Optimizing Designs Using Several Types of Memories on Modern FPGAs

Mehmet Gungor
Dept. of ECE
Northeastern University
Boston, MA
0000-0001-5154-1809

Kai Huang
Dept. of ECE
Northeastern University
Boston, MA
0000-0001-9173-9393

Stratis Ioannidis
Dept. of ECE
Northeastern University
Boston, MA
0000-0001-8355-4751

Miriam Leeser
Dept. of ECE
Northeastern University
Boston, MA
0000-0002-5624-056X

2022 IEEE High Performance Extreme Computing Conference (HPEC) | 978-1-6654-9786-2/22/$31.00 ©2022 IEEE | DOI: 10.1109/HPEC55821.2022.9926306

***Abstract*—Modern FPGAs targeting data centers are designed to accelerate problems with large data. They offer many different types of memory including on-chip and on-board memories. A recent addition is High Bandwidth Memory (HBM), whose advantages have been demonstrated by others. However, there is little research that looks at how interactions among different memory types impact application performance. We investigate how a combination of HBM and on-chip memory (BRAM or URAM) impact clock rate and overall application latency. In these designs, the on-chip memory is used as an on-chip cache for the larger amounts of data stored in HBM. Our experiments show that as the size of data stored in BRAM or URAM increases, the achievable clock speed is reduced. This in turn may result in degraded performance. We examine Garbled Circuits, an implementation of Secure Function Evaluation (SFE) with high memory demands and out-of-order data access, and examine how different choices of BRAM, URAM and HBM usage alters its performance.**

***Index Terms*—High Bandwidth Memory, FPGA**

I. Introduction

An increasing challenge in modern processor design is delivering data to the processing elements. This is challenging in manycore processors with increasingly sophisticated caching mechanisms and Graphics Processing Units (GPUs) with hundreds of cores, as well as in FPGA designs. To address this, processors and accelerators including CPUs, GPUs and FPGAs are offering High Bandwidth Memory (HBM) which, as its name implies, provides increased bandwidth to memory by connecting the memory controller directly through a substrate. Thus HBM is in package, but not on die. It offers high density coupled with significantly higher bandwidth than DDR, which is off-chip.

Modern FPGA architectures offer a variety of different memory resources. For example, the AMD/Xilinx Alveo U280 [1], used in this research, has four types of memory available to the FPGA designer, namely (1) on-chip Block RAM (BRAM), (2) on-chip Ultra RAM (URAM), (3) HBM and (4) off-chip DDR memory. Each of these types of memory has different characteristics, including size, latency, and access bandwidth. Intel FPGAs have a similar memory architecture. The Stratix 10, contains 20-kilobit (Kb) M20K blocks and 47.25-Megabit eSRAM blocks as well as HBM. To maximize bandwidth, a designer may try to use several or all the different types of memories to make use of each memory's distinct interface and bandwidth. Our research shows that the different types of memories interact in unexpected ways and that peak performance for memory-bound applications can only be achieved by understanding these complex interactions. A number of papers evaluate HBM memory, its performance and behavior [2]–[4]. Tools for efficient use of HBM, such as HBM Connect [5] are currently being developed. This previous research focuses on HBM performance alone and does not consider HBM in conjunction with other memory types.

Others have studied different types of FPGA memory on the Ultrascale+ architecture [6]. However, their study did not include HBM or UltraRAM, but only looked at Block RAM vs. DDR. We consider HBM and UltraRAM as well as BRAM. In contrast to previous work, our research investigates tradeoffs when using multiple different memories in the same design, and includes URAM, BRAM, and HBM in the memory types considered. We specifically target applications with large data needs, and focus on applications that do not access data in a streaming manner. These applications are typical of data center applications.

Many factors impact performance, including memory latency, and the size of memory used, as well as the pattern with which memory is accessed. For example, with HBM, the user can set the number of parallel ports. The overall performance of the design usually correlates with the maximum clock speed that the design can achieve, which is determined by other choices and can not be set directly. In our experiments, we observe that using a large percentage of the available BRAM or URAM on an FPGA can cause complex routing which reduces the maximum clock speed, and this can adversely impact overall application performance. We have also found that in some cases, a lower clock speed may provide the best overall performance if it corresponds to a lower latency for data accesses. These complex interactions make it difficult for a designer to choose the optimal memory layout for their application. This research focuses on studying these interactions with the goal of providing users of FPGAs for applications that require a large amount of data with advice on how best to make use of the different options.

