–14. https://www.cs.yale.edu/homes/jf/SMP2004.pdf

Frederiksen, T. K., Jakobsen, T. P., & Nielsen, J. B. (2014). Faster maliciously secure two-party computation using the GPU. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 8642*(grant 61061130540), 358–379. https://doi.org/10.1007/978-3-319-10879-7_21

Furukawa, J., & Lindell, Y. (2019). Two-thirds honest-majority MPC for malicious adversaries at almost the cost of semi-honest. *Proceedings of the ACM Conference on Computer and Communications Security*, 1557–1571. https://doi.org/10.1145/3319535.3339811

Furukawa, J., Lindell, Y., Nof, A., & Weinstein, O. (2017). High-throughput secure three-party computation for malicious adversaries and an honest majority. In J.-S. Coron & J. B. Nielsen (Eds.), *Advances in cryptology – eurocrypt 2017* (pp. 225–255). Springer International Publishing. https://doi.org/10.1007/978-3-319-56614-6_8

Geng, T., Li, A., Shi, R., Wu, C., Wang, T., Li, Y., Haghi, P., Tumeo, A., Che, S., Reinhardt, S., & Herbordt, M. (2020). AWB-GCN: A Graph Convolutional Network

Accelerator with Runtime Workload Rebalancing. *53rd IEEE/ACM International Symposium on Microarchitecture (MICRO)*. https://arxiv.org/pdf/1908.10834.pdf

Geng, T., Wang, T., Sanaullah, A., Yang, C., Xuy, R., Patel, R., & Herbordt, M. (2018). A framework for acceleration of CNN training on deeply-pipelined FPGA clusters with work and weight load balancing. *2018 28th International Conference on Field Programmable Logic and Applications (FPL 2018): 394–402.* https://doi.org/10.1109/FPL.2018.00074

Geng, T., Wang, T., Wu, C., Li, Y., Yang, C., Wu, W., Li, A., & Herbordt, M. (2021). O3BNN-R: An Out-Of-Order Architecture for High-Performance and Regularized BNN Inference. *TPDS*, *32*(1), 199–213. https://doi.org/10.1109/TPDS.2020.3013637

Geng, T., Wu, C., Tan, C., Fang, B., Li, A., & Herbordt, M. (2020). CQNN: a CGRA-based QNN Framework. *HPExC*. https://doi.org/10.1109/HPEC43674.2020.9286194

George, A. D., Herbordt, M. C., Lam, H., Lawande, A. G., Sheng, J., & Yang, C. (2016). Novo-G#: large-scale reconfigurable computing with direct and programmable interconnects. *2016 IEEE High Performance Extreme Computing Conference (HPEC)*, 1–7. https://doi.org/10.1109/HPEC.2016.7761639

Ghemawat, S., Gobioff, H., & Leung, S.-T. (2003). The Google File System. *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*, 29–43. https://doi.org/10.1145/945445.945450

Giannopoulos, T., & Mouris, D. (2018). *Privacy Preserving Medical Data Analytics using Secure Multi Party Computation. An End-To-End Use Case.* (Doctoral dissertation). National and Kapodistrian University of Athens. https://doi.org/10.13140/RG.2.2.19303.70562

Gokhale, M. B., & Graham, P. S. (2005). *Reconfigurable computing: Accelerating computation with field-programmable gate arrays.* Springer. https://doi.org/10.1007/b136834

Gonzalez, J. E., Low, Y., Gu, H., Bickson, D., & Guestrin, C. (2012). PowerGraph: Distributed Graph-Parallel Computation on Natural Graphs. *10th USENIX Symposium on Operating Systems Design and Implementation (OSDI 12)*, 17–30. https://www.usenix.org/conference/osdi12/technical-sessions/presentation/gonzalez

Gordon, S. D., Ranellucci, S., & Wang, X. (2018). Secure computation with low communication from cross-checking. In T. Peyrin & S. Galbraith (Eds.), *Advances in cryptology – ASIACRYPT 2018* (pp. 59–85). Springer International Publishing. https://doi.org/10.1007/978-3-030-03332-3_3

Haghi, P., Geng, T., Guo, A., Wang, T., & Herbordt, M. (2020). A Reconfigurable Compute-in-the-Network FPGA Assistant for High-Level Collective Support with Distributed Matrix Multiply Case Study. *2020 International Conference on Field-Programmable Technology (FPT)*.

