Distributed Hardware Accelerated Secure Joint Computation on the COPA Framework

Rushi Patel*, Pouya Haghi*, Shweta Jain‡, Andriy Kot‡, Venkata Krishnan‡,
Mayank Varia† and Martin Herbordt*
*College of Engineering, Boston University, Boston, MA
†Computing & Data Sciences, Boston University, Boston, MA
‡Intel Corporation, Hudson, MA
Email: *{ruship,haghi,herbordt}@bu.edu †varia@bu.edu ‡{shweta.jain,andriy.kot,venkata.krishnan}@intel.com

***Abstract*—Performance of distributed data center applications can be improved through use of FPGA-based SmartNICs, which provide additional functionality and enable higher bandwidth communication and lower latency. Until lately, however, the lack of a simple approach for customizing SmartNICs to application requirements has limited the potential benefits. Intel's Configurable Network Protocol Accelerator (COPA) provides a customizable FPGA framework that integrates both hardware and software development to improve computation and communication performance. In this first case study, we demonstrate the capabilities of the COPA framework with an application from cryptography – secure Multi-Party Computation (MPC) – that utilizes hardware accelerators connected directly to host memory and the COPA network. We find that using the COPA framework gives significant improvements to both computation and communication as compared to traditional implementations of MPC that use CPUs and NICs. A single MPC accelerator running on COPA enables more than 17Gb/s of communication bandwidth while using only 3% of Stratix 10 resources. We show that utilizing the COPA framework enables multiple MPC accelerators running in parallel to fully saturate a 100Gbps link enabling higher performance compared to traditional NICs.**

I. Introduction

In recent years, one of the biggest technological advancements has come from the growth of public datacenters. These distributed computing environments enable us to continuously surpass previous hardware limitations. Due to the increasing demand for access to datacenter resources, cloud providers strive to find new ways of increasing the overall datacenter performance. The introduction of SmartNICs [1]–[6] has improved the performance of datacenters by enabling intelligent and optimized networks. Microsoft has demonstrated [7]–[9] the use of dedicated FPGAs in SmartNICs for network function offload and cloud management services. Adoption of FPGA SmartNICs continues to increase as a means to accelerate network functions and offload packet processing tasks away from CPU resources [10]–[18].

SmartNICs can be implemented with ASICs providing the primary packet processing, which results in highly efficiency. But the drawback is that ASIC-based SmartNICs cannot be reconfigured or programmed to implement new network functionalities. To address this limitation, there is a growing trend to leverage FPGA-based SmartNICs in datacenters. The reconfigurability of FPGAs provides users with various options for network acceleration. However, FPGA SmartNIC adoption generally relies on vendor support in the form of intellectual property (IP) and software development kits. Even so, FPGAs remain more complex to program and integrated compared to traditional network solutions. The FPGA programmability hurdle and lack of an agnostic production environment between hardware and software has limited the use of FPGA-based SmartNICs to a few large datacenter service providers.

To address the above issues, Intel's Configurable Network Protocol Accelerator (COPA) [19], [20] was developed. COPA utilizes the open source software library, OpenFabric interface (OFI) libfabric [21], for platform-agnostic development and a standard for networking and acceleration invocation. In addition, the COPA hardware framework provides two options to reconfigurable accelerators, inline and lookaside, both of which are directly accessible from the libfabric API. COPA uses the on-board high speed transceivers, e.g., of the Intel Stratix 10 GX, and a uniquely designed architecture to enable high speed remote direct memory access (RDMA) between nodes at 100Gb/s line rate. Unique features include the ability for remote invocation of accelerators and headless operations for host free integration into a distributed data center environment. So far, however, there has been no published work demonstrating or evaluating COPA with respect to a distributed application; that is our goal here.

As a candidate application we have selected Multi-Party Computation (MPC), which would greatly benefit from the features available through the COPA framework. MPC is the cryptographic process of performing calculations on confidential data between multiple organizations while maintaining a level of confidentiality, integrity, and assurance of one's own private data. Parties encode and share their own private data between organizations while maintaining an agreed upon level of security guarantee. This form of joint computation is especially important for industries such as healthcare and finance, as user data is typically under protection through laws and regulations. FPGA accelerated Multi-Party Computation continues to be a progressive research topic [22]–[34] as significant performance improvements can be obtained from hardware acceleration.

This paper argues that combining the COPA tool-set with state of the art MPC algorithms can achieve a lower com-

munication bottleneck for high performance computation inside a datacenter environment. We show that utilizing the COPA system enables a method of performing low-level MPC operations with minimal CPU interaction while enabling improved performance compared to traditional CPU and NIC implementations.