The contributions of this research are:

This research is funded by NSF SaTC Grant No. 1717213.

1) A study of how different sizes of BRAM and URAM used in conjunction with HBM change the clock speed of applications,
2) A study of overall latency for an application with a non-streaming data access pattern and how results change as the memory used and size of the data scales, and
3) An analytical approach that predicts the drop in clock speed for an application based on its memory layout.

The rest of this paper is organized as follows. In Sec. II, background on the memories in the Xilinx Alveo U280 as well as related work. In Sec. III, we present the target applications, including Garbled Circuits. In Sec. IV, the experiments to analyze different types of memory usage are described along with results. An analysis of the lessons learned are presented in Sec. V, followed by conclusions and plans for future work.

II. Background

TABLE I
MEMORY FOR ALVEO U280

Memory	Capacity	Bandwidth	Ports	Rd Latency
Block RAM	9.072MB	5.4GB/s	2 per BRAM	1-2
UltraRAM	34.56MB	1.35GB/s	2 per block	1-5
HBM	8GB	460GB/s	32 (max)	40-50
DDR	32GB	38 GB/s	2	50-60

A. Memory Types

We target the Xilinx Alveo U280 [1] in this research, which is particularly designed for data center usage. The Alveo, based on the Xilinx Ultrascale+ architecture, has four distinct types of memory: Block RAM (BRAM), UltraRAM (URAM), HBM and DDR [7]. Information regarding these different memory types, is shown in Table II, and includes total capacity, peak bandwidth, maximum ports, and read latency in clock cycles.

Block RAM (BRAM) is integrated within the FPGA fabric and thus supports very low latency accesses. Each Block RAM in the Xilinx UltraScale architecture-based devices stores up to 36 Kbits of data and can be configured as either two independent 18 Kb RAMs, or one 36 Kb RAM. The U280 has 2016 Block RAMs which can be configured as true dual port or simple dual port, where simple dual port mode is less flexible than true dual port mode.

The Alveo U280 also has 960 UltraRAMs on chip. UltraRAM blocks are 288 Kb, organized as 72 bits x 4K entries. URAMs are single clock, synchronous memory blocks arranged in columns in the device. Each UltraRAM has two ports, and each port can perform either a read or a write operation per cycle. UltraRAM provides fast access as it is on chip, however with longer latency compared to BRAM.

HBM is an in package, but off chip memory that supports high bandwidth. The Alveo U280 has 8GB of HBM with peak memory bandwidth of 460GB/sec. In theory HBM supports up to 32 ports; however, the current Vitis tools allow developers to generate 16 AXI-4 ports for communicating with HBM. Peak bandwidth is achieved when all the ports are reading data at the same time from different banks in burst mode. If we let 8 or 16 ports issue 25 read operations to the same memory bank simultaneously and the ports do not operate in burst mode, the latency for each port is observed to be 55-60 clock cycles. The address stride we set for the experiments are 0, 64, 1K and 4K. For address strides of 0, 64 and 1K, the read latency falls in this range. However, for a stride of 4K, the performance degrades. If we use 8 ports, the average latency reaches 68 to 70 cycles. For 16 ports, the number can be over 80. Since fetches to the same bank are serialized, the overall bandwidth is low, and the resulting performance is no better than using DDR. Hence, the layout of memory needs to be considered to maximize the amount of memory that can be fetched in a short amount of time from HBM.

DDR is off chip memory. The Alveo has 32 GB of DDR memory with peak bandwidth of 38GB/sec. DDR is the largest memory, but provides an order of magnitude less bandwidth than HBM. Note that for both DDR and HBM, peak memory bandwidth depends on accessing memory locations in burst mode and on having different ports access different memory banks. We do not consider the use of DDR in this paper.

B. Garbled Circuits

Our research accelerates Secure Function Evaluation (SFE), specifically Garbled Circuits (GC), using FPGAs. In this model, there are two or more users with data which they wish to keep private, and a function to be evaluated over that data. All parties know the function being evaluated and learn the outcome of the evaluation, but users do not reveal their data. The threat model we follow is "honest but curious" where an adversary follows the protocol as specified, but tries to learn as much as possible.

