

Scheduling Memory Access Optimization for HBM Based on CLOS

Shuang Xue^{*†}, Huawei Liang^{*†}, Qizhe Wu^{*†} and Xi Jin^{*†}

^{*}State Key Laboratory of Particle Detection and Electronics, University of Science and Technology of China, Hefei, China

[†]Institute of Microelectronics, Department of Physics, University of Science and Technology of China, Hefei, China

Emails: {xs1159, lhw233, wqz1998}@mail.ustc.edu.cn, jinxi@ustc.edu.cn

Abstract—With the recent release of FPGA boards based on High Bandwidth Memory (HBM), developers could employ unparalleled external memory bandwidth. HBM provides large-scale aggregated memory bandwidth by exposing multiple memory channels to the processing unit. This allows more memory-constrained applications to benefit from FPGA acceleration. However, it is difficult to take full advantage of available bandwidth: when an application requires multiple processing elements to access multiple HBM channels, the limited number of horizontal connections of the built in crossbar in HBM can result in a significant reduction in effective bandwidth for global addressing. To solve this problem, we propose HBM connection, which is a high-performance custom interconnection for FPGA HBM board. The high-performance custom switching network based on CLOS is introduced to replace the built in crossbar to optimize HBM access scheduling, and increase throughput of AXI bus hosts and switching components. The validity of HBM connection is proved by Xilinx VCU128 HBM board. Based on the breadth-first search BFS case study, we conduct an experimental exploration. The result shows that HBM connection improves the effective performance by 2.5X.

Keywords—HBM , CLOS , access schedule , FPGA , BFS

I. INTRODUCTION

The growing demand for computational power has recently reached new heights with the widespread adoption of deep learning and big data technologies[1]. Performing advanced data analytics efficiently in data centers is becoming increasingly important, largely due to the exponential growth in the amount of data to process and the rise of machine learning. However, as Moore's Law and the Dennard scaling continue to slow down, these requirements are increasingly difficult to meet.

In recent years[3], FPGA have been widely used in various data processing tasks. For the accelerator of throughput optimization implemented on FPGA, high bandwidth access to data is crucial. One of the main bottlenecks in current FPGA design is the bandwidth to access the data to be consumed or generated. However, the relatively slow development of DRAM devices leads to stagnant performance of general-purpose processors, which is especially pronounced for FPGA because their external memory access is much slower than that of GPUs [3]. The resulting processor memory gap has been a major challenge for more than 20 years [4]. Two different memory architectures for high throughput applications, such as graphics cards and AI, are GDDR and HBM.

GDDR is memory for the graphics card, which is double rate synchronous dynamic random access memory. AMD believes GDDR5 won't be able to keep up with GPU performance growth rates, and that GDDR5's rising power consumption may soon be large enough to stop graphics performance growth. GDDR5, by contrast, requires more chip and circuit voltages to achieve high bandwidth.

High bandwidth memory is a CPU/GPU memory chip ("RAM"), which is many DDR chips stacked together and then packaged together with GPU to achieve a large-capacity, high-width DDR composite array. The HBM stack is not physically integrated with the CPU or GPU, but is tightly and quickly connected through the intermediary layer. The HBM has almost the same characteristics as the chip integrated RAM, and therefore, has higher speed and bandwidth. Recent advances in 3D chip stacking and packaging have made it possible to significantly improve memory throughput [5]. HBM is a 3D die-stacking memory that employs Through-Silicon Via (TSV) technology to stack DRAM(DDR SDRAM) vertically. HBM is package-level memory based on a wide input/output (I/O) parallel interface. In particular, HBM provides a terabyte of bandwidth per second, 1024 data I/Os on a finely spaced silicon meson. In addition, HBM radically reduces I/Os energy by reducing channel length through on-package integration. Because of these merits of HBM, It was recently used by several application-specific accelerators and showed promising performance [6]-[8].