In summary, our contributions are as follows:

- Examine the performance available utilizing the 100Gbps network and configurable lookaside accelerator option of the COPA FPGA framework.
- Adapt hardware accelerated MPC operations to the COPA infrastructure utilizing the configurable FPGA lookaside accelerator enabling significant performance improvements compared against CPU and NICs.
- Using only 3% of the FPGA fabric for secure joint computation, we show that having multiple accelerators running in parallel can saturate the potential 100Gb/s link available through COPA while performing over 6000 MPC operations per second.

II. Background

A. Configurable Network Protocol Accelerator

The COPA FPGA works by using the libfabric software layer with included extensions to queue commands for processing by the hardware. These commands include both RDMA functionality and accelerator specific commands. The RDMA functions use the COPA network TX and RX data paths to perform memory read and write functions without host involvement. Previous work has shown the COPA network can achieve up to 100Gb/S bandwidth with zero-copy direct memory access.

The accelerator specific commands include the use of a number of inline accelerators alongside the RDMA functions. Inline functions operate on packets during transit in a Bump-in-the-Wire method, allowing for data manipulation of packets during egress or ingress of edge nodes. Two examples of inline acceleration are checksum calculation on data in transit or encode/decode functions with pre-shared key pairs.

Additionally, commands can be constructed to trigger the lookaside accelerator functions both locally and on remote COPA nodes. Lookaside functions operate on data at rest, in host memory, and have direct communication with the COPA network to perform additional RDMA functions. This enables the lookaside accelerator to perform data transfer tasks without requiring the host to initiate network operations. Both inline and lookaside accelerator options can be reconfigured by users for application specific functions.

B. Secure Multi-Party Computation

Secret sharing-based multi party computation (MPC) is a method of secure joint computation that allows any number of party members N to work together to obtain a final output [35]–[37]. Confidential data is distributed to all party members in the form of *shares*, where each individual share does not contain enough information to learn anything about the secret data but all shares collectively can be used to decode the data.

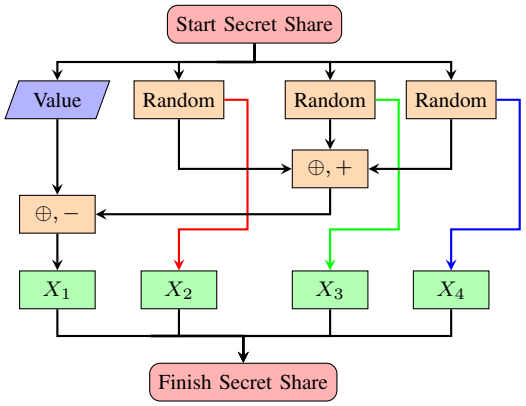

Fig. 1. Initial construction of four party secret shares

Each party member is considered an equal to another, and computation is performed synchronously between all members to maintain accurate share representations of the final value between all members. Specific operations requires members to communicate partial share information among all parties. A large number of communication dependent operations leads to a requirement for high bandwidth and low latency networks. As a consequence, the rate of computation in MPC is bottlenecked by network performance, making this application a prime target for COPA.

Secret sharing MPC is broken down into local and joint computation operations. With FPGAs secret sharing MPC can obtain performance improvements on both forms of computation tasks [33], [34]. Prior research shows that a single FPGA can fully saturate a standard 10GigE NIC while only utilizing a fraction of the available resources of the FPGA hardware. In addition, co-located party members in a datacenter provides an optimal environment to reduce communication latency further improving the performance of communication-dependent MPC operations.

To the best of our knowledge, we are the first to integrate Multi-Party Computation and SmartNIC functionality to improve upon communication bottlenecks. We believe the combination of MPC and COPA lookaside acceleration enables a significant improvement to both computation and communication performance thus eliminating previous network limitations. Utilizing the many unique features of the COPA framework, including remote accelerator triggering and payload processing, allow for headless behavior of many MPC operations between party members. This enables less CPU dependence during concurrent operations between party members and reduces the need for explicit synchronization by each host system.

We focus on the throughput and resource utilization of the multiply operation for a 4-party semi-honest majority MPC algorithm [37]. Our chosen algorithm requires each party member to hold 3-out-of-4 shares of each data element and communicate between all parties equally during calculation. In particular, the key is performing multiplication operations on shares of *128-bit* integer data types which generates three *128-bit* integers for communication to the other party members.

1) Share construction: Our initial experiment setup assumes each party member has control of a single FPGA in a Bump-in-the-Wire scenario tied to a host process under their control. Utilizing this hardware setup, we envision that all confidential data is in the node's possession and the FPGA exclusively interacts with only data in share format. As all confidential data remains tied to the host, this maintains full privacy of confidential data as any data passing through the COPA network is obfuscated in the form of shares.