Haghi, P., Guo, A., Xiong, Q., Patel, R., Yang, C., Geng, T., Broaddus, J. T., Marshall, R., Skjellum, A., & Herbordt, M. C. (2020). FPGAs in the network and novel communicator support accelerate MPI collectives. *2020 IEEE High Performance Extreme Computing Conference (HPEC)*, 1–10. https://doi.org/10.1109/HPEC43674.2020.9286200

Hajduczenia, M., Carlson, S. B., Dove, D., Laubach, M., Law, D., & Zimmerman, G. A. (2016). *Evolution of Ethernet Standards in IEEE 802.3 Working Group*. https://www.standardsuniversity.org/e-magazine/august-2016-volume-6/evolution-ethernet-standards-ieee-802-3-working-group/

Harris, M. (2015). *A Brief History of General-Purpose Computation on GPUs*. https://cs.unc.edu/50th/celebration/symposium/

Hastings, M., Hemenway, B., Noble, D., & Zdancewic, S. (2019). SoK: General purpose compilers for secure multi-party computation. *Proceedings - IEEE Symposium on Security and Privacy*, *2019-May*, 1220–1237. https://doi.org/10.1109/SP.2019.00028

Hauck, S., & DeHon, A. (2008). *Reconfigurable computing: The theory and practice of fpga-based computing*. Morgan Kaufmann.

Herbordt, M. C., VanCourt, T., Gu, Y., Sukhwani, B., Conti, A., Model, J., & DiSabello, D. (2007). Achieving high performance with FPGA-based computing. *IEEE Computer*, *40*(3), 50–57. https://doi.org/10.1109/MC.2007.79

Herbordt, M., Gu, Y., VanCourt, T., Model, J., Sukhwani, B., & Chiu, M. (2008). Computing models for FPGA-based accelerators with case studies in molecular modeling. *Computing in Science and Engineering*, *10*(6), 35–45. https://doi.org/10.1109/MCSE.2008.143

Hindman, B., Konwinski, A., Zaharia, M., Ghodsi, A., Joseph, A. D., Katz, R., Shenker, S., & Stoica, I. (2011). Mesos: A Platform for Fine-Grained Resource Sharing in the Data Center. *8th USENIX Symposium on Networked Systems Design and Implementation (NSDI 11)*. https://www.usenix.org/conference/nsdi11/mesos-platform-fine-grained-resource-sharing-data-center

Hsing, H. (2012). tiny_aes. https://opencores.org/projects/tiny_aes

Huang, K., Gungor, M., Fang, X., Ioannidis, S., & Leeser, M. (2019). Garbled circuits in the cloud using FPGA enabled nodes. *2019 IEEE High Performance Extreme*

Computing Conference, HPEC 2019, 1–6. https://doi.org/10.1109/HPEC.2019.8916407

Huang, Y., Evans, D., & Katz, J. (2012). Private Set Intersection: Are Garbled Circuits Better than Custom Protocols? *In 19th Network and Distributed Security Symposium (NDSS 2012). The Internet Society.* https://www.ndss-symposium.org/ndss2012/ndss-2012-programme/private-set-intersection-are-garbled-circuits-better-custom-protocols/

Hussain, S. U., & Koushanfar, F. (2019). FASE: FPGA acceleration of secure function evaluation. *Proceedings - 27th IEEE International Symposium on Field-Programmable Custom Computing Machines, FCCM 2019*, 280–288. https://doi.org/10.1109/FCCM.2019.00045

Hussain, S. U., Rouhani, B. D., Ghasemzadeh, M., & Koushanfar, F. (2018). MAXelerator: FPGA Accelerator for Privacy Preserving Multiply-Accumulate (MAC) on Cloud Servers. *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*, 1–6. https://doi.org/10.1109/dac.2018.8465770