Compared with traditional DRAM, HBM creates high bandwidth and low frequency video memory, which does not need high frequency on the basis of providing relatively large video memory bit width. For example, under the same 4GB capacity, HBM can provide 4096 bit video memory width, 8 times higher than 512bit of GDDR5. The low operating frequency makes the HBM 3 times more powerful per watt than the GDDR5. Therefore, compared with traditional memory, HBM has the merits of higher bandwidth and lower power consumption. HBM, however, typically possesses a lower clock, use a wider bus and more channels, which requires a higher degree of parallelization of memory access. In order to facilitate the use of HBM a global addressing crossover switch is implemented on FPGA, which greatly simplifies the system design. However, depending on their usage, these bus architectures can also introduce serious limitations in throughput and latency [8]. Therefore, when switching from

conventional DRAM to HBM, one cannot assume that the design is linearly proportional to the theoretical bandwidth. This makes HBM equipment often more difficult to handle and requires architectural adjustments to the accelerator. The large external memory bandwidth of HBM is derived from multiple independent HBM channels (Fig.1)[14]. To take full advantage of this architecture, we needed to determine the most efficient way to transfer data from multiple HBM channels to multiple processing elements (PEs). Therefore, the performance of FPGA HBM board is greatly affected by bus burst access. HBM interconnect architecture implements segmented switching networks to achieve global addressing, and horizontal connections route AXI transactions to targets when addresses do not belong to local switches. However, only two output buses are available in each direction. While this subsection enables the desired global addressing, it alters the latency of memory access, and the limited number of horizontal connections effectively reduces bandwidth if requests need to travel on the same bus.

Therefore, this paper proposes a high-performance custom interconnection for FPGA HBM boards to optimize HBM access scheduling. We firstly discuss the performance of HBM's built in crossbar and horizontal connections and analyze the reason for the bandwidth decrease when PEs accesses multiple PCHs. Then, we propose an interconnect solution to replace built in crossbar, that increases the effective bandwidth of HBM and eliminate performance bottlenecks for horizontal connections and improve HBM performance.

The contributions of this paper can be summarized as follows:

- An efficient and scalable HBM interconnection architecture designed based on cros network.
- RTL based solutions to achieve customized crossbar , and HBM access scheduling optimization.
- Evaluate the customized crossbar system on the VCU128 and analyze its performance.

The structure of this paper is as follows: The second section takes Xilinx device as an example to explain the architecture and problems of HBM2 memory subsystem. Then, the previous work is discussed in section III. Section IV provides a more comprehensive analysis of memory characteristics and the necessary architectural considerations. On this basis, our approach is evaluated with two sample designs in Section V. Finally, section VI summarizes this paper.

II. BACKGROUND

A. High Bandwidth Memory2

High Bandwidth Memory [5][17] is a 3D-stacked DRAM designed to provide a high memory bandwidth. There are 2 8 HBM dies and 1024 data I/Os in each stack. The HBM DRAM dies are connected to a base logic die using Through Silicon Via (TSV) technology. The base logic die connects to FPGA/GPU/CPU dies through an interposer. The maximum I/O data rate is improved from 1 Gbps in HBM1 to 2 Gbps in HBM2.

As a typical example, Fig.1 illustrates the HBM subsystem of Xilinx Alveo U280 [12]. The HBM subsystem contains two HBM stacks, each of which is divided into 16 pseudo channels (PCHs). As each PCH provides 2Gbit storage capacity, 32 HBM PCHs provide 8GB ($2 \times 16 \times 2 / 8$) storage capacity totally. Above the 32 PCHs, there are 16 memory channels (MCs), each of which controls the accesses towards two adjacent PCHs. The switch network is composed of 8 4x4 mini-switches, each of which connects two adjacent MCs and exposes 4 AXI interfaces to the FPGA. Thus, the HBM subsystem of U280 exposes 32 AXI ports totally to the user logic built in the FPGA. Every two adjacent mini-switches are connected with a bus to provide global addressing, such that a memory access towards an arbitrary PC can be issued from any AXI interface.

By conducting random accessing workloads, Shuhai [18] measured the performance of the HBM subsystem of U280, and observed that:

- A memory access to HBM suffers higher latency than that towards DDR4.
- Although the bandwidth of a single HBM PCH is smaller than DDR4, the aggregated bandwidth of HBM arrives at 425 GB/s when handling sequential accesses.

