Binarized ImageNet Inference in 29μs

Tong Geng*†, Ang Li*, Tianqi Wang†, Shuaiwen Leon Song*, Martin Herbordt†
*Pacific Northwest National Laboratory; †Boston University

1 INTRODUCTION

The past decade has witnessed the emergence of Deep Neural Networks (DNNs) and their widespread adoption in, not only image processing and speech recognition, but also supercomputing scenarios such as extreme big-data processing for in-situ analysis. To obtain improved prediction accuracy, DNNs are becoming deeper and more complex, leading to increasingly long processing latency and high resource demand. So far, the use of accelerators has achieved only limited benefit in real-time domains with strict latency constraints [2], such as autonomous driving and robotic control.

Since DNNs can often tolerate some inaccuracy, researchers have begun exploring reduced bit-width for DNN training and inference. Many new designs have been proposed; the Binarized-Neural-Network (BNN) [7] is particularly attractive. BNNs use a single bit to encode each neuron, significantly reducing compute complexity and memory demand, giving the potential to reduce inference delay by orders-of-magnitude. BNNs map particularly well to FPGAs: their configurability enables millions of one-bit ALUs to be implemented on a single device. In contrast, a report from Intel reports only 10% peak performance when running a BNN on a Xeon CPU and 7% for a Titan X GPU (with batch size of 10; a smaller batch size of about four is generally expected [6]).

Despite the attractiveness of FPGAs for BNNs, creating a good design is still a major challenge, especially for a large network with big data (e.g., AlexNet on ImageNet). To the best of our knowledge, the shortest reported latency of AlexNet inference is 1.16ms [5], which is still far from real-time. The delay is mainly due to:
1. The critical batch-normalization layer (NL) [1, 7] uses full-precision floating point (i.e., 2 FP MUL/DIV + 3 FP ADD/SUB).
2. To efficiently process a large BNN with a single FPGA, ideally (a) each layer is optimally designed (all control-flow-graphs are configured) and (b) layers do not need to be reconfigured. To the best of our knowledge, no existing work has achieved both objectives.
3. Almost all previous systems accelerate BNNs by exploiting data parallelism with layers processed sequentially. Hence, the overall latency of a network is the accumulated latency of every layer in addition to the communication and reconfiguration overhead. As a result, the latency expands with network depth. In addition, since a layer cannot start processing until the completion of the previous layer, huge on-chip storage is required to buffer all of the intermediate image data between layers.

Our main result is to address these three difficulties and demonstrate a single-FPGA design that can finish AlexNet BNN inference in 29us. We achieve this performance with the following contributions: **1.** We completely remove floating-point from the NL through layer fusion. **2.** By using model parallelism rather than data parallelism, we can simultaneously configure all layers and control flow graphs. Also, the design is flexible enough to achieve nearly perfect load balancing, leading to extremely high resource utilization. **3.** All convolution layers are fused and processed in parallel through inter-layer pipelining. Therefore, when the pipeline is full, latency is just the delay of a single convolution layer plus the FC layers. Note that the dependency pattern of the FC layer prevents it from being integrated into the current pipeline.

2 METHODS

Intra-Layer Fusion: The original BNN structure has 5 sub-layers in each CONV/FC layer: XNOR, POPCOUNT, Activation (ACT), NL, and the Binarization Layer (BL). Normalized outputs from NL, which are floating-point, are binarized in BL by being compared with 0. To eliminate the precision gap between NL (FP32) and BL (0/1), we fuse ACT, NL, and BL and replace them with a Comparison Layer (CL) (Fig. 1). In the simplified network, data from the POPCOUNT layer are binarized directly by a comparison with an integer threshold computed according to Eq. 1. The threshold is a floating point number, but since $x_{i,j}$ is the result of POPCOUNT, $max(x_{i,j}, 0)$ in BNN is an integer. When comparing an integer with a floating-point threshold, the threshold can be rounded up to an integer. The 1-bit output of CL is used as the input of the next CONV/FC layer. There is no accuracy loss during this simplification. With this intra-layer fusion, two comparisons from ACT and BL, along with 5 floating point operations from NL, are replaced by a single integer compare.