Husted, N., Myers, S., Shelat, A., & Grubbs, P. (2013). GPU and CPU parallelization of honest-but-curious secure two-party computation. *ACM International Conference Proceeding Series*, 169–178. https://doi.org/10.1145/2523649.2523681

Power 4 – The First Multi-Core, 1GHz Processor. (n.d.). https://www.ibm.com/ibm/history/ibm100/us/en/icons/power4/

Ion, M., Kreuter, B., Nergiz, A. E., Patel, S., Raykova, M., Saxena, S., Seth, K., Shanahan, D., & Yung, M. (2019). On Deploying Secure Computing Commercially: Private Intersection-Sum Protocols and their Business Applications. *IACR Cryptology ePrint Archive*, *2019*, 723. https://eprint.iacr.org/2019/723/20190618:153102

Iveson, S. (2013). *TCP Over IP Bandwidth Overhead*. https://packetpushers.net/tcp-over-ip-bandwidth-overhead/

Jain, S., Kumar, A., Mandal, S., Ong, J., Poutievski, L., Singh, A., Venkata, S., Wanderer, J., Zhou, J., Zhu, M., Zolla, J., Hölzle, U., Stuart, S., & Vahdat, A. (2013). B4: Experience with a globally-deployed software defined WAN. *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, 3–14. https://doi.org/10.1145/2486001.2486019

Järvinen, K., Kolesnikov, V., Sadeghi, A. R., & Schneider, T. (2010a). Embedded SFE: Offloading server and network using hardware tokens. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *6052 LNCS*, 207–221. https://doi.org/10.1007/978-3-642-14577-3_17

Järvinen, K., Kolesnikov, V., Sadeghi, A. R., & Schneider, T. (2010b). Garbled circuits for leakage-resilience: Hardware implementation and evaluation of one-time programs. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *6225 LNCS*, 383–397. https://doi.org/10.1007/978-3-642-15031-9_26

Jindal, R. P. (2009). From millibits to terabits per second and beyond - Over 60 years of innovation. *2009 2nd International Workshop on Electron Devices and Semiconductor Technology*, 1–6. https://doi.org/10.1109/EDST.2009.5166093

Kalid, S., Syed, A., Mohammad, A., & Halgamuge, M. N. (2017). Big-data NoSQL databases: A comparison and analysis of "Big-Table", "DynamoDB",and "Cassandra". *2017 IEEE 2nd International Conference on Big Data Analysis (ICBDA)*, 89–93. https://doi.org/10.1109/ICBDA.2017.8078782

Khan, M. A., Chiu, M., & Herbordt, M. C. (2013). FPGA-accelerated molecular dynamics. In W. Vanderbauwhede & K. Benkrid (Eds.), *High-performance computing using FPGAs* (pp. 105–135). Springer New York. https://doi.org/10.1007/978-1-4614-1791-0_4

Knott, B., Venkataraman, S., Hannun, A., Sengupta, S., Ibrahim, M., & van der Maaten, L. (2020). CrypTen: Secure Multi-Party Computation Meets Machine Learning. *Proceedings of the NeurIPS Workshop on Privacy-Preserving Machine Learning*. https://lvdmaaten.github.io/publications/papers/crypten.pdf

Kolesnikov, V., & Schneider, T. (2008). Improved garbled circuit: Free XOR gates and applications. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *5126 LNCS*(PART 2), 486–498. https://doi.org/10.1007/978-3-540-70583-3_40

Koomey, J., Berard, S., Sanchez, M., & Wong, H. (2011). Implications of Historical Trends in the Electrical Efficiency of Computing. *IEEE Annals of the History of Computing*, *33*(3), 46–54. https://doi.org/10.1109/MAHC.2010.28

Koti, N., Pancholi, M., Patra, A., & Suresh, A. (2020). *SWIFT: Super-fast and Robust Privacy-Preserving Machine Learning*. https://eprint.iacr.org/2020/592

Kumar, S. (2015). *Fundamental Limits to Moore's Law*. https://arxiv.org/pdf/1511.05956