B. HBM2 FPGA Platforms and Bulit_in Crossbar

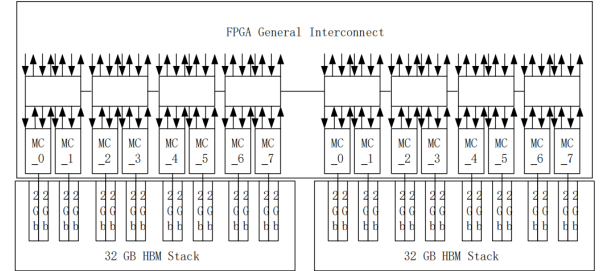


Fig. 1. Structure of HBM interfaces on the example of Xilinx Virtex UltraScale+ FPGAs

As depicted in Fig.1, Xilinx devices use two 4-Hi stacks with 4 GB capacity each. This results in a total of 32 PCH which are presented to the programmable logic (PL) as just as many 256 bit AXI3 busses with half the frequency. Hereby, every two share a common memory controller (MC) to perform the AXI to DDR protocol conversion. The global addressing interconnect structure is implemented as a segmented switch network in which every four AXI slaves and two MCs are directly connected by a local crossbar switch. In contrast to Intel, these are part of the FPGA and contain lateral connections to route AXI transactions towards the destination when the address does not belong to the local switch. However, only two outgoing busses are available in every direction. Hereby, the memory capacity of every PCH is contiguously mapped into successive sections of the global address space.

Although this kind of segmentation enables the desired global addressing, it has several major drawbacks. First, it

changes the latency of memory accesses since AXI transactions potentially need to be routed over several hops. This non-uniformity might be an issue for accelerators which expect data to arrive approximately at the same time. Second, the limited number of lateral connections effectively reduces the bandwidth if requests need to be routed over the same bus. Third, If this data is simply copied to HBM with such an address layout, it would be placed in the same PCH until its maximum capacity is reached. This potentially causes Bus Masters to contend for the same PCH and therefore severely decreases latency and throughput. Overall these issues can lead to worse results than with traditional DRAM due to the lower clock frequency. These problems need to be considered when architectures for HBM are drafted.

A Write sequence with near 100% MC throughput results in the largest drop in throughput due to the extra switching cycle. As a result, smaller blocks have a larger percentage loss than larger blocks. The following figure demonstrates the lateral throughput loss with 32-byte and 256-byte transactions. For the 32-byte transactions two masters are each issuing three 32-byte bursts which must laterally traverse the AXI switch. For the 256-byte transactions each master is issuing two 256-byte bursts which must laterally traverse the AXI switch. In the 32-byte case there is one clock inserted between each data burst for each master which results in a total of 12 clock cycles to move 6 beats of data. In the 256-byte case there is one idle clock inserted between each 256-byte burst which results in a total of 36 clock cycles to move 32 beats of data. Because two adjacent pseudo-channel ports such as M0 and M1 only have access to one lateral slave port, the total bandwidth for these ports is split between the two masters. This results in a total efficiency for the 32-byte transactions of about 25% due to the additional clock cycles for switching between masters as well as both masters sharing a single slave port. For the 256-byte transactions these same behaviors result in approximately 44.4% efficiency[14]. As Table I shows.

TABLE I
MEASURED LATERAL THROUGHPUT EFFICIENCY BASED ON 100% PAGE
HIT RATE WRITE STREAMS FOR 4H STACK

Block Size	Switch Limited BW Pct
32B	24.9%
64B	33.3%
128B	39.9%
256B	44.4%
512B	47%

Therefore, in order to optimize HBM access scheduling, we propose a custom crossbar to replace HBM's Built_in crossbar. Resolving the problem of the limited number of lateral connections and try to make data arrive at the same time to improve the uniformity.

In recent years, several articles have proposed their own solutions to the above HBM problems, such as HBM Connect[9] and Memory Access Optimizer (MAO) IP core[1]. HBM Connect uses a selective structure based on the butterfly switching network which could achieve higher effective bandwidth due

to this characteristics. The butterfly network reduces its hop distance at the later stages of the custom crossbar. But it is difficult to use in the case of multiple HBMs. For MAO IP core, its paper[1] does not introduce its specific structure, so it would not be discussed.

III. CASE STUDY

In order to quantify the resource consumption and the performance of HBM interconnect when PEs access multiple PCs, we select one applications for case studies: BFS. For BFS, each PE need to read from multiple PCHs, and each PCH also need to send messages to multiple PEs.