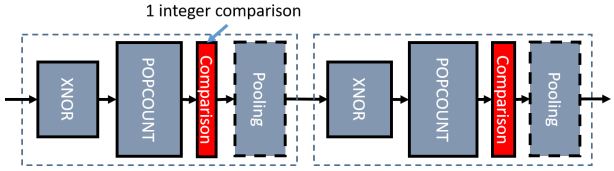


Figure 1: Optimized BNN with intra-layer fusion

$$y_{i,j} = \begin{cases} -1 & \text{if } max(x_{i,j}, 0) < \mathbb{E}_{*,j} - \frac{\beta_j \cdot \sqrt{Var[x_{*,j}] + \epsilon}}{\gamma_j} \\ 1 & \text{otherwise} \end{cases} \quad (1)$$

Implementation Details: Optimal designs of all layers are configured simultaneously. For full utilization, each CONV/FC layer is designed so that producer and consumer rates match.

Parameterized Architecture for single CONV layer. Fig. 2 illustrates the architecture of a single CONV layer. The parameters *PIC* and *POC* refer to the numbers of input and output channels processed in-parallel; *SIC* and *SOC* refer to the numbers of input and output channels processed sequentially. K denotes the kernel size. Data from all the input channels of the previous layer are buffered in the Shared Input Data Memory (SIDM). When the next layer starts being processed, $PIC * K$ data from the data memory are broadcast to the *POC* PEs.

Each PE has an XNOR engine, a POPCOUNT engine, and a comparator coupled with a local threshold buffer which is a distributed-RAM-based shift register. All PEs work in lockstep under direction of a control unit. PEs gather weights from shared weight memory. In order to provide enough weights to service the *POC* PEs, $POC*PIC*K*K$ weights must be accessed in parallel. We use Block RAMs (BRAM) in addition to Distributed RAM to buffer weights. To give BRAMs enough concurrency, weights for 32 different channels are packed. If there is a pooling layer, outputs of PEs are cached in a shared pooling buffer to wait for the data to be compared. The shared data memory and pooling buffer are designed in the same way as weight memory. Input/output data for 32 input/output channels are packed before they are buffered. Input, output, and weights for the channels which are processed sequentially are stored in

interleaved order. Data memory is implemented by K sets of FIFOs linked head-to-tail to support the sliding of kernel window.

Balancing data generation/consumption. For high utilization, *SIC* and *SOC* for each layer are tuned according to the data generation rate of the previous layer so that data generation and consumption rates match. For example, $SIC * SOC$ of a module following a $2 * 2$ pooling layer is scaled down by a factor of 4.

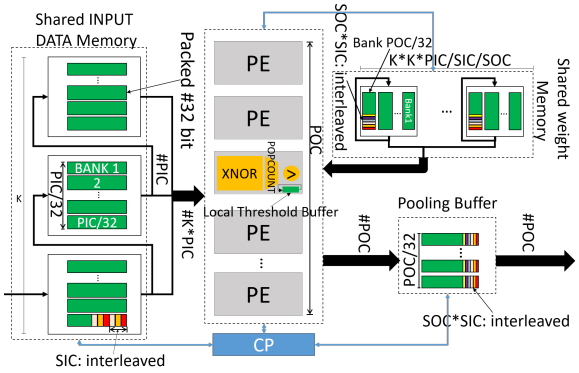
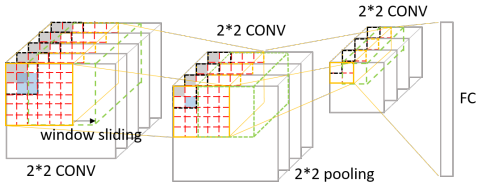
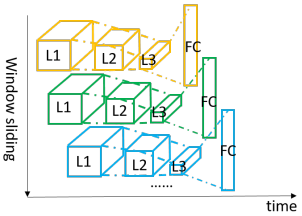


Figure 2: Parameterized architecture of single CONV layer: SOC, SIC, POC, and PIC can be tuned to degrade the parallelism for high utilization

Inter-layer fusion: A fine-grained inter-layer pipeline is used to fuse all CONV layers and the first FC layer, leading to overlapped latencies among these layers. With this pipeline, an operation in a layer is processed immediately when necessary inputs are available rather than waiting for the completion of the previous layer. Hence, all layers except the last couple of FC layers are processed in parallel and their latencies are overlapped to the point that their overall latency is that of a single layer in addition to the time waiting for the dependent inputs. Fig. 3(A) shows the data dependency of BNN. Fig. 3(B) illustrates the inter-layer pipeline: it not only reduces the latency greatly but also reduces storage demand because the activations are propagated and consumed quickly between layers.