Lakshman, A., & Malik, P. (2010). Cassandra: A Decentralized Structured Storage System. *SIGOPS Operating Systems Review*, *44*(2), 35–40. https://doi.org/10.1145/1773912.1773922

Leeser, M., Gungor, M., Huang, K., & Ioannidis, S. (2019). Accelerating large garbled circuits on an FPGA-enabled cloud. *Proceedings of H2RC 2019: 5th International Workshop on Heterogeneous High-Performance Reconfigurable Computing - Held*

in conjunction with SC 2019: The International Conference for High Performance Computing, Networking, Storage and Analysis, 19–25. https://doi.org/10.1109/H2RC49586.2019.00008

Liagouris, J., Kalavri, V., Faisal, M., & Varia, M. (2021). *Secrecy: Secure collaborative analytics on secret-shared data* [Technical report BUCS-TR-2021-001]. https://open.bu.edu/handle/2144/41958

Lohr, S. (2013, February 1). *The Origins of 'Big Data': An Etymological Detective Story*. https://bits.blogs.nytimes.com/2013/02/01/the-origins-of-big-data-an-etymological-detective-story/

Malewicz, G., Austern, M. H., Bik, A. J., Dehnert, J. C., Horn, I., Leiser, N., & Czajkowski, G. (2010). Pregel: a system for large-scale graph processing. *Proceedings of the 2010 international conference on Management of data*, 135–146. https://doi.org/10.1145/1807167.1807184

Malkhi, D., Nisan, N., Pinkas, B., & Sella, Y. (2004). Fairplay—A Secure Two-Party Computation System. *13th USENIX Security Symposium (USENIX Security 04)*. https://www.usenix.org/conference/13th-usenix-security-symposium/fairplay%E2%80%94-secure-two-party-computation-system

Marz, N. (2014). *History of Apache Storm and lessons learned*. http://nathanmarz.com/blog/history-of-apache-storm-and-lessons-learned.html

Mashey, John R. (1999). *Big Data ... and the Next Wave of InfraStress*. Retrieved March 15, 2021, from https://static.usenix.org/event/usenix99/invited_talks/mashey.pdf

Masuoka, F., & Iizuka, H. (1985). *Semiconductor memory device and method for manufacturing the same* (US4531203A). https://patents.google.com/patent/US4531203A/en

Mohassel, P., & Rindal, P. (2018). *ABY*^{3}: A mixed protocol framework for machine learning. *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 35–52. https://doi.org/10.1145/3243734.3243760

Moore, G. E. (2006). Cramming more components onto integrated circuits, Reprinted from Electronics, volume 38, number 8, April 19, 1965, pp.114 ff. *IEEE Solid-State Circuits Society Newsletter*, *11*(3), 33–35. https://doi.org/10.1109/N-SSC.2006.4785860

Morgan, S. (2021). *Cybercrime To Cost The World $10.5 Trillion Annually By 2025*. https://cybersecurityventures.com/cybercrime-damages-6-trillion-by-2021/

Mosayyebzadeh, A., Mohan, A., Tikale, S., Abdi, M., Schear, N., Hudson, T., Munson, C., Rudolph, L., Cooperman, G., Desnoyers, P., & Krieger, O. (2019). Supporting

Security Sensitive Tenants in a Bare-Metal Cloud. *2019 USENIX Annual Technical Conference (USENIX ATC 19)*, 587–602. https://www.usenix.org/conference/atc19/presentation/mosayyebzadeh

Naor, M., Pinkas, B., & Sumner, R. (1999). Privacy preserving auctions and mechanism design. *ACM International Conference Proceeding Series*, 129–139. https://doi.org/10.1145/336992.337028

Nielsen, J. (2019). *Nielsen's Law of Internet Bandwidth*. https://www.nngroup.com/articles/law-of-bandwidth/

Ohrimenko, O., Schuster, F., Fournet, C., Mehta, A., Nowozin, S., Vaswani, K., & Costa, M. (2016). Oblivious multi-party machine learning on trusted processors. *25th USENIX Security Symposium (USENIX Security 16)*, 619–636. https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/ohrimenko