Garbled circuits were initially introduced by Yao [8] for two users and has been extended to multiple users. The implementation of GC relies on cryptographic primitives. In the variant we study here (adapted from [9], [10]), Yao's protocol runs between (a) a set of private input owners, (b) an Evaluator, who wishes to evaluate a function over the private inputs, and (c) a third party called the Garbler, who facilitates and enables the secure computation. Several improvements over the original Yao's protocol have been proposed, that lead to both computational and communication cost reductions. These include point-and-permute [11], row reduction [12], and Free-XOR [13] optimizations, all of which we implement in our design.

Garbled Circuits work for any problem that can be expressed as a Boolean circuit. In our and many other implementations, this function is represented as a circuit made up of AND and XOR gates.[1] The Evaluator wishes to evaluate a function f, represented as a Boolean circuit of AND and XOR gates, over private user inputs $x_1, x_2, \ldots, x_n$. We break the problem into three phases, as shown in Fig. 1. In Phase I, the Garbler "garbles" each gate of the circuit, outputting (a) a "garbled

[1]Recall that AND and XOR gates form a complete basis for Boolean circuits.

978-1-6654-9786-2/22/$31.00 ©2022 IEEE

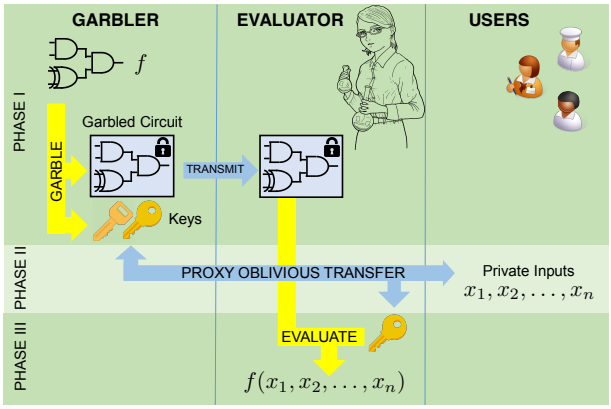

Fig. 1. Yao's Protocol Phases of Operation

circuit," namely, the garbled representation of every gate in the circuit representing f, and (b) a set of keys, each corresponding to a possible value in the string representing the inputs $x_1, \ldots, x_n$. These values are shared with the Evaluator. In Phase II, through proxy oblivious transfer [14], the Evaluator learns the keys corresponding to the true user inputs. In the final phase, the Evaluator uses the keys as input to the garbled circuit to evaluate the circuit, un-garbling the gates. At the conclusion of Phase III, the Evaluator learns $f(x_1, \ldots, x_n)$. More details of the design are described elsewhere [15], [16]. Here we focus on memory access and memory usage for large designs, as this is a bottleneck in implementing GC on FPGAs. The challenge lies in the fact that memory is accessed out of order rather than in a streaming fashion, and cannot be easily broken up into substructures.

C. Related Work in Memory on FPGAs

Previous research regarding memory on FPGAs focuses on one or two types of memory and not the three types considered here. There are several papers on the performance of HBM memory alone and how to optimize its use [2], [3]. Shuhai [3] focuses specifically on HBM memory benchmarking and provides a picture of HBM performance characteristics. They compare HBM to DDR4 and DDR3. They looked into refresh intervals, during which a memory transaction exhibits a significantly longer latency. Their work shows memory behavior with different memory mapping policy and localities. The total achievable memory throughput of HBM is much higher than DDR if all pseudo channels simultaneously access their associated banks, accelerating memory-intensive applications on FPGA. A Memory Access Optimizer (MAO) IP core is presented in [2]. It is used as an intermediate layer between accelerator and HBM interface. It redistributes and reorders memory transactions, and minimizes the lateral connections of Xilinx HBM interconnect so that the throughput of HBM is significantly improved.

The closest to this research is a study of the Ultrascale+ architecture [6]. However, their study does not include HBM or UltraRAM, but only looks at Block RAM vs. DDR. They quantitatively analyze memory especially DDR behaviors on xilinx Zynq board with different burst length, combinations of AXI ports, access patterns and multiplexing options. They proposed hardware design rules and optimizations for larger DDR throughput and performance. In previous work, we investigated the k-kmeans algorithm on AWS FPGA nodes [17]. This work did not consider HBM as it is not available on the FPGAs on the AWS platform.