Shares are generated into four pieces: three uniformly random values, and one value calculated with the input data. We denote each share as x_i and v as our confidential data being shared among the party. Shares are generated and distributed based on the agreed upon computation being performed, either arithmetic or Boolean logic. Shares are generated using uniformly random values x_1, x_2, x_3 which follow the rule stated in Equation 1. Based on the protocol form, values for x_i are created using either Equation 2 or Equation 3. The process of generating four unique shares for the confidential value v is visualized in Figure 1.

$$x_1, x_2, x_3, x_4 \in \mathbb{Z}_{2^n} \tag{1}$$

Arithmetic:

$$x_1 + x_2 + x_3 + x_4 = v \tag{2}$$

Boolean:

$$x_1 \oplus x_2 \oplus x_3 \oplus x_4 = v \tag{3}$$

2) Gate Operations - Computation phase: We focus on arithmetic MPC operations which consist of addition and multiplication. Shares are formed using *128-bit* values and all operations are based around modular arithmetic with a ring module of 2^{128}. Functionality for Boolean shares is similar to arithmetic and our implementation allows for these additional operations with a simple selector input to change gate function.

To perform any operation two shares are passed to each party, one for each data value 'v_1, v_2'. We denote an input for v_1 as x, v_2 as y and the output of the operation as z. For our algorithm of choice, MPC addition consists of local computation only and doesn't require interaction between members. Shown in Equation 4, shares can be added in parallel and only require local computation done synchronously between all party members.

Addition:

$$\begin{aligned} z_1 &= x_1 + y_1 \\ z_2 &= x_2 + y_2 \\ z_3 &= x_3 + y_3 \\ z_4 &= x_4 + y_4 \end{aligned} \tag{4}$$

The base multiplication algorithm requires both local computation of shares and a single round of communication between party members; we thus focus on the performance obtained through improvement of these communication rounds. Each multiplication operation is broken down into the accumulation of share multiplications as seen in Equation 5. Since each party member only contains 3-out-of-4 share information, multiplication cannot be done locally due to the missing share information. There are two scenarios in which the distribution of share information creates unique situations during multiplication operation.

TABLE I
PARTY '3-1' MISSING INFORMATION

Share	Party 1	Party 2	Party 3	Party 4
z_1'		x_1y_1	x_1y_1	x_1y_1
z_2'	x_2y_2		x_2y_2	x_2y_2
z_3'	x_3y_3	x_3y_3		x_3y_3
z_4'	x_4y_4	x_4y_4	x_4y_4	

TABLE II
PARTY '2-2' MISSING INFORMATION

Party 1	Party 2	Party 3	Party 4
		$x_1y_2 + x_2y_1$	$x_1y_2 + x_2y_1$
	$x_1y_3 + x_3y_1$		$x_1y_3 + x_3y_1$
	$x_1y_4 + x_4y_1$	$x_1y_4 + x_4y_1$	
$x_2y_3 + x_3y_2$			$x_2y_3 + x_3y_2$
$x_2y_4 + x_4y_2$		$x_2y_4 + x_4y_2$	
$x_3y_4 + x_4y_3$	$x_3y_4 + x_4y_3$		

Multiplication:

$$\begin{aligned} x * y &= x_1y_1 + x_1y_2 + x_1y_3 + x_1y_4 \\ &= x_2y_1 + x_2y_2 + x_2y_3 + x_2y_4 \\ &= x_3y_1 + x_3y_2 + x_3y_3 + x_3y_4 \\ &= x_4y_1 + x_4y_2 + x_4y_3 + x_4y_4 \end{aligned} \tag{5}$$

The first case occurs when performing the individual share multiplication of two of the same shares x_i and y_i, shown in Table I. Three of the four party members contain enough information to perform the local computation. In this scenario, no communication occurs as the party members which need the result z_i' contain all the necessary information. Local computation is performed and is included in the final multiplication result.

The second case is unique due to party members missing exactly half of their required information to fully complete the computation. As we demonstrate in Table II, only two party members contain enough information to perform the complete computation and the remaining two party members only have half of the necessary information. We will refer to this step as the '2-2' computation. To perform an accurate and secure '2-2' computation, the process involves usage of a correlated random value between the knowledgeable party members and one communication round with the remaining members.

To generate correlated random values, each party member maintains hold of a shared key with one other party member. The '2-2' calculation involves exactly two computations between a pair of party members, therefore six random numbers will be generated two for each shared key.

