Medusa: A Scalable Interconnect for Many-Port DNN Accelerators and Wide DRAM Controller Interfaces

Yongming Shen
Stony Brook University
yoshen@cs.stonybrook.edu

Tianchu Ji
Stony Brook University
tianchu.ji@stonybrook.edu

Michael Ferdman
Stony Brook University
mferdman@cs.stonybrook.edu

Peter Milder
Stony Brook University
peter.milder@stonybrook.edu

***Abstract*—To cope with the increasing demand and computational intensity of deep neural networks (DNNs), industry and academia have turned to accelerator technologies. In particular, FPGAs have been shown to provide a good balance between performance and energy efficiency for accelerating DNNs. While significant research has focused on how to build efficient layer processors, the computational building blocks of DNN accelerators, relatively little attention has been paid to the on-chip interconnects that sit between the layer processors and the FPGA's DRAM controller.**

We observe a disparity between DNN accelerator interfaces, which tend to comprise many narrow ports, and FPGA DRAM controller interfaces, which tend to be wide buses. This mismatch causes traditional interconnects to consume significant FPGA resources. To address this problem, we designed Medusa: an optimized FPGA memory interconnect which transposes data in the interconnect fabric, tailoring the interconnect to the needs of DNN layer processors. Compared to a traditional FPGA interconnect, our design can reduce LUT and FF use by 4.7x and 6.0x, and improves frequency by 1.8x.

I. Introduction

Deep neural networks (DNNs) [1], [2] are used to solve challenging machine learning problems. However, CPUs are failing to meet the high computational demand of DNNs. GPUs provide sufficient performance, but are limited by their high power consumption. In contrast, research has shown that FPGAs strike a good balance between performance and energy efficiency for accelerating DNNs.

A DNN comprises a pipeline of computing layers (3D convolution, sub-sampling, nonlinear activation, etc.). Correspondingly, an FPGA-based DNN accelerator comprises one or more layer processors, where each is specialized for computing one or more layers of the target DNN [3], [4].For large DNNs, DRAM is needed to store DNN parameters and layer inputs and outputs. Prior work has shown that DNN computation is highly bandwidth intensive [5], [6]. It is thus essential for the layer processors to fully utilize the available DRAM bandwidth. However, there exists a mismatch between the interface of an FPGA DRAM controller and the layer processors. The nature of FPGAs tends to restrict the frequency of layer processors, which results in the DRAM controller using a wide interface to expose the full DRAM bandwidth to the layer processors (512-bits for a single DDR3 channel). On the other hand, many state-of-the-art FPGA-based DNN accelerators [4], [5] assume the availability of many narrow read and write ports (8 or 16 bits), each with independent DRAM access. This is because narrow ports offer the most flexibility in optimizing the layer processors for the target DNN [4]. As such, a memory interconnect must be used to multiplex the wide DRAM controller interface to a large number of narrow read and write ports, while maintaining maximum bandwidth efficiency.

A memory interconnect performs data transfer as well as request arbitration. The challenge of multiplexing a wide DRAM controller interface lies in *data transfer*, which will be our focus. Mainstream designs of memory interconnects [7], [8] use a 1-to-*N* crossbar to multiplex the wide DRAM controller interface to *N* narrower ports. The crossbar needs to have the same width as the DRAM controller to ensure that the memory bandwidth is fully utilized. Each of the *N* endpoints of the crossbar must then connect to a FIFO to buffer burst transfers and a data-width converter to present a narrow port to the DNN accelerator. While straightforward, such designs are severely over-provisioned: the wide crossbar allows the full DRAM bandwidth to be directed to any narrow port on any cycle, but each narrow port only uses a fraction of the full bandwidth. This excessive flexibility of the interconnect consumes significant logic and wiring resources that can otherwise be used by the DNN accelerator.

To overcome this over-provisioning, a memory interconnect should be optimized to take advantage of the data transfer characteristics of DNN layer processors. In this regard, we make two critical observations. First, the narrow ports used by layer processors are all of the same width, and are all expected to be able to supply one word per cycle. This means that DRAM bandwidth should be statically and evenly partitioned across the narrow ports. Second, a layer processor knows its access pattern and can perform perfect prefetch for future data access, which means that a moderate latency increase in the memory interconnect will not affect system performance.