We apply HBM's access scheduling optimization to parallel implementation of breadth-first search. Firstly we processed the required graph and stored it in a CSR format. For input graphs, we classify data into two types: vertex data and graph data. Vertex data refers to the level array, and graph data refers to the neighbor list. We use precious on-chip memory resources (BRAM and URAM) to store all vertex data, and graphic data is stored in HBM for easy access in parallel. In order to maximize parallel computing to match HBM. We divide the graph data into multiple subgraphs and place each subgraph in HBM PCH. The IDs are divided into multiple intervals for load balancing reasons.

We need to divide the vertex ID space of the sample graph into two PEs: firstly, for load balancing reasons, the IDs are divided into two intervals, such as [0,2,4] and [1,3,5]. The graph data then is partitioned according to the partition results on the vertex ID space: the neighbor lists of vertices belonging to the same partition are placed in the same subgraph. The reason for this horizontal partitioning scheme is that it does not decompose the neighbor list, and longer neighbor lists mean more opportunities for sequential access to HBM, which helps increase memory bandwidth utilization during processing.

When traversing to a certain vertex, the corresponding neighbor list would be read from HBM's PCH, and then the neighbor node would be sent to the corresponding PE unit after the selection network. In PE unit, first thing is checking the node whether it has been traversed. If not, it would enter FIFO and become the next node to be traversed, otherwise it would be abandoned. Loop until all neighbor nodes have been traversed. The choice of network in this process is the focus of our attention, and we use customed crossbar to avoid the use of HBM's built_in crossbar, which would send nodes to the corresponding interface to avoid the bandwidth reduction caused by data collisions.

IV. CUSTOM CROSSBAR

A. Network Topology

As is mentioned above, it is not possible to reach the maximum bandwidth when an AXI master tries to access multiple PCHs. To reduce the contention, we add a custom crossbar.

There are several multi-level network topologies, including Omega, Clos, Benes, and butterfly networks. Among them, we choose Clos network because of the scalability and fixed

switching delay of Clos networks, and the orderiness of data transmission. However, if more than one cell reaches the same switching unit at the same time, a collision occurs. In order to deal with collision conflicts, we design Clos switching structure, by which the data is transmitted on a fixed path, although the self-routing of Clos switching structure is sacrificed in this way, the data can be orderly transmitted to the corresponding port without collision. The switching network can pre-allocate data to its corresponding switch according to the corresponding policy to avoid the use of HBM built_in crossbar and thus improve bandwidth and allow data to arrive simultaneously. As is shown in Fig.2 .We can consider a 16×16 Clos network design as a unit, and it also can be extended to 32×32 , 64×64 and so on.

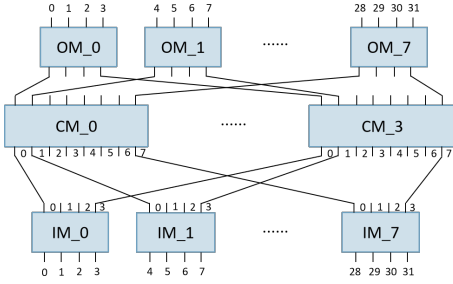


Fig. 2. Clos switching network

The three-stage Clos switching network is composed of input, intermediate and output stage. There is only one link between modules at each level. The input stage consists of $K \times N \times M$ crossbars, where N represents the number of input ports in each input module and M represents the number of output ports in each input module. The intermediate stage is composed of M crossbars of $K \times K$. K represents the number of input and output ports of the intermediate stage. The output stage consists of $K \times M \times N$ crossbars, where M represents the number of input ports in each output module and N represents the number of output ports in each output module.

B. Connect

TABLE II
ORTHOGONAL LINK SELECTION

slots/IM#0	0	1	2	3
subslot1	OM#0	OM#1	OM#2	OM#3
subslot2	OM#4	OM#5	OM#6	OM#7
slots/IM#1	0	1	2	3
subslot1	OM#1	OM#2	OM#3	OM#0
subslot2	OM#5	OM#6	OM#7	OM#4
slots/IM#2	0	1	2	3
subslot1	OM#2	OM#3	OM#0	OM#1
subslot2	OM#6	OM#7	OM#4	OM#5
slots/IM#3	0	1	2	3
subslot1	OM#3	OM#0	OM#1	OM#2
subslot2	OM#7	OM#4	OM#5	OM#6