(A) Data dependency of BNN



(B) Fine-grained inter-layer pipelining for inter-layer fusion

Figure 3: Data dependency of BNN and the proposed fine-grained inter-layer pipelining for inter-layer fusion

3 EXPERIMENTAL RESULTS

We use a Virtex VCU118 FPGA to evaluate performance, energy efficiency, utilization, and latency; the application is inference of AlexNet. We compare with existing work on BNN acceleration on GPUs, CPUs, and FPGAs. As GPUs and CPUs are extremely underutilized when executing BNNs in no-batch mode, the critical comparison is with the current state-of-the-art FPGA-based BNN accelerator. As shown in Table 1, our latency is 40× better than that design. The performance is evaluated with respect to Images/s. Using our design, 34480 224*224 images can be inferred per second. Energy efficiency is evaluated with Images/J. Our result is at least 32.7× better than the existing work. The utilization of our design is 99.7%, i.e., the percentage of idle stages in pipeline is only 0.3%. The resource usage for the implementation of AlexNet is listed in Table 2. The operating frequency is 200MHz.

Table 1: Comparison of Latency, Performance, and Energy Efficiency using GPU, FPGA, CPU for AlexNet inference

	CPU	GPU	FPGA	
Platform	Xeon E5-2640 [5]	Tesla K40 [5]	Stratix V [5]	Ours: VCU118
Latency (ms)	10789	1257	1.16 (1x)	0.029 (40x)
Performance (Images/s)	0.093	0.80	862 (1x)	34480 (40x)
Energy Efficiency (Images/J)	0.00098	0.0034	32.9 (1x)	1078 (32.7x)

Table 2: Resource Usage of LUT, FF, BRAM and DSP for the implementation of AlexNet

LUT	FF	BRAM	DSP
635k/1182k (53.7%)	1527k/2364k (64.6%)	1459/2160 (67.5%)	5808/6840 (84.9%)

4 CONCLUSION

In this work, a single-FPGA-based accelerator for ultra-low-latency inference of ImageNet is proposed. The design can complete the inference of Binarized AlexNet within 29μs with accuracy comparable to other BNN implementations. For extensions of some of these ideas to FPGA clusters [4] see [2, 3].

ACKNOWLEDGMENTS

This research was funded by Deep Learning for Scientific Discovery Investment Pacific Northwest National Laboratory's Laboratory Directed Research and Development Program. The Pacific Northwest National Laboratory is operated by Battelle for the U.S. Department of Energy under contract DE-AC05-76RL01830.

REFERENCES

[1] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio. 2016. Binarized neural networks: Training deep neural networks with weights and activations constrained to +1 or -1. *arXiv preprint arXiv:1602.02830* (2016).

[2] T. Geng, T. Wang, A. Sanaullah, C. Yang, R. Xu, R. Patel, and M.C. Herbordt. 2018. FPDeep: Acceleration and Load Balancing of CNN Training on FPGA Clusters. In *Int. Symp. Field-Programmable Custom Computing Machines*. 81–84.

[3] T. Geng, T. Wang, A. Sanaullah, C. Yang, R. Xuy, R. Patel, and M.C. Herbordt. 2018. FPDeep: A Framework for CNN Training Acceleration on FPGA Clusters. In *Proc. IEEE Conf. on Field Programmable Logic and Applications*.

[4] A. George, M. Herbordt, H. Lam, A. Lawande, J. Sheng, and C. Yang. 2016. Novo-G#: A Community Resource for Exploring Large-Scale Reconfigurable Computing Through Direct and Programmable Interconnects. In *IEEE High Perf. Extreme Computing Conf.*

[5] S. Liang, S. Yin, L. Liu, W. Luk, and S. Wei. 2018. FP-BNN: Binarized neural network on FPGA. *Neurocomputing* 275 (2018), 1072–1086.

[6] E. Nurvitadhi, D. Sheffield, J. Sim, A. Mishra, G. Venkatesh, and D. Marr. 2016. Accelerating binarized neural networks: comparison of FPGA, CPU, GPU, and ASIC. In *Int Conf Field-Programmable Technology*. 77–84.

[7] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi. 2016. XNOR-net: Imagenet classification using binary convolutional neural networks. In *European Conference on Computer Vision*. 525–542.