Bit-Slicing FPGA Accelerator for Quantized Neural Networks

Olexa Bilaniuk*, Sean Wagner†, Yvon Savaria‡ and Jean-Pierre David‡
*Mila, University of Montreal, Montreal, QC, Canada
†IBM Canada, Markham, ON, Canada
‡Polytechnique Montreal, Montreal, QC, Canada

***Abstract*—Deep Neural Networks (DNNs) become the state-of-the-art in several domains such as computer vision or speech recognition. However, using DNNs for embedded applications is still strongly limited because of their complexity and the energy required to process large data sets. In this paper, we present the architecture of an accelerator for quantized neural networks and its implementation on a Nallatech 385-A7 board with an Altera Stratix V GX A7 FPGA. The accelerator's design centers around the matrix-vector product as the key primitive, and exploits bit-slicing to extract maximum performance using low-precision arithmetic.**

***Index Terms*—Neural Networks, Accelerators, BNN, CNN, RNN, QNN, FPGA**

I. Introduction

Artificial neural networks have attracted much attention recently, for the (super-) human performance they offer in many applications. But their massive computational requirements make them impractical for generic processors, and hardware accelerators are needed. Several designs have cropped up and strike different balances of performance and accuracy against hardware resource usage: size, complexity, computation, time, storage, energy, area and other criteria. Although purely Application Specific Integrated Circuit (ASIC) hardware solutions seem precluded by the extremely rapid development cycle of new state-of-the-art neural networks, the well-known, more flexible solutions, like DSPs, CPUs or GPUs, have drawbacks such as high energy consumption and cost.

A promising path for improving the hardware performance of neural networks is extreme *quantization*: using low-precision and fixed-point rather than high-precision or floating-point arithmetic. A seminal work in this vein is BinaryNet (BNN) [1] in which both weights and activations are either +1 or -1. BNNs significantly decrease computational intensity, since operations are reduced to bitwise logical operators and popcounts. Ternary [2], two-bit [3], and six-bit [4] representations have also been seen. Significant energy savings can result from low-precision arithmetic, because of the reduced storage, transfer and computation requirements [5].

Several Quantized Neural Network (QNN) accelerators have been implemented using Field-Programmable Gate Arrays (FPGAs) due to their support for arbitrary quantization schemes, fast implementation, and energy efficiency [6]. BNNs are a common target for these accelerators [7], [8]. The FINN (Framework for Fast, Scalable Binarized Neural Network Inference) architecture [9] uses a streaming architecture that is customized at compile-time of the FPGA logic. Similarly, the AccELB project [10] implements a streaming data flow that allows for different bit-widths on each layer of the neural network. In DNNBuilder [11], the software analysis of trained network models is done to exploit multiple degrees of parallelism while balancing memory bandwidth to generate a highly optimized pipeline.

There are a number of challenges with FPGA-based QNNs. Using a single low-bit-width numerical format throughout the QNN can limit the overall achievable accuracy, but may simplify the design. Some designs do vary bit-widths with layers, but fix the bit-width at accelerator synthesis time. Bit-serial processing, such as used in the Loom [12] and BISMO [13] architectures, is one solution so long as high overall throughput via parallelism can be maintained. FPGAs are limited by logic resources, on-chip memory capacity and off-chip bandwidth. The on-chip capacity of the largest FPGAs is still too small (∼50Mbit) to contain large models, and off-chip data movements' energy cost can exceed that of on-chip movements by as much as two orders of magnitude [5].

In this paper, a new QNN accelerator architecture supporting arbitrary low-precision fixed-point formats is introduced. The proposed architecture enables to process the inference pass in large neural networks while implementing an efficient streaming data flow in a flexible pipelined scheme that limits movement of data to off-chip memory. Designs are implemented in an FPGA of modest size, while providing for high overall computational throughput and energy efficiency.

II. Proposed Architecture

Computations in today's neural networks are almost invariably a large number of linear-algebraic operations with a small admixture of elementwise non-linear activations [14]–[17]. While these activation functions may be relatively expensive transcendental functions, such activations have waned in popularity recently, and have been replaced by simple piecewise-linear functions such as the ReLU ($y = \max(0, x)$) [18] and its variants. Not only do they perform better, but they are also better-suited for hardware adaptations.