In order to avoid collision during data transmission in Clos network, an orthogonal link selection is adopted. To prevent data collision during transmission, IM cannot send

data through the same CM to the same OM at the same time. So we fixed the paths from IM to CM and from CM to OM. The port of each IM is sent to its destination OM via a different CM; CM would send data received from IM to the corresponding OM, so we just need to make sure that different IMs send data to different OM through the same CM at the same time. Since a single HBM stack has 16 PCHs, we set the input and output ports of IM and OM as 4 and the number of CM as 4, and the port of CM would change with the number of HBM stack. As shown in the fig.2, each IM port corresponds to a different OM, so data collisions would not occur. For multiple HBM stacks, we employ the connection of a single HBM stack as a base unit and divide a time slot into multiple time subslots (the number of HBM stacks applied). Therefore, data output by the IM output port needs to be sent to the corresponding OM unit in the corresponding time subslot according to the policy. For example, for 2 HBM stacks, time slot is divided into 2 subslots, the 4th output ports of IM_0 are connected to the first input port of 4th CM, and then connected to OM. Data from the first output port of IM is sent from CM_0 to OM_0 in time subslot 1, and from CM_0 to OM_4 in time subslot 2, and so on. As is shown in Table II .

C. Module

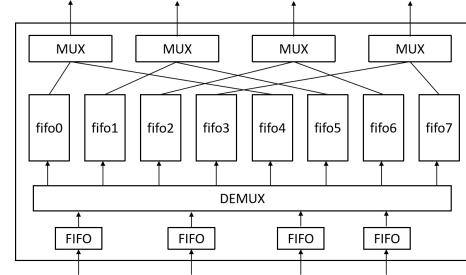


Fig. 3. Structure of IM

1) *IM*: As shown in the Fig.3, each input level unit is composed of five parts. The first part is FIFO, this part is used to store the transmitted data in a burst to avoid the loss or invalidity of transmitted data; the second part is the data distributors(demux) connected to the input ports,through which the input data is sent to the corresponding FIFOs; the third part is the rrarbiters connecting between data distributor and FIFO, which prevents the different data distributor distributed data into the same FIFO at the same time, avoiding the data collision conflict; the fourth part is FIFO corresponding to the output level unit, and there would be a corresponding FIFO in IM for each OM unit; the fifth part is the mux connected to the output port of FIFO and the output interface of the input level unit,respectively, which send FIFO datas to the corresponding IM output port according to the time. The IM output port is connected to the corresponding CM input port.

2) *CM*: As shown in the Fig.4, the intermediate level unit is composed of two parts. The first part is the data allocator, which selects data and sends datas to the corresponding data selector according to the time subslot. The second part is the

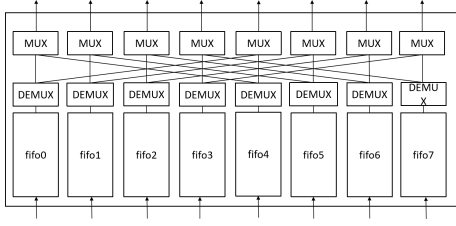


Fig. 4. Structure of CM

data selector, which transmits the data received from the data distributor to the corresponding CM output port. The input port of CM is connected to the output port of corresponding IM, and the output port of CM is connected to the input port of corresponding OM. CM's function is used to send the data received from IM to the corresponding OM unit according to the different time. As each CM is connected to OMs, data from IM can be transferred to the corresponding OM cells as needed, and the transmission path is thus selected by time subslot, which does not lead to data collision.

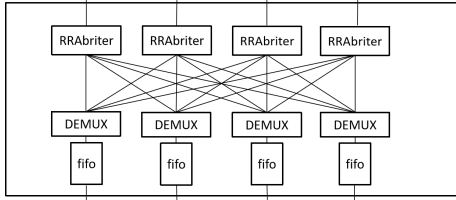


Fig. 5. Structure of OM

3) *OM*: The output level unit is similar to the input level unit, consisting of data distributor, rrarbiter and FIFO. As shown in the Fig.5, The OM allocates the received data to corresponding ports. After IM and CM selection, data that in the range of OM would be sent to the corresponding OM output ports. Therefore, it is very convenient to allocate the received data to the corresponding OM output port to complete the task.