HBM on FPGAs is used to accelerate many applications, for example [18]. This paper presents range selection, hash join, and stochastic gradient descent. They integrate the designs into a columnar database (MonetDB) and show that FPGA+HBM based solutions are able to surpass the highest performance of CPUs. Similarly, [19] proposed a hash table targeting HBM-enabled FPGAs. Their design is tailored for HBM architecture, allowing flexible mapping between processing engines and HBM channels. GraphLily [20] is a graph linear algebra overlay to accelerate graph processing on HBM-equipped FPGAs. Their implementation uses 24% BRAM and 51% URAM and the overlay frequency is 165 Mhz. ScalaBFS [21] is an accelerator for Breadth-First Search. It decouples memory accessing from processing to scale its performance with HBM memory.

While several of these studies are similar to our approach, none consider the impacts of HBM in conjunction with BRAM and URAM usage, especially when a large portion of BRAM or URAM resources are required. An exception is [22] where researchers present a sparse DNN inference engine on the HBM-enabled Alveo U280 platform, built around FPGA-optimized DSA blocks. They construct a multi-ported memory using AXI stream as a switch between URAM banks. Although one block contains small amounts of BRAM and URAM utilization, they carefully floorplan the implementation to achieve high frequency if multiple kernels are instantiated on an FPGA device. While this research uses multiple different types of memory it is carefully designed and focuses on a single application.

III. Applications

In this research we investigated two applications with different memory access patterns: (1) vector addition (from the Vitis Xilinx examples) using very large inputs, and (2) Garbled Circuits (GC) described below. Vector ADD (VADD) has streaming memory access while GC exhibits random memory access. Both VADD and GC use a monolithic memory structure. We also investigated memory access patterns for convolution, but since data access can easily be broken into blocks, it does not exhibit the same types of issues with using large memory that are observed with monolithic data structures, so we do not consider it further.

A. Vector ADD

Vector ADD (VADD) is a straightforward application. We use the version distributed with the Xilinx Vitis tools [23]. This example adds vectors of 32 bit integers; the size of the

978-1-6654-9786-2/22/$31.00 ©2022 IEEE

vector is a parameter. We vary the size of the memory in this example from 1 to 30 Megabytes in increments of 1 Megabyte.

B. Garbled Circuits

This research investigates the use of different types of memory in conjunction with Garbled Circuits(GC) accelerated with FPGAs [15], [16], [24], [25]. A complete system involves both garbler and evaluator. Here we focus on the garbler, which is the more computationally intensive part of the problem. From a memory perspective, the circuit has inputs and a representation of the circuit split into layers, and produces outputs. Most importantly, during circuit operation keys are generated in each layer that are used in subsequent layers and need to be stored. For the large problems we are targeting, this can involve over 400 megabytes of data to be stored that will need to be accessed out of order in later layers. The largest problem investigated in this research stores over 26 million keys, each of which requires 128 bits.

A GC implementation consist of a number of Garbled AND gates and Garbled XOR gates, and the surrounding logic required to fetch and store inputs and outputs and correctly sequence all operations. We refer to the gates as cores in our design. They are much more complex than typical logic gates, and have 128 bit wide input keys. A garbled XOR gate consists of 128 XOR gates in parallel, while a Garbled AND gate is composed of three AES cores as well as additional logic.

The GC design processes multiple keys in parallel, and provides multiple copies of identical cores to do the processing. Since the design is memory rather than compute bound, increasing the number of cores does not improve application performance. The amount of parallelism that can be supported depends on the amount of data that can be fetched concurrently and sent to the cores. This example motivated this study into the uses of different types of memories.

IV. Experiments and Results

Our basic experiment is to use HBM to hold data and to use on-chip memory, either BRAM or URAM as a direct mapped cache for operating on that data. Note that different applications have different memory access structures. For VADD data can be streamed in, while for GC, data access is random. In both these applications, memory is viewed as one large monolithic structure broken up into pieces to map to the on-chip resources. We observe that the achievable clock speed drops as the percentage of on-chip memory used increases. Our goal is to predict the performance of applications based on parameters such as memory access type and percentage of on-board memory usage.