978-1-7281-0397-6/19/$31.00 ©2019 IEEE

A QNN accelerator design is therefore called to dedicate most of its resources to linear operations, while balancing them with non-linear activation throughput. This linear operation is most commonly the BLAS Level 3 operation GEMM (GEneral Matrix-Matrix product), since all linear operations can be expressed in terms of matrix multiplications. GEMM is also often the most optimized primitive in high-performance computing. All other linear operations are then retrofitted into that primitive. An example of this philosophy is NVIDIA's cuDNN [19], a collection of neural network kernels for RNN and CNN acceleration, in which all convolutions are reinterpreted in terms of the matrix multiplication of the input tensor by a suitable Toeplitz matrix materialized just-in-time within the device.

GEMM can be very easily mapped to FPGA with a 2D systolic array. But GEMM is not universally suitable as a primitive. For instance, an RNN accelerator deployed in the field does not have the benefit of batching, and its matrix-matrix linear operations therefore involve a "matrix" of a single column - a vector. Complications such as strided or dilated (sparse) convolutions [20] mean that efficiently materializing the correct matrices can be a difficult task. It is of interest that a large GEMM primitive generally implies streaming of whole tensors to and from external memory, precisely what we decry as energy-wasteful in Section I.

Therefore, a slightly different approach is adopted in this paper. Conceding that GEMM is not always ideal, we will instead adopt the BLAS Level 2 operation GEMV (GEneral Matrix-Vector product) as our basic linear operation. Riffing on [12], [13], we will perform GEMV products with multi-bit operands bit-serially, yet the GEMV operation as a whole is parallelized using bit-slicing. Multiple independent GEMV units can be instantiated, and can provide extra, inter, or intra-layer parallelism; We will call these independent units *Matrix-Vector Units*. These MVUs communicate over an interconnect, sending messages containing the data vector they're operating upon.

A. Matrix-Vector Unit

The Matrix-Vector Unit (MVU) is at the heart of the design. Each MVU performs one $n \times n$-element by n-element matrix-vector multiplication, where both the matrix and vector are binary (their elements are single bits). The result of such a product is a new n-element vector of $\sim \log_2 n$ bits each that is then accumulated into an n-wide vector accumulator register every clock cycle.

An MVU also integrates a combined max-pooling and Rectified Linear UNit (ReLU) at the accumulator output to downsample and reduce the bitwidth of the accumulator to binary or ternary precision, fit for storage or sending over the interconnect.

The matrix-vector product primitive suffices to implement almost all common linear operations in a neural network:

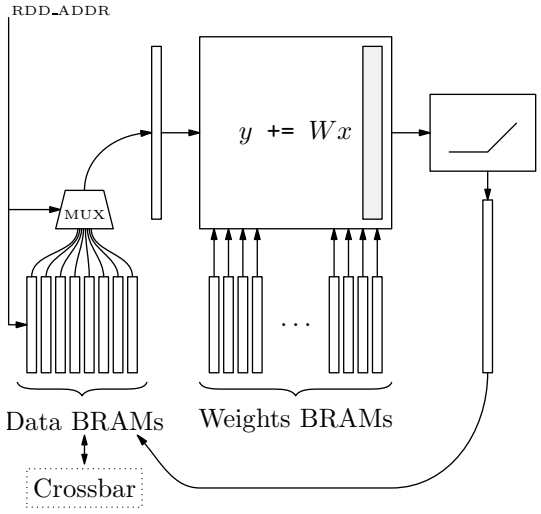

Fig. 1. MVU

matrix-vector products, matrix-matrix products, 1D, 2D and 3D convolutions with or without striding and/or dilation, and at arbitrary precision using *bit slicing*.

B. Bit Slicing

Judd [21] introduced a strategy allowing neural network accelerators to use arbitrary-precision : bit-serial multiplication. We make use of a similar strategy in our design.

Any arbitrary-precision r-bit by s-bit scalar multiplication can be performed bit-serially by a sequence of rs 1-bit-by-1-bit multiplications (AND operations), shifts and accumulations. We extend this to arbitrary-precision $n \times n$ matrix-vector operations by performing, over the course of rs steps, rs 1-bit-by-1-bit, $n \times n$ matrix-vector products, and likewise accumulating and shifting their result to produce a high-precision matrix-vector product.

In order to reduce the hardware requirement to just an adder and a 1-bit left-shifter, we adopt a specific *order of encounter* for each bit product, namely a zig-zag from the most-significant products to the least-significant products, as illustrated in Figure 2. The zig-zag pattern begins at the product of the MSB by the MSB, then all bit-products of the same weight are performed before shifting left the accumulator by 1 bit and moving on to the next-lower-weight set of products, finishing at the product of the LSB by the LSB.