V. EXPERIMENTAL RESULT

TABLE III
FPGA RESOURCE UNIT CONSUMPTION OF HBM INTERCONNECT

	LUT	FF	DSP	BRAM
HBM clos interconnect	36k	35k	0	390
HBM Connect[9]	87k	62k	0	248

We realize parallel BFS algorithm on VCU128 development board by employing HBM and memory access scheduling module. For programming, we utilize Xilinx's Vitis[16][21] and Vivado HLS[22] 2020.2 tools. There is another interconnect named HBM Connect[9], which use butterfly network as architecture. The Table III shows the resource consumption of HBM interconnect module on FPGA[15], we consume approximately 35k LUT, 35k FIFO, and 390 BRAM for a single 16×16 HBM clos interconnect. The Table IV shows

the effective bandwidth of HBM built_in interconnect module, the effective bandwidth of HBM clos interconnect module and HBM Connect[9] on FPGA. Compared HBM Connect[9], the effective bandwidth of the clos interconnect is lower, but it uses less resources and it has scalability, which could be expanded to use more HBMs easily due to the orthogonal link selection. According to the Table I, in theory the bandwidth of the clos interconnect should be two to four times as efficient as HBM built_in interconnect. The experimental results show that the effective bandwidth of the HBM clos interconnect is more than twice that of the HBM built_in interconnect, which is in line with expectations.

TABLE IV
EFFECTIVE BANDWIDTH OF HBM INTERCONNECT

	KClk(MHz)	EffBW(GB/s)
HBM built_in interconnect	277	65
HBM clos interconnect	300	170
HBM Connect[9]	207	203

VI. CONCLUSION

In this paper, we propose a scheduling memory access architecture based on clos interconnect network, and analyze the influence of HBM built_in interconnect structure on bandwidth. A experiment is done to test its performance. The experimental results show that proposed scheme can be implemented and effectively reduce the influence of HBM internal interconnection on bandwidth. We believe the scheduling memory access architecture will be improved in the future, so that the bandwidth of HBM can be used more effectively in parallel algorithm calculation.

REFERENCES

- [1] P. Holzinger, D. Reiser, T. Hahn and M. Reichenbach, "Fast HBM Access with FPGAs: Analysis, Architectures, and Applications," 2021 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), 2021, pp. 152-159.
- [2] Wissolik, Mike and Zacher, Darren and Torza, Anthony and Day, Brandon, "Virtex UltraScale+ HBM FPGA: A Revolutionary Increase in Memory Performance" Jul. 2019. [Online]. Available: https://www.xilinx.com/support/documentation/white_papers/wp485-hbm.pdf
- [3] J. Cong, Z. Fang, M. Lo, H. Wang, J. Xu, and S. Zhang, "Understanding Performance Differences of FPGAs and GPUs," in IEEE 26th Annu. Int. Symp. Field-Prog. Custom Comput. Mach. (FCCM), 2018, pp. 93-96.
- [4] M. V. Wilkes, "The Memory Gap and the Future of High Performance Memories," SIGARCH Comput. Archit. News, vol. 29, no. 1, p. 27, Mar. 2001.
- [5] H. Jun, J. Cho, K. Lee, H. Son, K. Kim, H. Jin, and K. Kim, "HBM (High Bandwidth Memory) DRAM Technology and Architecture," in IEEE Int. Memory Workshop (IMW), May 2017.
- [6] G. Singh, D. Diamantopoulos, C. Hagleitner, J. Gomez-Luna, S. Stuijk, O. Mutlu, and H. Corporaal, "NERO: A Near High-Bandwidth Memory Stencil Accelerator for Weather Prediction Modeling," in 30th Int. Conf. Field-Prog. Log. and Applicat. (FPL), 2020, pp. 9-17.
- [7] R. Ben Abdelhamid and Y. Yamaguchi, "A Block-Based Systolic Array on an HBM2 FPGA for DNA Sequence Alignment," in Appl. Reconfigurable Comput. Archit., Tools, and Applicat., F. Rincón, J. Barba, H. K. H. So, P. Diniz, and J. Caba, Eds. Cham: Springer International Publishing, 2020, pp. 298-313.
- [8] K. Kara, C. Hagleitner, D. Diamantopoulos, D. Syrivelis, and G. Alonso, "High Bandwidth Memory on FPGAs: A Data Analytics Perspective," in 2020 30th Int. Conf. Field-Prog. Log. and Applicat. (FPL), 2020.