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., ... Chintala, S. (2019). PyTorch: An Imperative Style, High-Performance Deep Learning Library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, & R. Garnett (Eds.), *Advances in neural information processing systems*. Curran Associates, Inc. https://proceedings.neurips.cc/paper/2019/file/bdbca288fee7f92f2bfa9f7012727740-Paper.pdf

Patel, R., Wolfe, P. .-F., Munafo, R., Varia, M., & Herbordt, M. (2020). Arithmetic and Boolean Secret Sharing MPC on FPGAs in the Data Center. *2020 IEEE High Performance Extreme Computing Conference (HPEC)*, 1–8. https://doi.org/10.1109/HPEC43674.2020.9286159

Plessl, C. (2018). Bringing FPGAs to HPC Production Systems and Codes. *H2RC'18 workshop at Supercomputing (SC'18)*. https://doi.org/10.13140/RG.2.2.34327.42407

Pu, S., & Liu, J. (2013). Computing Privacy-Preserving Edit Distance and Smith-Waterman Problems on the GPU Architecture. *IACR Cryptology ePrint Archive*. http://eprint.iacr.org/2013/204.pdf

Pu, S., Duan, P., & Liu, J.-C. (2011). Fastplay-A Parallelization Model and Implementation of SMC on CUDA based GPU Cluster Architecture. *IACR Cryptology ePrint Archive*, *2011*, 97. https://eprint.iacr.org/2011/097.pdf

Putnam, A. (2014). A Reconfigurable Fabric for Accelerating Large-Scale Datacenter Services. *ISCA*, 13–24. https://doi.org/10.1109/ISCA.2014.6853195

Raina, R., Madhavan, A., & Ng, A. Y. (2009). Large-Scale Deep Unsupervised Learning Using Graphics Processors. *Proceedings of the 26th Annual International Conference on Machine Learning*, 873–880. https://doi.org/10.1145/1553374.1553486

IaaS vs PaaS vs SaaS. (n.d.). https://www.redhat.com/en/topics/cloud-computing/iaas-vs-paas-vs-saas

Rehman, H. A., Wolfe, P.-F., Jain, S., Hu, S., & Lin, Y. (2021). *MS Poster Presentations Awards*. Retrieved March 14, 2021, from https://www.bu.edu/eng/files/2021/03/MPC-in-Cloud.pdf

Robinson, C. (2021). *Ampere Altra Max M128-30 128 Core Arm Server Update*. https://www.servethehome.com/ampere-altra-max-m128-30-128-core-arm-server-update/

Sanaullah, A., & Herbordt, M. (2018). FPGA HPC using OpenCL: Case Study in 3D FFT. *9th International Symposium on Highly-Efficient Accelerators and Reconfigurable Technologies*, 1–6. https://doi.org/10.1145/3241793.3241800

Sanaullah, A., Yang, C., Alexeev, Y., Yoshii, K., & Herbordt, M. (2018). Real-Time Data Analysis for Medical Diagnosis using FPGA Accelerated Neural Networks. *BMC Bioinformatics*, *19 Supplement 18*. https://doi.org/10.1186/s12859-018-2505-7

Sayilar, G., & Chiou, D. (2014). Cryptoraptor: High throughput reconfigurable cryptographic processor. *2014 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 155–161. https://doi.org/10.1109/ICCAD.2014.7001346

Schneider, T., & Zohner, M. (2013). GMW vs. Yao? Efficient secure two-party computation with low depth circuits. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *7859 LNCS*, 275–292. https://doi.org/10.1007/978-3-642-39884-1_23

Services, A. W. (2016). aws_fpga. https://github.com/aws/aws-fpga.git

Shamir, A. (1979). How to share a secret. *Communications of the ACM*, *22*(11), 612–613. https://doi.org/10.1145/359168.359176

Sheng, J., Xiong, Q., Yang, C., & Herbordt, M. (2017). Collective Communication on FPGA Clusters with Static Scheduling. *ACM SIGARCH Computer Architecture News*, *44*(4). https://doi.org/10.1145/3039902.3039904