We used the VADD example [23] as a baseline to determine the expected drop in clock speed based on memory usage. We conducted a range of experiments using BRAM and URAM, and measured the clock rate for VADD as the buffer size is varied from 51200 in increments of 51200 elements, or 1 Megabyte. The blue line in Fig. 2 shows the results when storing data in BRAM, and varying the buffer size from 1 to 8 Megabytes. The blue line in Fig. 3 shows how the clock speed changes when storing data in URAM, and varying the buffer size from 1 to 30 Megabytes in increments of 1 Megabyte. Note that there is more URAM available so more data can be stored. Also note that BRAM clock speeds are smoother and thus easier to predict. To predict this behavior, we applied different models and calculated the coefficient of determination. Exponential regression fit gives the highest R-squared value so we chose exponential regression to model the data. We used exponential regression to create equations (one for BRAM one for URAM) that maps percentage of available memory used to clock speed. The predictions are shown in orange in Figures 2 and 3 for BRAM and URAM respectively. We use the resulting equations to predict results for other applications, namely Garbled Circuits.

A. Experimental Setup

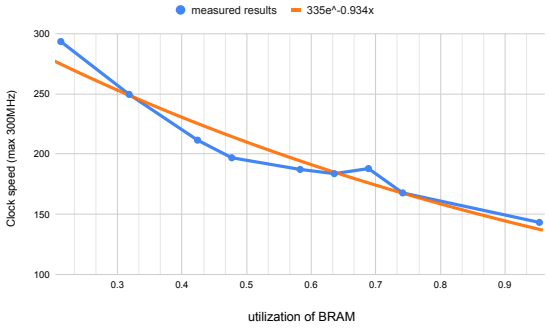

Fig. 2. BRAM Clock Speed vs. utilization

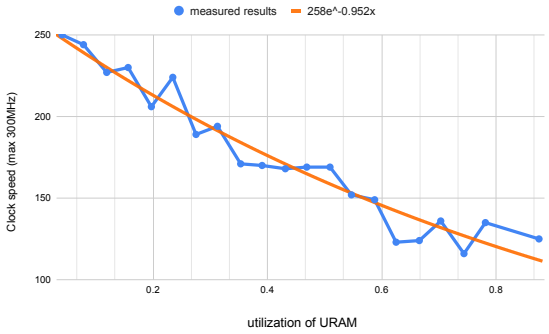

Fig. 3. URAM Clock Speed vs. Utilization.

TABLE II
Garbled Circuits design clock speeds (MHz) with URAM vs BRAM memories

Design	BRAM			URAM		
	Clock speed	estimate	difference percent	Clock speed	estimate	difference percent
HBM+50K	271	268	1.21	220.7	256	15.8
HBM+100K	238	235	1.30	189.2	246	30.0
HBM+200K	169	181	7.01	176.3	235	33.1
HBM+400K				159.3	213	33.9
HBM+800K				136.2	175	28.2

As seen in Table II for garbled circuits designs, the clock speed drops as the percentage of on-chip memory increases.

978-1-6654-9786-2/22/$31.00 ©2022 IEEE

Using the equation from VADD, we predicted the expected clock speed. BRAM clock speed predictions were better than 90% accurate for BRAM, and better than 65% accurate for URAM. In general, experiments with BRAM results in more predictable outcomes than URAM. This is due to the fact that BRAM is more distributed across the chip and thus results in less congestion. The congestion observed when using URAM translates to less predictable performance.

B. Performance

Using a lot of on chip memory along with HBM causes the clock speed to drop and we expect application performance to be correlated with clock speed. However, this may not be the case as this drop in clock speed may not result in worse overall performance. BRAM overall has a 10x faster access rate compared to HBM. This is due to the combined impacts of faster access time (1 vs 40 cycles) and lower access width (128 vs 512 bits).

The way that data is mapped to BRAM can impact overall performance. For a convolution example, we did not see a drop in clock speed when using large amounts of BRAM. As BRAM is distributed across the chip, there is not significant congestion in the design and the clock speed only drops significantly when a large percentage of memory is used. In addition, data for convolution can be easily broken into blocks and assigned to local memories.

