# Hardware Acceleration of CNNs with Coherent FPGAs

Extended Abstract

Md Syadus Sefat
Ingram School of Engineering
Texas State University
m_s597@txstate.edu

Semih Aslan
Ingram School of Engineering
Texas State University
aslan@txstate.edu

Apan Qasem
Computer Science Dept.
Texas State University
apan@txstate.edu

**ACM Reference Format:**
Md Syadus Sefat, Semih Aslan, and Apan Qasem. 2018. Hardware Acceleration of CNNs with Coherent FPGAs: Extended Abstract. In *Proceedings of International Conference for High Performance Computing, Networking, Storage and Analysis (SC'18).* Dallas, TX, USA, 2 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

## 1 INTRODUCTION

In recent years, Convolution Neural Networks (CNN) have emerged as a powerful technique for a range of Deep Learning applications. In particular, CNNs can provide highly accurate predictions for face detection and image classification problems in Computer Vision. Hardware acceleration of neural networks is a promising direction [3, 5, 7]. Although currently FPGAs cannot deliver the raw compute power demanded by large-scale deep learning applications, accelerated implementations generally yield better energy efficiency in the form of improved performance/watt [1]. This is critical for inference tasks which are often deployed for execution on battery-powered edge devices.

Hardware acceleration of CNNs poses significant challenges. Real-world CNNs are constructed with millions of model parameters requiring hundreds of megabytes of storage for each layer, which far exceeds the capabilities of current FPGAs. As a result, accelerated implementations incur high I/O overhead and performance is often dominated by data transfer time over a low-bandwidth I/O bus such as PCIe. Another major obstacle with FPGA acceleration, CNN or otherwise, is the time to development. Recent introduction of high-level synthesis tools, such as OpenCL and Vivado HLS has increased FPGA programmability with respect to data path representation. Nonetheless, a custom FPGAs still requires writing device driver code to access memory-mapped I/O and communicate with the CPU, which can significantly add to the development time.

This paper describes a new flexible approach to implementing energy-efficient CNNs on FPGAs. Our design leverages the Coherent Accelerator Processor Interface (CAPI) which provides a cache-coherent view of system memory to attached accelerators [4]. Convolution layers are formulated as matrix multiplication kernels and then accelerated on a CAPI-supported Kintex FPGA board. Our implementation bypasses the need for device driver code and


*SC'18, November 2018, Dallas, TX, USA*

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

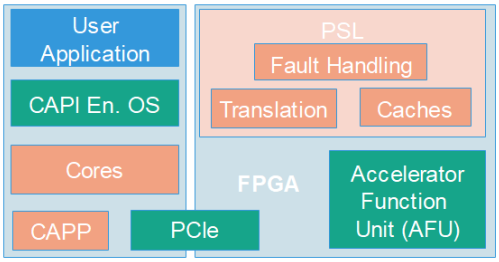


**Figure 1: POWER8 CAPI Architecture**

significantly reduces the communication and I/O transfer overhead. To improve the performance of the entire application, not just the convolution layers, we propose a collaborative model of execution in which the control of the data flow within the accelerator is kept independent, freeing-up CPU cores to work on other parts of the application. For further performance enhancements, we propose a technique to exploit data locality in the cache, situated in the CAPI Power Service Layer (PSL). Finally, we develop a resource-conscious implementation for more efficient utilization of resources and improved scalability.

## 2 COHERENT ACCELERATORS

The CAPI technology, introduced by IBM in 2014, enables coherent connection to custom acceleration engines within a heterogeneous compute unit. In the CAPI paradigm, the custom designed accelerator is kept inside the Accelerator Function Unit (AFU). CAPI enables the system to use the AFU unit as a coherent peer to the CPU cores. The Coherent Accelerator Processor Proxy (CAPP) unit, implemented as a functional unit within the processor and the PSL unit on the FPGA, work together to maintain communication and create a coherent view of the system memory space. Fig. 1 shows a block diagram of the CAPI architecture. The PSL contains a 256 KB cache which can be accessed by the accelerator. An application running on the CPU sends a Work Element Descriptor (WED) to the AFU. WED contains pointers to meta-data and data in the user space on which accelerated computation is performed by the AFU.

In traditional HW-SW collaboration paradigm, the accelerator is attached as a memory-mapped I/O device. A device driver performs the virtual to physical address translation and delivers the addresses of the pinned kernel buffer to the accelerator. CAPI adds an Effective-to-Real-Address Translator (ERAT) within the PSL that translates the addresses, greatly reducing address translation overhead. In traditional systems, when an accelerator finishes its computation, it writes the data in the kernel space. Then a device driver copies the data from kernel space to the user space, generates a pointer to the data and passes it to the application. Thus, the same

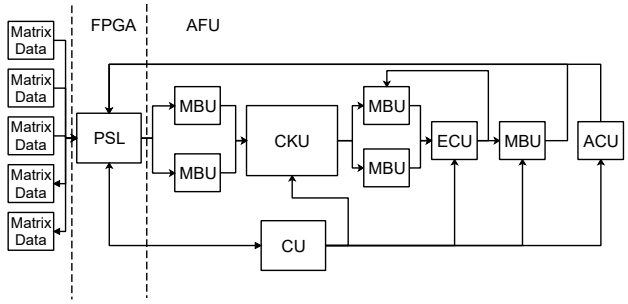


**Figure 2: Simplified Hardware Architecture**

data is copied *twice*. In CAPI, the pointer to the user space data is sent from the application directly to the FPGA, thereby eliminating the cost of copying data from kernel to user space.