- [9] Choi, Y.-k., et al. (2021). HBM Connect: High-Performance HLS Interconnect for FPGA HBM. The 2021 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays. Virtual Event, USA, Association for Computing Machinery: 116–126.
- [10] M. Guoqiang, H. Rui, W. Jiangwei, K. Hongwei and L. Rengang, "A FPGA based intra-parallel architecture for PageRank graph processing," 2020 IEEE International Conference on Edge Computing (EDGE), 2020, pp. 31-38.
- [11] Intel. 2020. High Bandwidth Memory (HBM2) Interface Intel FPGA IP User Guide. <https://www.intel.com/>
- [12] Xilinx. 2020. Alveo U280 Data Center Accelerator Card User Guide. https://www.xilinx.com/support/documentation/boards_and_kits/accelerator-cards/ug1314-u280-reconfig-accel.pdf
- [13] Xilinx. 2020. Alveo U50 Data Center Accelerator Card User Guide. https://www.xilinx.com/support/documentation/boards_and_kits/accelerator-cards/ug1371-u50-reconfig-accel.pdf
- [14] Xilinx. 2020. AXI High Bandwidth Memory Controller v1.0. https://www.xilinx.com/support/documentation/ip_documentation/hbm/v1_0/pg276-axi-hbm.pdf
- [15] Xilinx. 2020. UltraScale Architecture Memory Resources (UG573). <https://www.xilinx.com/>
- [16] R. Li, H. Huang, Z. Wang, Z. Shao, X. Liao, and H. Jin. 2020. Optimizing memory performance of Xilinx FPGAs under Vitis. ArXiv Preprint (2020). <https://arxiv.org/abs/2010.08916>
- [17] JEDEC. 2020. High Bandwidth Memory (HBM) DRAM. <https://www.jedec.org/standards-documents/docs/jesd235a>
- [18] Y. Choi, Y. Chi, J. Wang, L. Guo, and J. Cong. 2020. When HLS meets FPGA HBM: Benchmarking and bandwidth optimization. ArXiv Preprint (2020). <https://arxiv.org/abs/2010.06075>
- [19] Z. Wang, H. Huang, J. Zhang, and G. Alonso. 2020. Shuhai: Benchmarking High Bandwidth Memory on FPGAs. In IEEE Ann. Int. Symp. Field-Programmable Custom Computing Machines.
- [20] J. Cong, Z. Fang, M. Lo, H. Wang, J. Xu, and S. Zhang. 2018. Understanding performance differences of FPGAs and GPUs. In IEEE Ann. Int. Symp. Field-Programmable Custom Computing Machines. 93–96
- [21] Xilinx. 2020. Vitis Unified Software Platform. <https://www.xilinx.com/products/design-tools/vitis/vitis-platform.html>
- [22] Xilinx. 2020. Vivado High-level Synthesis (UG902). <https://www.xilinx.com/>
- [23] ARM. 2011. AMBA AXI and ACE Protocol Specification AXI3, AXI4, and AXI4-Lite, ACE and ACE-Lite. www.arm.com



Shuang Xue received his B.S. degree in 2020 from University of Science and Technology of China, and he is currently a Master student in Department of Physics in University of Science and Technology of China, Anhui, China, under the supervision of Prof. Xi Jin. His current research work os mainly on FPGA-based Hardware

Accelerator design.



Haiwen Liang received his B.S. degree in 2020 from University of Science and Technology of China, and he is currently a Ph.D. student in Department of Physics in University of Science and Technology of China, Anhui, China, under the supervision of Prof. Xi Jin. His current research work os mainly on FPGA-based Hardware Accelerator design.



Qizhe Wu received his B.S. degree in 2020 from Zhejiang University of Technology, China, and he is currently a Ph.D. student in Department of Physics in University of Science and Technology of China, Anhui, China, under the supervision of Prof. Xi Jin. His current research work os mainly on FPGA-based Hardware Accelerator design.



Xi Jin received his B.S. degree from University of Science and Technology of China, and he is currently an associate professor in Department of Physics in University of Science and Technology of China, Anhui, China, under the supervision of Prof. Xi Jin. His research interests include SOC design technology, VLSI design, computer-aided design methodologies for SoC system integration and FPGA-based Hardware structure design.