Behavior using URAM is more challenging to predict, likely because the routing and congestion that results when using URAM is more complex. Due to pipelined access, URAM memory accesses require 2 or 3 clock cycles and the bitwidth for accessing memory is 72 bits. Thus, the speedup from accessing data from URAM vs. HBM is approximately 5 times faster than the same data being stored on HBM.

Table III shows how the overall performance in milliseconds changes for different Garbled Circuits experiments when the size of the problem and the size of the BRAM is varied. Here the application being garbled is K-means, which was chosen because it can easily be scaled in size. The size of the problem is described as `dp_cxj` where dp is the number of data points, c is the number of classes and j is the number of iterations. The largest problem we report on, 1000_4x2, has 1000 data points, with four classes and runs for two iterations. It generates more than 26 million intermediate keys. For the smaller sizes of 100_2x2 and 100_4x2 the data fits completely into BRAM. The drop in performance for larger BRAMs is a result of the clock speed drop and no other effect. When a combination of BRAM and HBM are required, the performance is more complicated. As expected, larger amounts of BRAM result in better overall performance even with a clock speed drop due to the shorter latency to access data present in the BRAM.

The clock speed drop for larger amounts of BRAM in conjunction with HBM is due to routing congestion. When 50K of BRAM is used, the FPGA design and BRAM are placed close to the HBM. In contrast, as shown in the placement diagram in Fig. 4, when 400K of BRAM is used, the BRAM (shown as yellow rectangles) is distributed across the chip. The blue in the figure shows the FPGA design, and the yellow at the bottom represents the HBM. The design with 50K of BRAM uses less than 10% of the available BRAM, while the design with 400k uses 70%. The FPGA design remains the same and consumes fewer than 10% of other resources. The distribution of BRAM across the chip results in routing congestion and a slower clock rate.

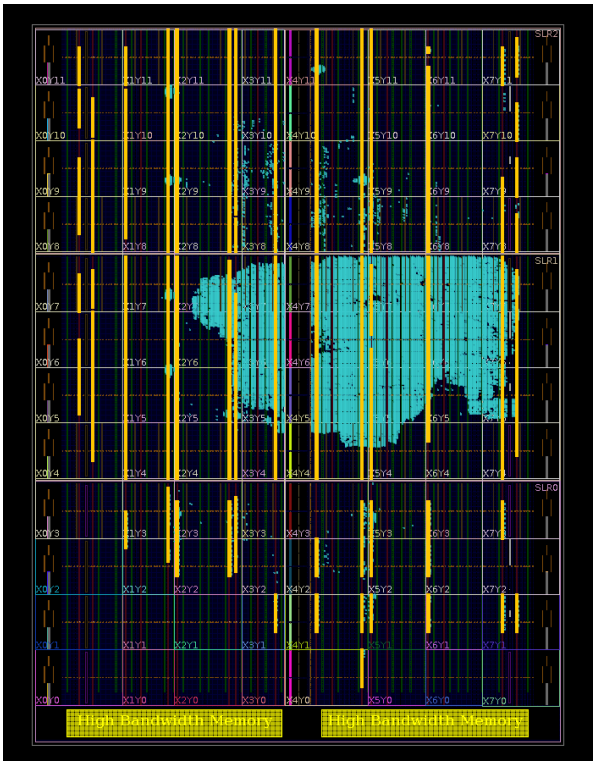

Fig. 4. Placement of GC Design with 400K BRAM

Table IV compares performance among Garbled Circuits Designs of K-means that use HBM alone, HBM plus BRAM and HBM plus BRAM plus URAM. All results are given in milliseconds of total latency. When BRAM and URAM are included, 100 KBytes of URAM and 100 KBytes of BRAM were used. Our experiments show that HBM plus BRAM deliver the best results. The conclusion from this experiment is that URAM is an excellent resource, but not effective to use in conjunction with both HBM and BRAM.