Based on our observations, we designed Medusa, a resource-efficient, performant, and scalable memory interconnect. In our design, the crossbar, FIFOs, and data-width converters are replaced with a transposition unit. Within the transposition unit, a shifter replaces the crossbar and data-width converters, resulting in significant logic simplification. Moreover, instead of a shallow FIFO per port, the transposition unit uses a deep shared buffer. This allows BRAMs to be efficiently used for buffering, freeing up LUTs and wires for other uses. Importantly, with only a minor *constant* latency increase, Medusa guarantees the same data transfer characteristics as the traditional interconnects, and can be used as a drop-in replacement without changing the layer processor or memory request arbiter design.

Compared to a traditional interconnect, Medusa multiplexes a 512-bit DRAM controller interface across 32 16-bit read ports and 32 16-bit write ports using 4.7x and 6.0x fewer LUTs and FFs, while also improving frequency by 1.8x. For a 1024-bit DRAM controller interface, Medusa runs at 225MHz, while routing congestion limits traditional designs to under 25MHz.

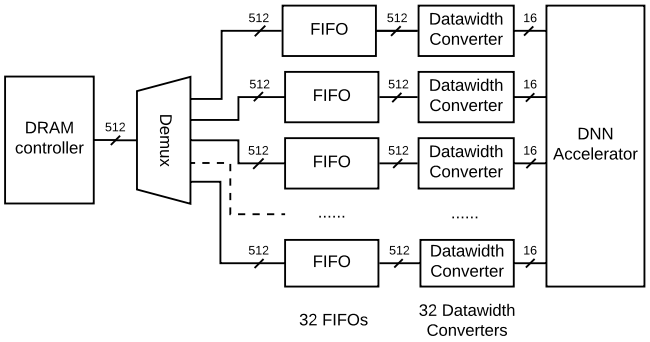


Fig. 1. The baseline memory read data transfer network.

II. Traditional Memory Interconnects

In this section, we present a baseline memory interconnect which is representative of existing designs [7], [8], and we qualitatively discuss its scalability challenges.

A. Baseline Data Transfer Logic

The baseline interconnect's data transfer logic has two parts: one for memory read, and the other for memory write.

1) Baseline Memory Read Data Transfer: Figure 1 shows the baseline memory read data transfer network. In this example, 32 16-bit accelerator read ports share access to one 512-bit DRAM controller interface. The design uses a 1-to-N demux to route input data from the memory controller to N FIFOs, where each FIFO has the same width as the memory interface. This means that the demux can accept a new input from the memory controller on every cycle, allowing the maximum memory bandwidth to be consumed. Each FIFO is provisioned to be large enough to hold the largest burst that a narrow read port can request, so that burst transfer to a single narrow port does not create back pressure. The output of each FIFO is connected to a data width converter, which converts data from the memory interface width to the narrow read port width.

2) Baseline Memory Write Data Transfer: The baseline memory write data transfer network is similar to the read data transfer network, except the data words flow in the opposite direction. Each of the N accelerator write ports feeds into a data width converter, then into a FIFO. Each FIFO has the same width as the memory controller and can hold the maximum burst from a port. On each cycle, an N-to-1 mux chooses the output from one of the FIFOs to write to the memory controller. By using FIFOs to accumulate complete bursts of data, data from the same burst can be sent to the memory controller using the full bandwidth of the memory controller interface.

B. Baseline Scalability Problems

Although the baseline presents a straightforward solution to allow full DRAM bandwidth usage and to eliminate any data switching conflicts among narrow memory ports, its wide demux and mux are over-provisioned in terms of their connectivity. For example, the demux used in the read network has the ability to direct *all* of the read bandwidth to any of the read ports on any cycle. Such flexibility is useful in applications where the partitioning of memory bandwidth to read ports needs to change over time. However, in the context of DNN accelerators, the memory read bandwidth is expected to simply be evenly divided among all the read ports [4]. As such, the extra flexibility of the wide demux only incurs wasted logic (the muxes of the data width converters) and wiring resources. The write network incurs analogous resource waste.