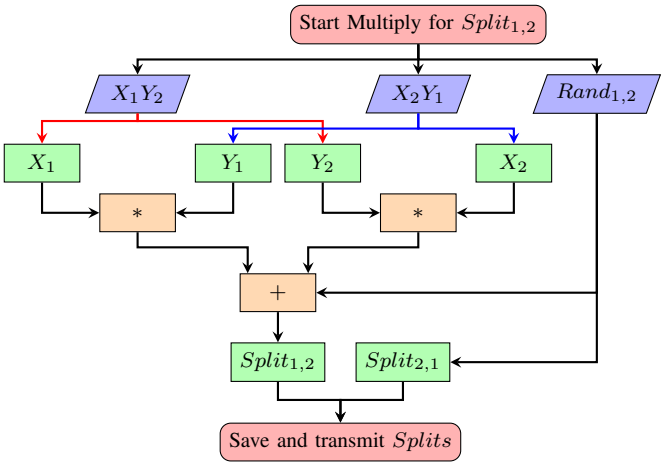

Fig. 2. Four Party Multiply Operation Split Calculation

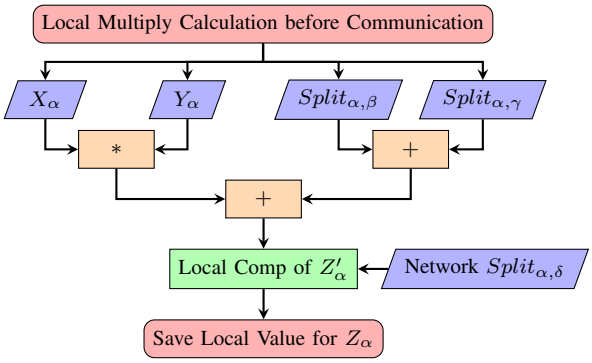

Fig. 3. Four Party Complete Multiply Operation

Parties with full knowledge of sub-calculations in '2-2' perform the local computation with their information. Directly sharing this information with the other two parties would result in a convergence in knowledge and thus reduce the security of the operation. Instead the correlated random number is used to create a new sharing pair with the completed local computation shown in Figure 2. These values will be shared with the remaining party members to complete the operation.

Prior to the communication round, each party member combines the earlier computed $x_i * y_i$ result and two selected splits from each pair of the local computations in '2-2' per share. This pre-communication requirement is shown in Figure 3. They store this value in local memory awaiting the values shared by the other party members to fully complete the multiplication operation.

3) Communication phase: Each pair of party members perform individual '2-2' and jointly choose which piece should be sent to the remaining parties. For base protocol functionality, only a single value must be obtained from each other party member to have an accurate and complete multiplication operation. Since the '2-2' computation involves two party members jointly computing the same information a unique opportunity is created to add additional malicious security. Malicious security is done with the second party member

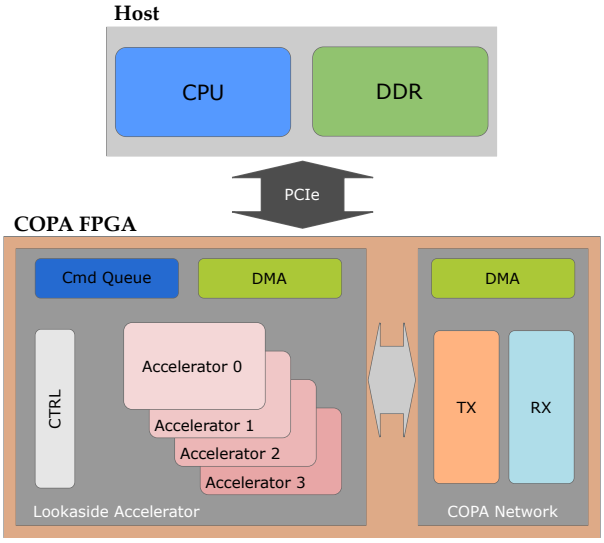

Fig. 4. COPA architecture connected to host system through PCIe. The COPA FPGA contains a lookaside accelerator implementation directly connected to the COPA network allowing for network functions without host interaction.

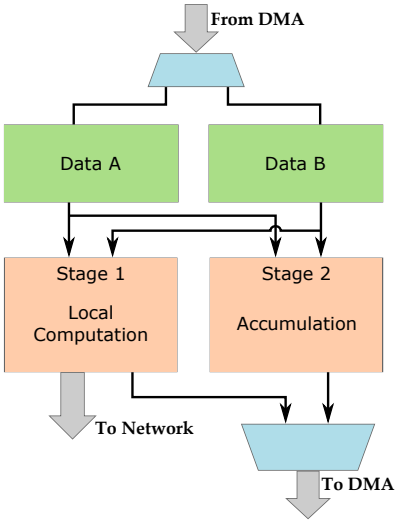

Fig. 5. Two-stage MPC implementation as single lookaside accelerator.

sharing additional information to the same destination as the first. The second round of communication typically involves creating a hashed value of completed share information and performing another round of communication upon completion of the above steps. Hashes of reduced size are typically used to minimize the total amount of communication bits in the overall process, which reduces the overall bandwidth requirement of the added malicious security step. The locally generated hashed value and transmitted value over the network are compared and a mismatch in calculated hashes would indicate computation has either failed or altered by a party member. The party members can then abort future computation to avoid the risk of data leakage in the case of a malicious actor. Since all intermediate data is in the form of shares, no additional information would be available by a single party member, and trusted members will not share any data after aborting the current process.