If the bitwidth of the weights is w bits and the bitwidth of the data is d bits, the cost of bit-serial multiplication is precisely wd, and scales linearly in the precision of both operands. This allows fine-grained exploration of the precision-performance trade-off, and dynamic adaptation of precision to every layer's needs.

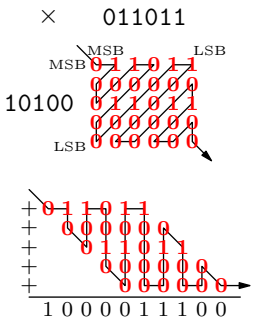

Fig. 2. Scalar Bit-Serial Product. Zig-Zag Order of Encounter.

C. Data and Weights Layout

To exploit bit-slicing, the data vectors and weights matrix tiles in each MVU are stored in striped fashion. All most-significant-bits are stored and processed together, as are all second-most-significant bits, ...and so on till the least-significant bits.

Data vectors are $2n$ bits large, allowing for n ternary or binary elements. For the purposes of quantized CNN and RNN, element i of the data vector belongs to feature map $i \pmod n$ of the convolutional or fully-connected layer. All elements also share the same spatial coordinates within the tensor; For instance, when implementing 2D convolution, all elements of the same vector share the same (x, y) coordinates. Data vectors are relatively small, and so can be communicated to other MVUs.

Data vectors are spread across 32 *data banks*. Each bank supports one read and one write simultaneously; Multiple simultaneous reads or writes are handled by deconfliction logic with a static priority scheme.

Weights tiles are $n \times n$ bits large and binary-only. For the purposes of quantized CNN, they map a subset of n input feature maps to n output feature maps. For RNN and GEMM, they are simply an $n \times n$ tile of the matrix product being implemented. This organization allows data vectors to still represent a single spatial coordinate after matrix-vector multiplication. Weights tiles are static to the MVU and do not move, on account of their large size.

D. Interconnect

More than one MVU may be instantiated in a design, if resources allow. This allows multiple MVUs to work collaboratively on the same, or different, layers. To allow data vector message-passing between independent MVUs, a fast yet flexible interconnect is required, both at prototyping time and at deployment time. It is desirable that it blocks either rarely or never, and that it supports multicast and broadcast of data vectors.

1) Crossbar: The simplest and most common interconnect that supports this is the basic *crossbar* interconnect, which links every MVU to every other MVU. For N MVUs, this leads to N^2 *crosspoints*, and therefore the crossbar's logic cost scales as $O(N^2)$ and its delay scales as $O(\log_2 n)$.

Nevertheless, for a reasonably small number of MVUs, the crossbar remains an acceptable solution. On a Stratix V GX A7, up to $N = 8$ MVUs of size 64×64 can be simultaneously synthesized, resulting in a tolerable $8 \times 8 = 64$ crosspoints.

2) Crosspoint Pruning: When deploying a neural network onto the MVUs, it may be possible to determine ahead-of-time that a particular neural network will never exercise a particular MVU-MVU crosspoint. If so, it can be pruned away from the crossbar at design compile time. If each MVU communicates with only one or two other MVUs in a ring/star topology (as is common for feedforward networks with few skip connections), this reduces the cost of the interconnect to $O(n)$.

III. Neural Network Implementation

A. Linear operations

The use of matrix-vector products and our data layout allows us to implement all common varieties of convolution, as well as classic GEMM, by tiling them into their constituent matrix-vector products. Convolutions of any width or height, subsampled and/or dilated (a-trous), are supported through appropriate indexing of the data banks, which may be done in software.

B. Non-Linear operations

Extreme quantization simplifies somewhat the choice of non-linearity. In BNNs, non-linearities are not necessary; Mere thresholding against zero suffices. For less severely-quantized neural networks, the ReLU family offers a good compromise between complexity and performance. We adopt PACT [22], which has a superset of ReLU's capabilities and possesses a learned upper bound α.

C. Data Flow

An FPGA's on-chip memory is too limited to store entirely the data tensors that result from CNN processing. But spilling and reloading the tensors from off-chip memory is undesirable, and severely limited by the available bandwidth. A strategy to reduce the on-chip BRAM memory use is to process tensor data slice-by-slice, retaining only a local neighbourhood around the current location, and performing multiple layers simultaneously. This strategy reduces the memory cost of filtering an image of width w and height h from $O(wh)$ to $O(wF_H)$, where F_H is the height in taps of the aforementioned filter. It also reuses on-chip data for further processing, rather than spilling it.