Sheng, J., Yang, C., Caulfield, A., Papamichael, M., & Herbordt, M. (2017). HPC on FPGA Clouds: 3D FFTs and Implications for Molecular Dynamics. *27th International Conference on Field Programmable Logic and Applications*. https://doi.org/10.23919/FPL.2017.8056853

Sheng, J., Yang, C., & Herbordt, M. (2015). Towards Low-Latency Communication on FPGA Clusters with 3D FFT Case Study. *HEART*. https://www.bu.edu/caadlab/HEART15.pdf

Sheng, J., Yang, C., & Herbordt, M. (2018). High Performance Dynamic Communication on Reconfigurable Clusters. *28th International Conference on Field Programmable Logic and Applications*. https://doi.org/10.1109/FPL.2018.00044

Shvachko, K., Kuang, H., Radia, S., & Chansler, R. (2010). The Hadoop Distributed File System. *2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, 1–10. https://doi.org/10.1109/MSST.2010.5496972

Songhori, E. M., Riazi, M. S., Hussain, S. U., Sadeghi, A. R., & Koushanfar, F. (2019). ARM2GC: Succinct garbled processor for secure computation. *Proceedings - Design Automation Conference*. https://doi.org/10.1145/3316781.3317777

Songhori, E. M., Zeitouni, S., Dessouky, G., Schneider, T., Sadeghi, A. R., & Koushanfar, F. (2016). GarbledCPU: A MIPS processor for secure computation in hardware. *Proceedings - Design Automation Conference*, *05-09-June*. https://doi.org/10.1145/2897937.2898027

Stern, J., Xiong, Q., Skjellum, A., & Herbordt, M. (2018). A Novel Approach to Supporting Communicators for In-Switch Processing of MPI Collectives. *Workshop on Exascale MPI*. https://www.bu.edu/caadlab/ExaMPI18a.pdf

Sun, J., Peterson, G. D., & Storaasli, O. O. (2008). High-performance mixed-precision linear solver for FPGAs. *IEEE Transactions on Computers*, *57*(12), 1614–1623. https://doi.org/10.1109/TC.2008.89

Thusoo, A., Sarma, J. S., Jain, N., Shao, Z., Chakka, P., Anthony, S., Liu, H., Wyckoff, P., & Murthy, R. (2009). Hive: A Warehousing Solution over a Map-Reduce Framework. *Proceedings of the VLDB Endowment*, *2*(2), 1626–1629. https://doi.org/10.14778/1687553.1687609

Trimberger, S. M. (2015). Three Ages of FPGAs: A Retrospective on the First Thirty Years of FPGA Technology. *Proceedings of the IEEE*, *103*(3), 318–331. https://doi.org/10.1109/JPROC.2015.2392104

VanRoekel, S. (2011). *FedRAMP Policy Memo*. https://cic.gsa.gov/references/policy#FedRAMP

Verma, A., Pedrosa, L., Korupolu, M., Oppenheimer, D., Tune, E., & Wilkes, J. (2015). Large-Scale Cluster Management at Google with Borg. *Proceedings of the Tenth European Conference on Computer Systems*. https://doi.org/10.1145/2741948.2741964

Wagoner, O. (2019). *When to use IaaS, FaaS, PaaS, and CaaS*. https://developer.ibm.com/depmodels/cloud/articles/when-to-use-iaas-faas-paas-and-caas/

Wang, T., Geng, T., Li, A., Jin, X., & Herbordt, M. (2020). FPDeep: Scalable Acceleration of CNN Training on Deeply-Pipelined FPGA Clusters. *TC, C-69*(8), 1143–1158. https://doi.org/10.1109/TC.2020.3000118

Weil, S. A., Brandt, S. A., Miller, E. L., Long, D. D. E., & Maltzahn, C. (2006). Ceph: A Scalable, High-Performance Distributed File System. *7th USENIX Symposium on Operating Systems Design and Implementation (OSDI 06)*. https://www.usenix.org/conference/osdi-06/ceph-scalable-high-performance-distributed-file-system