III. Architecture Implementation

A. Initial Protocol Requirements

Our initial design uses the host to generate, and COPA unidirectional PUT functions to distribute, shares of private data to the other parties. Each share is generated through of the use of a random number generator and a specific calculation following the agreed upon algorithm protocol. In our implementation, we use the 4-party MPC protocol of Dalskov, et al. [37]; we discuss algorithm specifics in Section IV-A. As all confidential data remains tied to the host, this maintains full privacy of confidential data as any data passing through the COPA network is obfuscated in the form of shares.

In addition to distributing shares between parties, a set of keys are also allocated for further computation when pseudorandom generated numbers must be known by two or more parties. Each party member uses a set of unique keys to generate random numbers concurrently with other party members; however, each party member does not contain knowledge of all keys. This concurrent behavior is important to maintain protocol accuracy between party members during operation and can help avoid additional communication.

B. Lookaside Accelerator

The COPA lookaside architecture uses separate accelerator logic outside of the COPA network as seen in Figure 4. Acceleration is initially controlled by the host through a unique command containing the source data location, destination location, length of data, and type of operation. A global control unit manages incoming commands in the queue and assigns them to appropriate accelerators. This feature enables a single host to issue commands to different accelerators for added parallelism or unique functions. We use only a single MPC accelerator for our initial tests, but discuss the improvements available with additional accelerators. Each accelerator initially collects the source data through a DMA operation from the host memory. If source data is unavailable locally, i.e., found on a remote COPA node, then a COPA network command is generated and used to obtain data from the remote node prior to DMA operation. Following the local DMA completion, acceleration is performed and final calculations are sent back to host memory through a second DMA operation. If the destination memory location is for a remote node, then the COPA network is again used to send the final completed values to another COPA node on the network.

1) MPC Accelerator Core: To fully utilize the COPA lookaside accelerator we split up the multiplication operations into the local computation stage and a post communication accumulation stage as seen in Figure 5. Data is first obtained by the DMA logic and stored into two sets of on-chip memory, Data A and Data B. Data in these two on-chip memory regions are based on the stage of computation being performed. First stage computation takes two lists of values in share form, while the second stage contains the intermediate share data and communication data received from other party members. The local computation stage uses a pseudo random number generator and on-chip resources to perform the initial MPC calculations and save the intermediate share information back to host memory. Addition operations only require the use of the first stage accelerator to perform calculation on input data and generate complete shares.

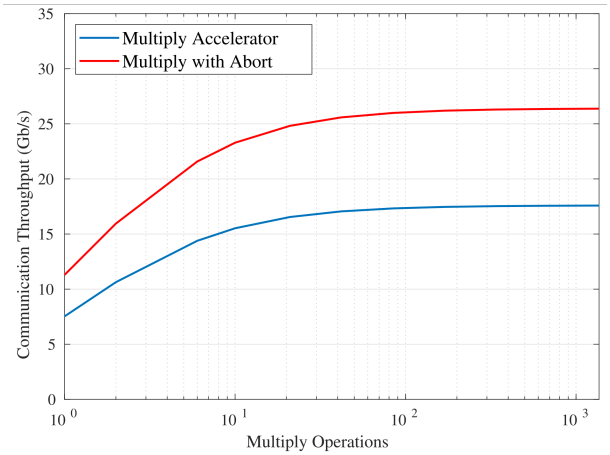

Fig. 6. Throughput comparison between base MPC accelerator and MPC with malicious security running at 275 MHz with varying batch sizes of multiply operations. Malicious security requires an additional collision resistant hash value to be transferred. Saturation of the base accelerator is over 17.5Gb/s and with malicious security is over 26.3Gb/s.

TABLE III
Single MPC lookaside FPGA utilization

Stratix 10 FPGA	Raw (Total Percentage)
Freq	250MHz - 275MHz
ALM	10667 (1%)
Memory bits	5,156,500 (2%)
RAM blocks	668 (6%)
DSP blocks	150 (3%)

Multiplication operations use the first stage to prepare partial local shares and data for communication to other party members. Stage 1 hardware implementation is fully pipelined with the help of parallel random number generators working in unison to produce the six necessary random values per cycle of input data. This results in local computation being available every cycle after the initial startup delay.