V. Discussion

High Bandwith Memory (HBM) is large memory that delivers data at high rates to processing elements. Several previous studies have investigated the use of HBM and how to take advantage of it on FPGAs. We present a study that looks at using HBM in conjunction with on-chip memory. Our results show that the clock rate drops when using a large percentage of available on chip memory. While expected clock rate for designs with large amounts of BRAM can easily be predicted, predicting URAM behavior is more challenging. This is likely

978-1-6654-9786-2/22/$31.00 ©2022 IEEE

TABLE III
APPLICATION LATENCY FOR DIFFERENT BRAM SIZES WITH THE SAME DESIGN

K-Mans	HBM + 50k BRAM (ms)	HBM + 100k BRAM (ms)	HBM + 200k BRAM (ms)
100_2x2	103	125	109
100_4x2	198	244	218
100_8x2	389	575	462
1000_2x2	1382	1603	1468
1000_4x2	3199	3338	3138

TABLE IV
LATENCY OF DESIGNS WITH DIFFERENT TYPES OF MEMORY

K-Means	HBM,BRAM,URAM(ms)	HBM(ms)	HBM,BRAM(ms)
100_2x2	178	181	125
100_4x2	356	361	244
100_8x2	706	724	575
1000_2x2	2053	1806	1603
1000_4x2	4184	3543	3338

due to the fact that URAM, while larger than BRAM, is concentrated in a few locations and thus results in more congested routing. In addition, the type of memory access matters. Designs whose memory can easily be partitioned into smaller blocks, such as convolution, do not see the same clock speed drop as exhibited by designs where data is accessed from one large memory space.

In addition, a clock speed drop does not necessarily result in reduced performance. This is due to the fact that data stored in on-chip memories can be accessed with lower latency than HBM. As a result, performance of designs with lower clock speed may result in better overall application performance.

When comparing usage of all three types of memory together (HBM, BRAM, and URAM) our experiments show that the best results are achieved when combining HBM and BRAM. We continue to develop models to capture this behavior in order to provide guidance to the designer. We use Garbled Circuits as an application. GC has characteristics we expect to see in other data center applications, where users want to accelerate their designs without spending a large amount of time optimizing the layout. We will continue to investigate how the models we develop apply to other applications.

VI. CONCLUSIONS

The goal of this research is to predict the expected behavior of big data applications from parameters of those applications. The research presented here represents a first step in that direction. Namely, we can predict the drop in clock speed based on the size of BRAM or URAM used. We are working on a tool that will predict performance based on size of on-chip memory, number of data accesses, and type of memory access and make recommendations to the user regarding the amount and types of memories that they should use. High Level Synthesis makes it difficult to control the different types of memory usage directly. We plan to feed our predictions into a tool to help users specify different memory types for their application.

REFERENCES

[1] Xilinx, "Alveo U280 Data Center Accelerator Card." [Online]. Available: https://www.xilinx.com/products/boards-and-kits/alveo/u280.html

[2] P. Holzinger, D. Reiser, T. Hahn, and M. Reichenbach, "Fast HBM Access with FPGAs: Analysis, Architectures, and Applications," in *2021 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, Jun. 2021, pp. 152–159.

[3] Z. Wang, H. Huang, J. Zhang, and G. Alonso, "Shuhai: Benchmarking High Bandwidth Memory On FPGAS," in *2020 IEEE 28th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. Fayetteville, AR, USA: IEEE, May 2020, pp. 111–119. [Online]. Available: https://ieeexplore.ieee.org/document/9114755/

[4] K. Kara, C. Hagleitner, D. Diamantopoulos, D. Syrivelis, and G. Alonso, "High Bandwidth Memory on FPGAs: A Data Analytics Perspective," *arXiv:2004.01635 [cs]*, Apr. 2020, arXiv: 2004.01635. [Online]. Available: http://arxiv.org/abs/2004.01635

[5] Y.-k. Choi, Y. Chi, W. Qiao, N. Samardzic, and J. Cong, "HBM Connect: High-Performance HLS Interconnect for FPGA HBM," in *The 2021 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. Virtual Event USA: ACM, Feb. 2021, pp. 116–126. [Online]. Available: https://dl.acm.org/doi/10.1145/3431920.3439301

[6] K. Manev, A. Vaishnav, and D. Koch, "Unexpected Diversity: Quantitative Memory Analysis for Zynq UltraScale+ Systems," in *2019 International Conference on Field-Programmable Technology (ICFPT)*. Tianjin, China: IEEE, Dec. 2019, pp. 179–187. [Online]. Available: https://ieeexplore.ieee.org/document/8977835/

[7] Xilinx, "UltraScale Architecture Memory Resources User Guide," March 2021. [Online]. Available: https://www.xilinx.com/support/documentation/user_guides/ug573-ultrascalememoryresources.pdf

[8] A. Yao, "How to generate and exchange secrets," in *IEEE Symposium on Foundations of Computer Science (FOCS)*, 1986.