Moreover, the combination of wide and shallow FIFOs leads to inefficient use of FPGA resources. Implementing the shallow FIFOs using BRAMs wastes BRAM capacity, while using LUTRAM consumes a large amount of logic. Additionally, a large number of buses (as wide as the DRAM controller interface) is widely distributed within this design. Handling wide buses introduces challenges with FPGA routing, greatly limiting the peak clock frequency when scaling to wider memory interfaces.

III. Medusa: An Optimized Memory Interconnect

We propose a scalable high performance memory interconnect which is based on data transposition. Figures 2a and 2b provide a high-level overview of the interconnect. Both memory read and write use two data buffers, a rotation unit, and control logic.

Our design evenly partitions the DRAM bandwidth to each port of the DNN accelerator by transposing data instead of routing it with wide demuxes and muxes, thus reducing FPGA resource and routing complexity, without compromising DRAM bandwidth utilization.

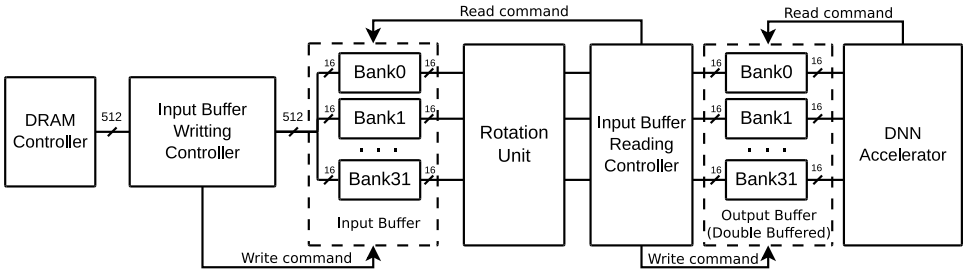
A. Bandwidth Partitioning Through Transposition

Here we provide detailed descriptions of how transposition is used for memory read and write.

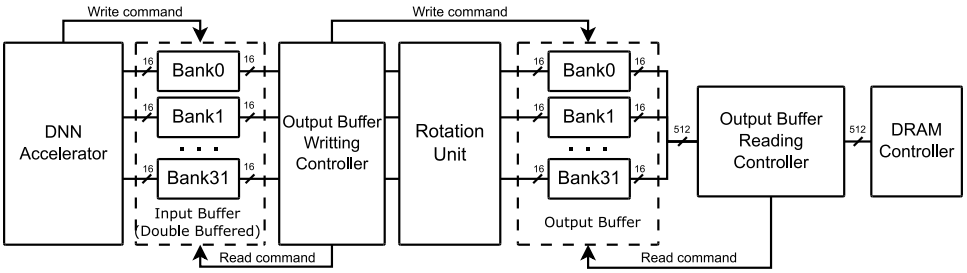
1) Transposition for Memory Read: Figure 3 shows an example of transposition for memory read. Each memory line is $W_{line} = 64$ bits, each accelerator port is $W_{acc} = 16$ bits wide, and $N = W_{line}/W_{acc} = 4$ accelerator ports are used. We mark each data word with coordinates (x,y), where x represents the word's destination accelerator port, and y is the word's index within its containing memory line. Words in the same memory line are always destined to the same accelerator port, and are sent to the destination port in increasing index order. Each W_{line}-bit memory line is stored across the input buffer banks (seen at the bottom of the figure). Specifically, words that are destined to accelerator port i are stored in address i of each of the input buffer banks.

Transposition is performed by reading data words from the input buffer, rotating them, and storing them in appropriate locations in the output buffer. First, at cycle c, words along the diagonal $(0,c \bmod N)$ to $(N-1,(c+N-1) \bmod N)$ are read. For example, Figure 3a shows $c = 0$, where words $(0,0),...,(3,3)$ are read, and Figure 3b shows $c = 1$, where words $(0,1),...,(3,0)$ are read. The rotation unit then takes these N words and rotates them to the left by $c \bmod N$ locations. For example, Figure 3c shows that during $c = 3$, the words are rotated 3 positions to the left. Lastly, the output buffer stores the words into transposed locations: on cycle c, bank i will store data into address $(i+c) \bmod N$. The transposition completes in N cycles, after which each accelerator port can read from its corresponding output buffer bank.