On completion of the first stage, party members prepare data for the communication phase to pass along via the COPA network. The lookaside accelerator uses dedicate hardware command queues to create PUT commands which simultaneously transmit the prepared data to each correct party member and signal the host system for stage 2 operation on data transmission completion. This process both prepares the data for processing in the second stage of the multiply operation and enables the host to trigger second stage operation. Stage 2 performs a simple accumulation of the locally generated intermediate shares and ingress data from party members, saving the result back into host memory for future computation.

IV. Results

A. Experiment Setup

We implement our hardware design on the COPA framework using Intel Stratix 10 FPGAs interconnected with

100GigE high speed transceivers. Each party maintains ownership of a single FPGA connected to a host system using the COPA framework for communication between party members. Acceleration is performed through the use of unique lookaside accelerator commands sent from each host system to the FPGA lookaside accelerator through a dedicated accelerator queue. The lookaside command format allows for batch operations on a stream of data from a specified source and saves local computation back to host memory while preparing the network data for transfer to each party member.

Each party member is directly connected with every other party member. We focus on the bandwidth requirements of a single party member, thus do not consider additional latency of switching hardware and any additional control flow commands in our data collection. Both simulation of accelerator throughput and real world bandwidth results are used for final verification of results.

B. Analysis

Resource utilization for a single MPC lookaside accelerator can be found in Table III. This shows the implementation uses minimal resources which allows for the inclusion of more accelerators into each COPA FPGA; these additional accelerators may include multiple instances of the MPC core operations or additional functionality for High Performance Computing applications such as collectives. With the inclusion of a single MPC accelerator, Figure 6 shows how much data is available for communication based on the input length of the lookaside accelerator source data.

The pipeline implementation of the accelerator allows for data to be processed and ready for communication every cycle, after an initial startup delay accessing host memory. Using a single accelerator and batching multiplication operations over a stream of source data, the accelerator performs enough computation to saturate a traditional 10Gb/s link. These results are similar to past implementations [33], [34] and show that integration with the COPA system is beneficial to improve the total throughput possible with these hardware implemented MPC operations.

Examining the throughput of large batches of multiplication operations, Figure 6 shows a single accelerator performing the basic algorithm (without abort) can saturate a 17.5Gb/s connection, while the inclusion of additional malicious security for abort requires larger than 26.3Gb/s connection to avoid saturation. We can therefore include up to 6 MPC accelerators without abort, or 4 MPC accelerators with abort, to saturate the COPA network.

In addition to the communication improvements, the COPA system enables a set-and-forget method for acceleration and communication which frees up each host processor to perform additional non-MPC functions. Queuing operations into the lookaside accelerator, with knowledge that data will be shared appropriately, allows for final completion of each operation without the need to block the process on each step.

V. Conclusion

The COPA framework enables hardware acceleration and improved network functions for, potentially, many different applications. We show how an MPC implementation fits into the COPA framework and enables improvements to both computation and communication by using the lookaside accelerator features and improved network data transfer. In addition, MPC running on the COPA system enables the use of the an open-source software library, OFI, as an alternative to specialized MPC software used by each party member. We aim to increase the size of our tests with additional MPC algorithms aiming for two party, three party, and four party computation alternatives. We also aim to enable more accelerator options to improve on secure joint computation through the means of memory operations such as scatter/gather. This will enable additional improvements to both MPC throughput and network communication.

Our future work will include a full end-to-end method of MPC using the COPA hardware/software infrastructure for cloud/data centers, MPC-as-a-Service. Fully utilizing the COPA features, we aim to create a fully autonomous MPC system allowing for any host to perform trusted operations on the collective data. With COPA remote invocation of target accelerator nodes, we aim to enable complete MPC applications to run with only a single host triggering the operation and each additional party member acting like a headless target node. By using all of these features MPC-as-a-Service can be a viable method of trusted secure joint computation with a minimal barrier to entry.

Acknowledgment

This material is based upon work supported by the National Science Foundation under Grants No. 1718135 and 1915763, by a grant from Red Hat, and by DARPA and the Naval Information Warfare Center (NIWC) under Contract No. N66001-15-C-4071.

References

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

[2] NVIDIA. Bluefield™ smartnic ethernet. [Online]. Available: https://www.mellanox.com/products/BlueField-SmartNIC-Ethernet

[3] ——. NVIDIA Mellanox Innova-2 Flex Open Programmable SmartNIC. [Online]. Available: https://www.nvidia.com/en-us/networking/ethernet/innova-2-flex/

[4] H. Shahzad, A. Sanaullah, and M. Herbordt, "Survey and Future Trends for FPGA Cloud Architectures," in *IEEE High Performance Extreme Computing Conference*, 2021, dOI: 10.1109/HPEC49654.2021.9622807.