[9] M. Naor, B. Pinkas, and R. Sumner, "Privacy preserving auctions and mechanism design," in *1st ACM Conference on Electronic Commerce*, 1999.

[10] V. Nikolaenko, S. Ioannidis, U. Weinsberg, M. Joye, N. Taft, and D. Boneh, "Privacy-preserving matrix factorization," in *ACM CCS*, 2013.

[11] D. Beaver, S. Micali, and P. Rogaway, "The round complexity of secure protocols," in *Proceedings of the twenty-second annual ACM symposium on Theory of computing*. ACM, 1990, pp. 503–513.

[12] B. Pinkas, T. Schneider, N. P. Smart, and S. C. Williams, "Secure two-party computation is practical," in *ASIACRYPT*, 2009.

[13] V. Kolesnikov and T. Schneider, "Improved Garbled Circuit: Free XOR Gates and Applications," in *ICALP*, 2008.

[14] M. Naor and B. Pinkas, "Efficient oblivious transfer protocols," in *Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms*. Society for Industrial and Applied Mathematics, 2001, pp. 448–457.

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

[16] X. Fang, S. Ioannidis, and M. Leeser, "SIFO: Secure Computational Infrastructure Using FPGA Overlays," *International Journal of Reconfigurable Computing*, vol. 2019, pp. 1–18, Dec. 2019. [Online]. Available: https://www.hindawi.com/journals/ijrc/2019/1439763/

[17] K. Huang, M. Gungor, S. Ioannidis, and M. Leeser, "Optimizing use of different types of memory for fpgas in high performance computing," in *2020 IEEE High Performance Extreme Computing Conference (HPEC)*. IEEE, 2020, pp. 1–7.

978-1-6654-9786-2/22/$31.00 ©2022 IEEE

[18] K. Kara, C. Hagleitner, D. Diamantopoulos, D. Syrivelis, and G. Alonso, "High bandwidth memory on fpgas: A data analytics perspective," in *2020 30th International Conference on Field-Programmable Logic and Applications (FPL)*, 2020, pp. 1–8.

[19] Y. Yang, S. R. Kuppannagari, and V. K. Prasanna, "A high throughput parallel hash table accelerator on hbm-enabled fpgas," in *2020 International Conference on Field-Programmable Technology (ICFPT)*, 2020, pp. 148–153.

[20] Y. Hu, Y. Du, E. Ustun, and Z. Zhang, "Graphlily: Accelerating graph linear algebra on hbm-equipped fpgas," in *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, 2021, pp. 1–9.

[21] C. Liu, Z. Shao, K. Li, M. Wu, J. Chen, R. Li, X. Liao, and H. Jin, "Scalabfs: A scalable bfs accelerator on fpga-hbm platform," in *The 2021 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 147. [Online]. Available: https://doi.org/10.1145/3431920.3439463

[22] A. K. Jain, S. Kumar, A. Tripathi, and D. Gaitonde, "Sparse deep neural network acceleration on hbm-enabled fpga platform," in *2021 IEEE High Performance Extreme Computing Conference (HPEC)*, 2021, pp. 1–7.

[23] Xilinx, "Xilinx vitis tutorials: VADD," 2022. [Online]. Available: https://github.com/Xilinx/Vitis-Tutorials/blob/2022.1/Getting_Started/Vitis/example/src/vadd.cpp

[24] M. Bellare, V. T. Hoang, and P. Rogaway, "Foundations of garbled circuits," in *Proceedings of the 2012 ACM conference on Computer and communications security - CCS '12*. Raleigh, North Carolina, USA: ACM Press, 2012, p. 784. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2382196.2382279

[25] E. M. Songhori, S. U. Hussain, A.-R. Sadeghi, T. Schneider, and F. Koushanfar, "TinyGarble: Highly Compressed and Scalable Sequential Garbled Circuits," in *2015 IEEE Symposium on Security and Privacy*. San Jose, CA: IEEE, May 2015, pp. 411–428. [Online]. Available: https://ieeexplore.ieee.org/document/7163039/

978-1-6654-9786-2/22/$31.00 ©2022 IEEE