2) Transposition for Memory Write: Memory writes are performed similarly, but with data flowing in the opposite direction. Each accelerator port writes data words into its own bank of the input buffer. The interconnect then transposes input buffer banks to rows in the output buffer.

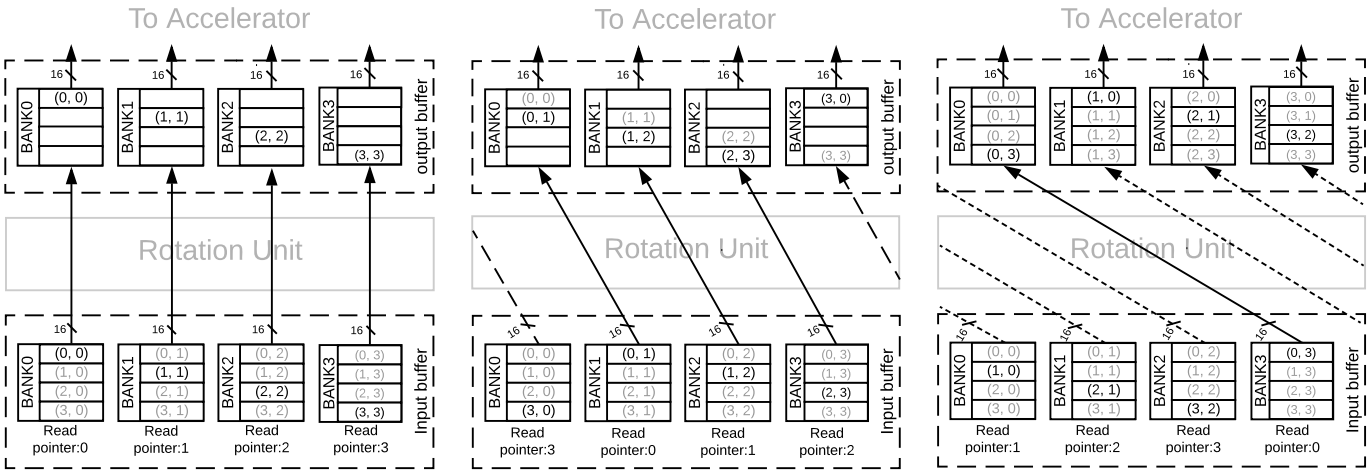


(a) Medusa's memory read data transfer network



(b) Medusa's memory write data transfer network

Fig. 2. High level view of the memory interconnect Medusa. Controller modules keeps track of data and space availability in buffers. Buffers next to the DNN accelerator are double buffered. A line from the DRAM controller is 512-bit (W_{line}). Each port of the DNN accelerator is 16-bit wide ($W_{acc} = 16$ bits).



(a) Cycle 0. (b) Cycle 1. (c) Cycle 3 with transposition result.

Fig. 3. A detailed transposition example for memory read.

For both memory read and write, the interconnect is capable of processing one W_{line}-bit line per cycle, as all parts (the rotation unit, input buffer read/write, output buffer read/write) operate on W_{line}-bit data in parallel. Therefore the system can deliver the full bandwidth of the DRAM controller interface to the accelerator ports. Furthermore, the bandwidth is evenly partitioned across the ports, matching the accelerator's requirements.

B. Rotation Unit Design

The data rotation unit takes N values of W_{acc} bits each and left-rotates them in increments of W_{acc} bits (rotating by $W_{acc} \times c$ bits in cycle c). Figure 4 shows an example rotation unit with $N = 8$ ports. This unit, using a barrel shifter structure, passes data through $\log_2(N)$ levels of logic, where level ℓ is capable of rotating the word by the bit length of 2^ℓ words. Stage ℓ is controlled by bit ℓ of the binary encoding of the desired rotation amount, where logic-1 indicates that the stage should rotate. Data rotation can either be performed in a single cycle or be pipelined, depending on the frequency requirements.