[5] C. Bobda, J. Mandebi, P. Chow, M. Ewais, N. Tarafdar, J. Vega, K. Eguro, D. Koch, S. Handagala, M. Leeser, M. Herbordt, H. Shahzad, P. Hofstee, B. Ringlein, J. Szefer, A. Sanaullah, and R. Tessier, "The Future of FPGA Acceleration in Datacenters and the Cloud," *ACM Transactions on Reconfigurable Technology and Systems*, vol. 15, no. 3, pp. 1–42, 2022, doi: 10.1145/3506713.

[6] M. Zink, D. Irwin, E. Cecchet, H. Saplakoglu, O. Krieger, M. Herbordt, M. Daitzman, P. Desnoyers, M. Leeser, and S. Handagala, "The Open Cloud Testbed (OCT): A Platform for Research into new Cloud Technologies," in *IEEE International Conference on Cloud Networking (IEEE CloudNet)*, 2021, doi: 10.1109/CloudNet53349.2021.9657109.

[7] A. Caulfield, E. Chung, A. Putnam, H. Angepat, J. Fowers, M. Haselman, S. Heil, M. Humphrey, P. Kaur, J.-Y. Kim, D. Lo, T. Massengill, K. Ovtcharov, M. Papamichael, L. Woods, S. Lanka, D. Chiou, and D. Burger, "A cloud-scale acceleration architecture," in *49th IEEE/ACM Int. Symp. Microarchitecture*, 2016, pp. 1–13.

[8] D. Firestone, A. Putnam, S. Mundkur, D. Chiou, A. Dabagh, M. Andrewartha, H. Angepat, V. Bhanu, A. Caulfield, E. Chung, H. K. Chandrappa, S. Chaturmohta, M. Humphrey, J. Lavier, N. Lam, F. Liu, K. Ovtcharov, J. Padhye, G. Popuri, S. Raindel, T. Sapre, M. Shaw, G. Silva, M. Sivakumar, N. Srivastava, A. Verma, Q. Zuhair, D. Bansal, D. Burger, K. Vaid, D. A. Maltz, and A. Greenberg, "Azure accelerated networking: SmartNICs in the public cloud," in *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*. Renton, WA: USENIX Association, Apr. 2018, pp. 51–66. [Online]. Available: https://www.usenix.org/conference/nsdi18/presentation/firestone

[9] B. Li, K. Tan, L. L. Luo, Y. Peng, R. Luo, N. Xu, Y. Xiong, P. Cheng, and E. Chen, "Clicknp: Highly flexible and high performance network processing with reconfigurable hardware," in *Proceedings of the 2016 ACM SIGCOMM Conference*, ser. SIGCOMM '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 1–14. [Online]. Available: https://doi.org/10.1145/2934872.2934897

[10] Y. Le, H. Chang, S. Mukherjee, L. Wang, A. Akella, M. M. Swift, and T. V. Lakshman, "Uno: Uniflying host and smart nic offload for flexible packet processing," in *Proceedings of the 2017 Symposium on Cloud Computing*, ser. SoCC '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 506–519. [Online]. Available: https://doi.org/10.1145/3127479.3132252

[11] W. Schonbein, R. E. Grant, M. G. F. Dosanjh, and D. Arnold, "Inca: In-network compute assistance," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '19. New York, NY, USA: Association for Computing Machinery, 2019. [Online]. Available: https://doi.org/10.1145/3295500.3356153

[12] H. Eran, L. Zeno, M. Tork, G. Malka, and M. Silberstein, "NICA: An infrastructure for inline acceleration of network applications," in *2019 USENIX Annual Technical Conference (USENIX ATC 19)*. Renton, WA: USENIX Association, Jul. 2019, pp. 345–362. [Online]. Available: https://www.usenix.org/conference/atc19/presentation/eran

[13] Q. Xiong, C. Yang, R. Patel, T. Geng, A. Skjellum, and M. Herbordt, "GhostSZ: A Transparent SZ Lossy Compression Framework with FPGAs," in *2019 IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2019, pp. 258–266, doi: 10.1109/FCCM.2019.00042.

[14] S. Miano, R. Doriguzzi-Corin, F. Risso, D. Siracusa, and R. Sommese, "Introducing smartnics in server-based data plane processing: The ddos mitigation use case," *IEEE Access*, vol. 7, pp. 107 161–107 170, 2019.

[15] M. Tork, L. Maudlej, and M. Silberstein, *Lynx: A SmartNIC-Driven Accelerator-Centric Architecture for Network Servers*. New York, NY, USA: Association for Computing Machinery, 2020, p. 117–131. [Online]. Available: https://doi.org/10.1145/3373376.3378528

[16] S. Grant, A. Yelam, M. Bland, and A. C. Snoeren, "SmartNIC Performance Isolation with FairNIC: Programmable Networking for the Cloud," in *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication*, ser. SIGCOMM '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 681–693. [Online]. Available: https://doi.org/10.1145/3387514.3405895

[17] P. Haghi, T. Geng, A. Guo, T. Wang, and M. Herbordt, "Reconfigurable Compute-in-the-Network FPGA Assistant for High-Level Collective Support with Distributed Matrix Multiply Case Study," in *IEEE Conference on Field Programmable Technology*, 2020.