White, B., Lepreau, J., Stoller, L., Ricci, R., Guruprasad, S., Newbold, M., Hibler, M., Barb, C., & Joglekar, A. (2003). An Integrated Experimental Environment for Distributed Systems and Networks. *SIGOPS Operating Systems Review, 36*(SI), 255–270. https://doi.org/10.1145/844128.844152

Wolfe, P. .-F., Patel, R., Munafo, R., Varia, M., & Herbordt, M. (2020). Secret Sharing MPC on FPGAs in the Datacenter. *2020 30th International Conference on Field-Programmable Logic and Applications (FPL)*, 236–242. https://doi.org/10.1109/FPL50879.2020.00047

Wu, C., Geng, T., Bandara, S., Yang, C., Sachdeva, V., Sherman, W., & Herbordt, M. (2021). Upgrade of FPGA Range-Limited Molecular Dynamics to Handle Hundreds of Processors. *2021 IEEE 29th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*.

Wyden, R. (2019). Student Right to Know Before You Go Act of 2019. https://www.congress.gov/bill/116th-congress/senate-bill/681/all-info

Xiong, Q., Bangalore, P., Skjellum, A., & Herbordt, M. (2018). MPI Derived Datatypes: Performance and Portability Issues. *25th European MPI Users' Group Meeting*. https://doi.org/10.1145/3236367.3236378

Xiong, Q., Skjellum, A., & Herbordt, M. (2018). Accelerating MPI Message Matching Through FPGA Offload. *2018 28th International Conference on Field Programmable Logic and Applications (FPL)*, 191–1914. https://doi.org/10.1109/FPL.2018.00039

Yakoubov, S. (2017). *A Gentle Introduction to Yao's Garbled Circuits*. http://web.mit.edu/sonka89/www/papers/2017ygc.pdf

Yang, C., Geng, T., Wang, T., Patel, R., Xiong, Q., Sanaullah, A., Lin, C., Sachdeva, V., Sherman, W., & Herbordt, M. (2019). Fully Integrated FPGA Molecular Dynamics Simulations. *International Conference for High Performance Computing, Networking, Storage and Analysis*, 1–31. https://doi.org/10.1145/3295500.3356179

Yang, C., Geng, T., Wang, T., Sheng, J., Lin, C., Sachdeva, V., Sherman, W., & Herbordt, M. (2019). Molecular Dynamics Range-Limited Force Evaluation Optimized for FPGA. *2019 IEEE 30th International Conference on Application-specific*

Systems, Architectures and Processors (ASAP), 263–271. https://doi.org/10.1109/ASAP.2019.00016

Yang, C., Sheng, J., Patel, R., Sanaullah, A., Sachdeva, V., & Herbordt, M. (2017). OpenCL for HPC with FPGAs: Case Study in Molecular Electrostatics. *2017 IEEE High Performance Extreme Computing Conference (HPEC)*, 1–8. https://doi.org/10.1109/HPEC.2017.8091078

Yao, A. C. (1986). How to generate and exchange secrets. *27th Annual Symposium on Foundations of Computer Science (sfcs 1986)*, 162–167. https://doi.org/10.1109/SFCS.1986.25

Yao, A. C. (1982). Protocols for Secure Computations. *Annual Symposium on Foundations of Computer Science - Proceedings*, 160–164. https://doi.org/10.1109/sfcs.1982.38

Zaharia, M., Chowdhury, M., Franklin, M. J., Shenker, S., & Stoica, I. (2010). Spark: Cluster computing with working sets. *2nd USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 10)*. https://www.usenix.org/conference/hotcloud-10/spark-cluster-computing-working-sets

Zahur, S., Rosulek, M., & Evans, D. (2015). Two halves make a whole. In E. Oswald & M. Fischlin (Eds.), *Advances in cryptology - eurocrypt 2015* (pp. 220–250). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-662-46803-6_8

Zhao, C., Zhao, S., Zhao, M., Chen, Z., Gao, C.-Z., Li, H., & Tan, Y.-a. (2019). Secure multi-party computation: Theory, practice and applications. *Information Sciences*, *476*, 357–372. https://doi.org/https://doi.org/10.1016/j.ins.2018.10.024

CURRICULUM VITAE