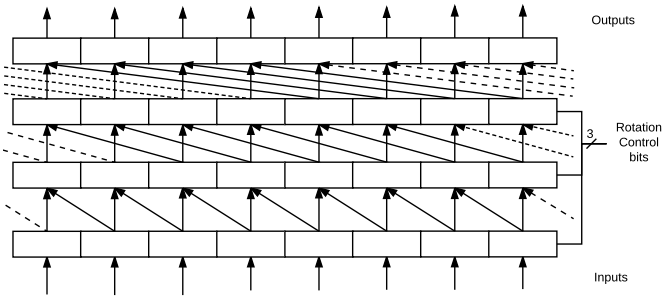


Fig. 4. An example data rotation unit for supporting eight ports.

C. Support for Burst Transfer

Support for burst data transfers is necessary to utilize the bandwidth available from the DRAM controller.

1) Burst Transfer for Memory Read: A request can generate a burst of line transfers to its port. Therefore, the input buffer must be large enough to accommodate at least one burst per port. In other words, the input buffer capacity must be at least $MaxBurstLen \times N$, with N being the number of ports. For each port, head and tail pointers are maintained to track its input buffer space. In each cycle, only the lines at the head pointers participate in rotation. A head pointer is incremented when the line it points to has finished transposition. Tail pointers control where incoming memory lines are written.

2) Burst Transfer for Memory Write: The output buffer capacity must be at least $MaxBurstLen \times N$. Similar to the case of memory read, head and tail pointers are used to keep track of the buffer space for each port. Notably, for memory write, the request arbiter must monitor data coming from the write ports, and only issue requests for ports that have accumulated enough data in the output buffer to finish the write request. This requirement also applies to the baseline interconnect.

D. Interconnect Scalability

By replacing the baseline wide mux/demux and data-width converters with rotation units, the Medusa interconnect significantly reduces logic cost from $W_{line} \times (N-1)$ 2-to-1 one-bit muxes down to $W_{line} \times \log_2(N)$ [9]. Furthermore, the Medusa interconnect consolidates the shallow and wide FIFOs of the baseline design into large buffers with deep and narrow banks, making them amenable to efficient storage in BRAM.

E. Latency Overhead

Compared to the baseline, our transposition-based design has a *constant* latency overhead of W_{line}/W_{acc} cycles. This happens because a memory line can only be consumed after it has been transposed. For a typical case, $W_{line}/W_{acc} = 512/16 = 32$. In the context of DNN accelerators, this latency overhead has a negligible impact on performance, because DNN layer processors double buffer their inputs and perform perfect prefetch of data into the idle buffers.

Note that, even for burst transfers, the latency overhead of Medusa is still W_{line}/W_{acc} cycles. This is because as soon as the head of a burst arrives, transposition can start.

F. Data Transfer Characteristics

In the example in Figure 3, the buffer has data available for each port at the time when the transposition begins. However, this is *not* a requirement of the design. The control logic starts transposition for a port without waiting for the other ports, and a port can join the transposition when transfers on the other ports are already in progress. In other words, the transposition design does *not* incur any interference among ports.

Overall, except for the constant latency overhead explained in Section III-E, the data transfer characteristics of the Medusa interconnect are identical to that of the baseline.

TABLE I
MEDUSA VS. BASELINE (FPGA RESOURCE USE).

		LUT	FF	BRAM-18K	DSP
Baseline	Read Network	18,168 (4.2%)	19,210 (2.2%)	0 (0%)	0 (0%)
	Write Network	26,810 (6.2%)	35,451 (4.1%)	0 (0%)	0 (0%)
	Total	198,887 (45.9%)	240,449 (27.8%)	726 (24.7%)	2,048 (56.9%)
Medusa	Read Network	4,733 (1.1%)	4,759 (0.6%)	32 (1.1%)	0 (0%)
	Write Network	4,777 (1.1%)	4,325 (0.5%)	32 (1.1%)	0 (0%)
	Total	156,409 (36.1%)	195,158 (22.5%)	790 (26.9%)	2,048 (56.9%)

IV. EVALUATION

We compare the Medusa transposition-based interconnect and the baseline interconnect by looking at their resource use, performance, and scalability. Note that both interconnects use the same request arbitration logic, hence our evaluation focuses on the data transfer networks within the interconnects.