[18] A. Guo, T. Geng, Y. Zhang, P. Haghi, C. Wu, C. Tan, Y. Lin, A. Li, and M. Herbordt, "A Framework for Neural Network Inference on FPGA-Centric SmartNICs," in *International Conference on Field-Programmable Logic and Applications*, 2022.

[19] V. Krishnan, O. Serres, and M. Blocksome, "COnfigurable Network Protocol Accelerator (COPA)," in *2020 IEEE Symposium on High-Performance Interconnects (HOTI)*, 2020.

[20] ——, "Configurable Network Protocol Accelerator (COPA)," *IEEE Micro*, vol. 41, no. 1, pp. 8–14, 2020.

[21] P. Grun, S. Hefty, S. Sur, D. Goodell, R. D. Russell, H. Pritchard, and J. M. Squyres, "A brief introduction to the openfabrics interfaces - a new network api for maximizing high performance application efficiency," in *2015 IEEE 23rd Annual Symposium on High-Performance Interconnects*, 2015, pp. 34–39.

[22] K. Järvinen, V. Kolesnikov, A. R. Sadeghi, and T. Schneider, "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)*, vol. 6052 LNCS, pp. 207–221, 2010.

[23] ——, "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)*, vol. 6225 LNCS, pp. 383–397, 2010.

[24] T. K. Frederiksen, T. P. Jakobsen, and J. B. Nielsen, "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)*, vol. 8642, no. grant 61061130540, pp. 358–379, 2014.

[25] E. M. Songhori, S. Zeitouni, G. Dessouky, T. Schneider, A. R. Sadeghi, and F. Koushanfar, "GarbledCPU: A MIPS processor for secure computation in hardware," *Proceedings - Design Automation Conference*, vol. 05-09-June, 2016.

[26] S. U. Hussain, B. D. Rouhani, M. Ghasemzadeh, and F. Koushanfar, "MAXelerator: FPGA Accelerator for Privacy Preserving Multiply-Accumulate (MAC) on Cloud Servers," *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*, pp. 1–6, 2018.

[27] E. M. Songhori, M. S. Riazi, S. U. Hussain, A. R. Sadeghi, and F. Koushanfar, "ARM2GC: Succinct garbled processor for secure computation," *Proceedings - Design Automation Conference*, 2019.

[28] S. U. Hussain and F. Koushanfar, "FASE: FPGA acceleration of secure function evaluation," *Proceedings - 27th IEEE International Symposium on Field-Programmable Custom Computing Machines, FCCM 2019*, pp. 280–288, 2019.

[29] X. Fang, S. Ioannidis, and M. Leeser, "Secure function evaluation using an FPGA overlay architecture," *FPGA 2017 - Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pp. 257–266, 2017.

[30] ——, "SIFO: Secure computational infrastructure using FPGA overlays," *International Journal of Reconfigurable Computing*, vol. 2019, 2019.

[31] K. Huang, M. Gungor, X. Fang, S. Ioannidis, and M. Leeser, "Garbled circuits in the cloud using FPGA enabled nodes," *2019 IEEE High Performance Extreme Computing Conference, HPEC 2019*, pp. 1–6, 2019.

[32] M. Leeser, M. Gungor, K. Huang, and S. Ioannidis, "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*, pp. 19–25, 2019.

[33] P.-F. Wolfe, R. Patel, R. Munafo, M. Varia, and M. Herbordt, "Secret Sharing MPC on FPGAs in the Datacenter," in *IEEE Conference on Field Programmable Logic and Applications*, 2020.

[34] R. Patel, P.-F. Wolfe, R. Munafo, M. Varia, and M. Herbordt, "Arithmetic and Boolean Secret Sharing MPC on FPGAs in the Data Center," in *IEEE High Performance Extreme Computing Conference*, 2020.

[35] T. Araki, J. Furukawa, Y. Lindell, A. Nof, and K. Ohara, "High-throughput semi-honest secure three-party computation with an honest majority," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 805–817. [Online]. Available: https://doi.org/10.1145/2976749.2978331

[36] D. Demmler, T. Schneider, and M. Zohner, "ABY - A framework for efficient mixed-protocol secure two-party computation," in *NDSS*. The Internet Society, 2015.

[37] A. Dalskov, D. Escudero, and M. Keller, "Fantastic four: Honest-majority four-party secure computation with malicious security," in *30th {USENIX} Security Symposium ({USENIX} Security 21)*, 2021.