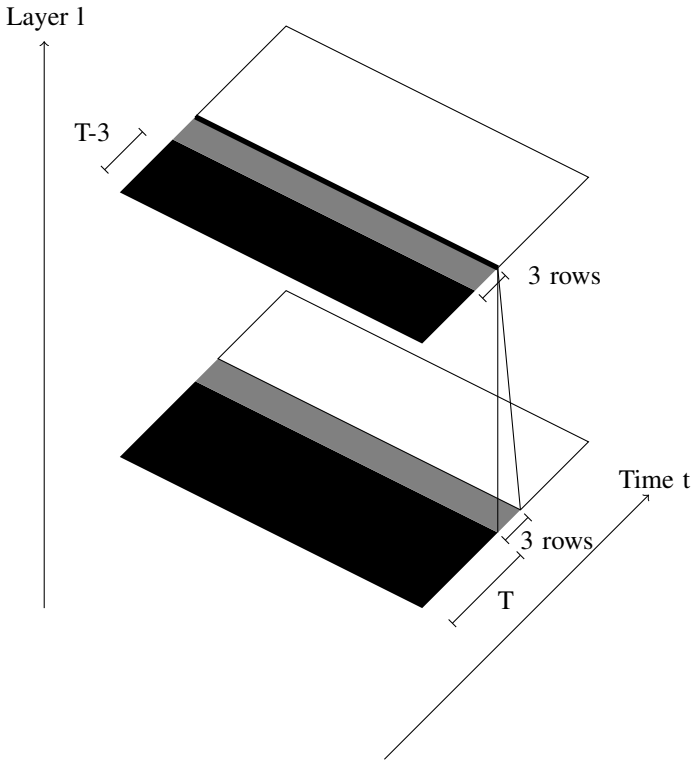

Fig. 3. Data Flow

Resource	Usage
ALM	157929 (67%)
BRAM	32 Mbit (64%)
DSP Slices	0 (0%)
Frequency	250 MHz
Power*	20.883 W

Fig. 4. Resources for eight-MVU 64×64 design. *Toggle rate 25%

Unit	#	ALM
Interconnect	1	3096
MVU	8	19000
Dot-Product	64	12000
Bank Conflict Resolvers	32	5400
Accumulator	64	1300
Max-Pooling	64	2250

Fig. 5. Breakdown of logic usage per unit

D. Layer Mapping to MVUs

In order to make best use of the parallel processing capabilities of the MVUs and to minimize the traffic to/from the limited external memory interface, while accounting for the locality of the weights to individual MVUs, it is desirable to assign the weights of sequential layers of a neural network to consecutive MVUs. In the case of CNNs, this allows all MVUs to begin processing their respective image rows of data as soon as they become available, preventing them from idling.

Additionally, if the CNN is purely or approximately feed-forward, mapping its layers in a round-robin fashion results in a roughly even distribution of weights between all MVUs, and only a lightweight ring interconnect is required.

IV. Results

We simulated the design using Altera Quartus Prime 15.1. It was also synthesized, placed and routed for a target Altera Stratix V GX A7 FPGA. Power consumption predicted by Quartus is approximately 20.883W for an 8-MVU design, as seen in Figure 4. At 250 MHz, the total throughput is 8.2 binary-ternary TMAC/s. This is to be compared with an NVIDIA P100 GPU's 300W for 10 single-precision TFLOP/s.

V. Conclusion

We have proposed a novel, compact, parametrizable power-efficient neural network architecture for quantized neural networks that has several desirable properties. Its performance scales dynamically with the precision of both the neural network's weights and its activations, reaching maximum performance for BinaryNet acceleration (1-bit weight, 1-bit activation), through use of bit-slicing. The architecture is scalable, since as many MVUs as desired within the constraints of the target FPGA may be synthesized, and the neural network's layers partitioned among them. A well-chosen data layout allows for a streaming data-flow that minimizes spills and reloads from off-chip memory, reducing power consumption. On Stratix V GX A7, eight 64×64 MVUs may carry out 250K ternary-binary matrix-vector operations per second, or 8.2 TMAC/s, using 20.883 Watts, or 392 GMAC/s/W.