We used Bluespec to implement both interconnects. We perform synthesis as well as place and route (P&R) using Xilinx Vivado 2016.4, targeting Virtex-7 690T. To ensure validity, we checked that when multiplexing a 256-bit port to 16 16-bit ports, our baseline implementation used fewer resources than an equivalent implementation built with Xilinx IPs [9]. To get representative results, when doing synthesis and P&R for an interconnect, a convolutional layer processor [4] is added and connected to all the narrow read/write ports. The configuration of the layer processor is suitable for VGGNet [1]. To make it easy to experiment with different design scales, we replaced the DRAM controller and PCIe controller with stubs in our test designs. The exclusion of these components gives *equal benefit* to the baseline and transposition-based designs. Additional details of our experiments can be found in arXiv [9].

A. Hardware Resource Usage

To evaluate the hardware resources required by Medusa, we thoroughly evaluate a representative design point, which uses a 512-bit memory interface and a layer processor with 32 16-bit read and write ports and 2048 DSP slices. For each read/write port, the interconnect can buffer a maximum burst of 32×512-bits.

Table I shows the resource breakdown of the two designs. For each design, we present the resource use of the whole design, the read data-transfer network, and the write data-transfer network in isolation. The percentages show resource use relative to the capacity of a Virtex-7 690T.

First, we focus on the data transfer networks in isolation. For memory read, compared to the baseline, the Medusa transposition-based network reduces LUT use by 3.84x and FF use by 4.04x, at a cost of 32 BRAMs. For memory write, the Medusa transposition-based network reduces LUT use by 5.61x and FF use by 8.20x, also at a cost of 32 BRAMs. Combined, the Medusa networks achieve 4.73x LUT and 6.02x FF savings, at a minor BRAM cost.

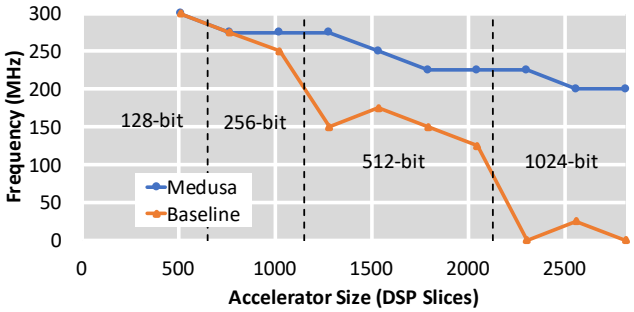


Fig. 5. Change in peak frequency as the accelerator scales.

We next consider the entire design, including the layer processor and memory interconnect (Total). The baseline uses 1.27x more LUTs and 1.23x more FFs than the Medusa transposition-based design, whereas the transposition-based design uses 1.09x more BRAM. This shows that the LUT and FF savings achieved by the Medusa data transfer networks are significant even in the context of a resource-heavy layer processor. In the baseline, the read and write data-transfer networks account for 22.6% of the total LUT use and 22.7% of the total FF use of the accelerator. Medusa reduced these to 6.1% and 4.7%, respectively.

The Medusa network's efficiency stems from its lower logic complexity and its ability to make efficient use of BRAMs, saving LUTs, FFs, and routing resources. The Medusa design uses a total of 64 BRAMs to efficiently buffer data. In contrast, if the baseline design were to use BRAMs in its data-transfer networks, 960 BRAMs would be needed, making it a poor trade-off with respect to the savings in FFs and LUTs. This is because each 18-Kbit BRAM is 36 bits wide, and each 32x512-bit FIFO would consume 15 BRAMs, requiring a total of 960 BRAMs for 32 memory-read FIFOs and 32 memory-write FIFOs.

B. Performance and Scalability

To evaluate Medusa's effect on performance, we find the peak post-P&R frequency of each design, searching in 25MHz steps. For scalability, we adjust the layer processor's size from 512 to 2816 DSP slices, in steps of 256 DSP slices. Larger designs require more read/write ports, so the DRAM interface width also grows from 128 bits to 1024 bits.