## 3 HARDWARE DESIGN

To solve the challenges of serving memory intensive CNN computation on accelerators, our hardware exploits parallelism while performing different operations in several layers of a CNN with the benefit of having seamless access in the shared coherent memory. The hardware architecture (Fig. 2) inside the AFU consists of computational kernel units (CKUs), a controller unit (CU), multi-ported buffer unit (MBU), extra computational unit (ECU) and an address computing unit (ACU). The CKUs have specialized computational units such as dot block (DB) and ReLu block (RB) for tackling the most common CNN operations. The DBs are the largest computational blocks in the accelerator. An $n \times n$ DB kernel consists of $n$ vector dot block (VDB) units. Each VDB unit computes the vector dot product of two vectors of size $n$. A VDB unit consists of $n$ multipliers and $(n/2)$ adders. The CU is responsible for extracting information from meta-data in WED and maintaining the state of the hardware. The ACU calculates the virtual address for reading and writing user space data to- and from- the shared memory.

### 3.1 Collaborative execution

If the matrix size is greater than the available on-chip memory of the FPGA then it becomes necessary to sub-divide the matrix and copy it from system memory in multiple transactions. In traditional systems, the SW stack assists the accelerator device each time the hardware needs to read data from or write data to system memory. Consequently, the running application (and the CPU) remains busy in communicating with the hardware accelerator while the accelerator performs the memory transactions. In the CAPI architecture, the AFU can perform I/O transactions independently. This allows the SW stack and the CPUs to perform other useful tasks in the application while the HW is busy in I/O transactions. This collaborative model, which improves overall utilization of computing resources, can be particularly beneficial for heterogeneous supercomputers running large-scale deep learning applications.

### 3.2 Resource awareness

We modified the main computation unit in VDB to make it more resource-conscious. This new VDB, VDB_RC, optimizes the area utilization on the FPGA without sacrificing performance. The design of VDB_RC is driven by the key insight that although the latency within the block can increase due to reduced parallelism, the compact design allows instantiation of a higher number of VDB_RC computation blocks, increasing overall performance. For an $n \times n$ VDB unit, the computation block consists of one row of $(n/2)$ parallel multipliers and one row of $(3n/8)$ adders, split into $(n/4)$ and $(n/8)$ units respectively.

## 4 PRELIMINARY RESULTS

We implemented the hardware architecture on a Kintex KU115 FPGA running at 250 MHz to work with 32-bit floating point data. Table 1 compares the performance of our work with previous works in terms of GOPS and GOPS/w.

**Table 1: Comparison with previous works**

| | [6] | [2] | Ours |
|---|---|---|---|
| year | 2015 | 2016 | 2018 |
| Platform | Virtex7 VX458t | Zynq XC7Z045 | CAPI KU115 |
| Clock (MHz) | 100 | 150 | 250 |
| Quantization | 32-bit float | 16-bit fixed | 32-bit float |
| Performance (GOP/s) | 61.62 | 136.97 | 155.08 |
| Power(W) | 18.61 | 9.63 | 9.82 |
| Power Efficiency (GOP/s/W) | 3.31 | 14.22 | 15.80 |

## 5 CONCLUSIONS AND FUTURE WORK

To the best of our knowledge, this is the first paper that implements CNN operations on a CAPI enabled hardware accelerator. Preliminary experiments show substantial performance gains over previous methods and make the case for further exploration of hardware accelerated design of CNN and other deep learning applications. In future, we plan to deploy all performance-critical CNN operations in a multi-FPGA heterogeneous environment.

## ACKNOWLEDGMENTS

This work was supported by the National Science Foundation through awards CNS-1305302 and CNS-1253292 and by IBM Corporation through a Faculty Award and equipment donation.

## REFERENCES

[1] E. Nurvitadhi and et al. Can FPGAs beat GPUs in accelerating next-generation deep neural networks? In *ACM FPGA*, 2017.
[2] J. Qiu and et at. Going deeper with embedded fpga platform for convolutional neural network. In *ACM/SIGDA*, pages 26–35. ACM, 2016.
[3] H. Sharma and et al. From high-level deep neural models to FPGAs. In *MICRO*, pages 1–12, Oct. 2016.
[4] J. Stuecheli and et al. CAPI: A coherent accelerator processor interface. *IBM Journal of Research and Development*, 59(1):7–1, 2015.
[5] N. Suda and et al. Throughput-Optimized OpenCL-based FPGA Accelerator for Large-Scale Convolutional Neural Networks. In *ACM FPGA*, 2016.
[6] C. Zhang and et al. Optimizing fpga-based accelerator design for deep convolutional neural networks. In *ACM/SIGDA*, pages 161–170. ACM, 2015.
[7] C. Zhang and et al. Caffeine: Towards Uniformed Representation and Acceleration for Deep Convolutional Neural Networks. ICCAD '16, pages 12:1–12:8. ACM, 2016.