Acknowledgment

The authors would like to thank IBM and the SOSCIP program for providing access to an FPGA development environment and support, and COHESA and Samsung for other support and funding. During the course of this research, GPUs from Compute Canada's Cedar & Graham clusters were made use of. A special thank-you to Brian Kingsbury of IBM for a very important conversation that led to the abandonment of a previous approach in favour of the one presented in this paper.

References

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

[2] C. Zhu, S. Han, H. Mao, and W. J. Dally, "Trained ternary quantization," in *International Conference on Learning Representations*, 2017.

[3] G. Venkatesh, E. Nurvitadhi, and D. Marr, "Accelerating deep convolutional networks using low-precision and sparsity," *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Mar 2017. [Online]. Available: http://dx.doi.org/10.1109/ICASSP.2017.7952679

[4] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Quantized neural networks: Training neural networks with low precision weights and activations," 2016.

[5] M. Horowitz, "1.1 computing's energy problem (and what we can do about it)," in *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2014 IEEE International*. IEEE, 2014, pp. 10–14.

[6] G. Lacey, G. W. Taylor, and S. Areibi, "Deep learning on fpgas: Past, present, and future," *arXiv preprint arXiv:1602.04283*, 2016.

[7] E. Nurvitadhi, D. Sheffield, J. Sim, A. Mishra, G. Venkatesh, and D. Marr, "Accelerating binarized neural networks: Comparison of fpga, cpu, gpu, and asic," in *2016 International Conference on Field-Programmable Technology (FPT)*, Dec 2016, pp. 77–84.

[8] A. Mishra, E. Nurvitadhi, J. J. Cook, and D. Marr, "WRPN: Wide reduced-precision networks," in *International Conference on Learning Representations*, 2018. [Online]. Available: https://openreview.net/forum?id=B1ZvaaeAZ

[9] Y. Umuroglu, N. J. Fraser, G. Gambardella, M. Blott, P. Leong, M. Jahre, and K. Vissers, "Finn: A framework for fast, scalable binarized neural network inference," in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. ACM, 2017, pp. 65–74.

[10] J. Wang, Q. Lou, X. Zhanng, C. Z. Zhu, Y. Lin, and D. Chen, "Design flow of accelerating hybrid extremely low bit-width neural network in embedded fpga," in *International Conference on Field-Programmable Logic and Applications (FPL)*, 2018.

[11] X. Zhang, J. Wang, C. Zhu, Y. Lin, J. Xiong, W.-m. Hwu, and D. Chen, "Dnnbuilder: an automated tool for building high-performance dnn hardware accelerators for fpgas," in *International Conference on Computer Aided Design (ICCAD)*, 2018.

[12] S. Sharify, A. D. Lascorz, K. Siu, P. Judd, and A. Moshovos, "Loom: Exploiting weight and activation precisions to accelerate convolutional neural networks," in *Proceedings of the 55th Annual Design Automation Conference*. ACM, 2018, p. 20.

[13] Y. Umuroglu, L. Rasnayake, and M. Sjalander, "Bismo: A scalable bit-serial matrix multiplication overlay for reconfigurable computing," in *Field Programmable Logic and Applications (FPL), 2018 28th International Conference on*, ser. FPL '18, 2018.

[14] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Neural Information Processing Systems*, vol. 25, 01 2012.

[15] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *International Conference on Learning Representations*, 2015.

[16] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun 2016. [Online]. Available: http://dx.doi.org/10.1109/CVPR.2016.90

[17] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun 2015. [Online]. Available: http://dx.doi.org/10.1109/CVPR.2015.7298594

[18] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," in *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, 2011, pp. 315–323.

[19] S. Chetlur, C. Woolley, P. Vandermersch, J. Cohen, J. Tran, B. Catanzaro, and E. Shelhamer, "cudnn: Efficient primitives for deep learning," *arXiv preprint arXiv:1410.0759*, 2014.

[20] V. Dumoulin and F. Visin, "A guide to convolution arithmetic for deep learning," *ArXiv e-prints*, mar 2016.

[21] P. Judd, J. Albericio, T. Hetherington, T. M. Aamodt, and A. Moshovos, "Stripes: Bit-serial deep neural network computing," in *Microarchitecture (MICRO), 2016 49th Annual IEEE/ACM International Symposium on*. IEEE, 2016, pp. 1–12.

[22] J. Choi, Z. Wang, S. Venkataramani, P. I.-J. Chuang, V. Srinivasan, and K. Gopalakrishnan, "Pact: Parameterized clipping activation for quantized neural networks," *arXiv preprint arXiv:1805.06085*, 2018.