Figure 5 shows how the peak reachable frequency changed as the accelerator's size increased. The vertical dashed lines divide the data points into four regions, based on DRAM interface width. Points at 0MHz indicate that Vivado was not able meet timing at 25MHz. The data points at 2048 DSP slices correspond to the designs examined in Section IV-A. In this case, the baseline's peak frequency is 125MHz, whereas the design using Medusa can reach 225MHz, 1.8x faster. As the designs scale upwards, the performance benefit of Medusa increases: once the DRAM interface grows to 1024 bits, Medusa can reach 225MHz, yet the baseline failed to meet timing at 25MHz.

V. Related Work

Our work focuses on providing an efficient memory interconnect for DNN accelerators that require access to DRAM through many narrow read and write ports. Some designs [10], [11] avoid the need for such an interconnect by altering the layout of data in DRAM. The main drawback of this approach is that it limits the choices of data flows inside the layer processors, which can lead to underutilization of compute units [4]. Other designs [3], [12] avoid the width mismatch problem by using narrow memory controller buses, which can result in bandwidth bottlenecks even when DRAM bandwidth is available.

VI. Conclusions

This paper presented a resource efficient and high-performance memory interconnect for connecting many-port DNN accelerators to wide DRAM controller interfaces. We analyzed and experimented with commonly-used mux/demux-based interconnects, and concluded that they were over-provisioned and had serious scalability limitations.

To address this problem, we tailored our design to the needs of DNN accelerators and used a transposition unit to implement memory bandwidth partitioning. Our design has lower logic complexity and can efficiently use BRAMs to reduce LUTRAM use. Experiments showed that, compared to the baseline design, our Medusa design reduced LUT and FF usage by 4.7x and 6.0x respectively, and improved peak frequency by 1.8x.

Acknowledgments

This material is based on work supported by the National Science Foundation (NSF) under Grant Nos. 1533739 and 1453460. The experiments were conducted with equipment purchased through NSF CISE Research Infrastructure Grant No. 1405641.

References

[1] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *CoRR*, vol. abs/1409.1556, 2014.
[2] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, 2015.
[3] H. Sharma, J. Park, D. Mahajan, E. Amaro, J. K. Kim, C. Shao, A. Mishra, and H. Esmaeilzadeh, "From high-level deep neural models to FPGAs," in *Proceedings of the 49th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO '16, 2016, pp. 1–12.
[4] Y. Shen, M. Ferdman, and P. Milder, "Maximizing CNN accelerator efficiency through resource partitioning," in *Proceedings of the 44th Annual International Symposium on Computer Architecture*, ser. ISCA '17, 2017, pp. 535–547.
[5] Y. Shen, M. Ferdman, and P. Milder, "Escher: A CNN accelerator with flexible buffering to minimize off-chip transfer," in *Proceedings of the 25th IEEE International Symposium on Field-Programmable Custom Computing Machines*, ser. FCCM '17, 2017, pp. 93–100.
[6] M. Alwani, H. Chen, M. Ferdman, and P. Milder, "Fused-layer CNN accelerators," in *Proceedings of the 49th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO '16, 2016, pp. 1–12.
[7] Xilinx, "AXI Interconnect v2.1," 2017.
[8] Altera, "Qsys Interconnect," 2013.
[9] Y. Shen, T. Ji, M. Ferdman, and P. Milder, "Medusa: A scalable interconnect for many-port DNN accelerators and wide DRAM controller interfaces," *arXiv preprint arXiv:1807.04013*, 2018.
[10] C. Zhang, Z. Fang, P. Zhou, P. Pan, and J. Cong, "Caffeine: Towards uniformed representation and acceleration for deep convolutional neural networks," in *Proceedings of the 35th International Conference on Computer-Aided Design*, ser. ICCAD '16, 2016, pp. 1–8.
[11] J. Qiu, J. Wang, S. Yao, K. Guo, B. Li, E. Zhou, J. Yu, T. Tang, N. Xu, S. Song, Y. Wang, and H. Yang, "Going deeper with embedded FPGA platform for convolutional neural network," in *Proceedings of the 24th ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '16, 2016, pp. 26–35.
[12] Y.-H. Chen, J. Emer, and V. Sze, "Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks," in *Proceedings of the 43rd International Symposium on Computer Architecture*, ser. ISCA '16, 2016, pp. 367–379.