WP514 (v1.0) October 28, 2019

FPGAs in the Emerging DNN Inference Landscape

FPGAs can play a fundamental role in adjusting compute architecture to the specifics of a given neural network topology, providing the functionality required to adapt the device to the customer's exact environment.

ABSTRACT

Reflecting on the differences in application requirements, a key trend emerging in both DNN inference workloads and hardware accelerator architectures is their diversity and rapid evolution. This white paper provides an overview of recent developments in regards to both algorithms and architectures and takes a look at how FPGAs fit into this changing landscape.

Introduction

At the heart of the current industrial revolution is the roll-out of machine learning (ML) algorithms, specifically deep neural networks (DNNs). They achieve impressive results in computer vision and speech recognition, and are increasingly being adopted for other tasks. DNNs are first trained on a labeled dataset, and afterwards can be used for inference on previously unseen data as part of an application. The latter is also often referred to as deployment and is the main focus of this white paper.

The large compute and storage requirements associated with DNN deployment necessitate acceleration. Furthermore, different constraints might be imposed on accuracy, cost, power, model size, throughput, and latency depending on the use case. Real-time and safety-critical applications such as augmented reality, drone control, and autonomous driving are not suitable for offloading to the cloud due to low-latency requirements and data transmission overhead. In cloud-computing and ML-as-a-Service contexts, data centers face ever-increasing throughput requirements to process astronomical scales of data, bringing additional challenges in energy efficiency to minimize operating expenses. While cloud service latency is less critical compared to embedded scenarios, it still translates directly into customer experience for interactive applications. For instance, Jouppi et al. [Ref 1] quote a response time limit of 7ms for an interactive user experience in cloud-based services.

Due to these challenges, a virtual Cambrian explosion of different DNN models and accelerators has taken place in the past few years. Reflecting the differences in application requirements, a key trend emerging in both DNN inference workloads and hardware accelerator architectures is their diversity. This white paper provides an overview of recent developments in both aspects and takes a look at how FPGAs fit into this changing landscape.

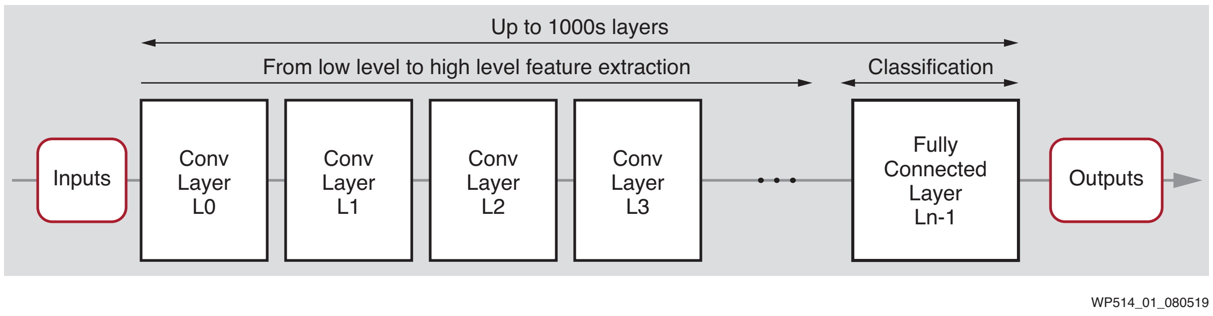

Figure 1: **Basic Convolutional Neural Network Topology**

A Quick Introduction to Deep Neural Networks

DNNs are typically feed-forward computational graphs constructed from one or more layers, where large networks can contain hundreds to thousands of layers. Each layer consists of neurons interconnected with synapses, each associated with a weight. Each neuron computes a weighted sum of its receptive field, followed by a non-linear activation function. For computer vision, convolutional layers are commonly used, where the receptive field extends over multiple typically 2-dimensional feature maps that are convolved with multiple typically 2-dimensional filters. The pseudo-code for the resulting computation is displayed in Figure 2.

```
for each layer l
 for each output feature map plane o
  for each output feature map row r
   for each output feature map col c
    for each filter plane p
     for each filter row x
      for each filter col y
       A[l][o][r][c] += A[l-1][p][r-x][c-y] * w[l][o][p][x][y]
```

WP51_02_091919

Figure 2: **Pseudo-code for Typical DNN Computation**

Machine Learning (ML) frameworks such as PyTorch, Tensor Flow, and Caffe work on representations based on these computational graphs, scheduling and mapping the computation onto hardware for both training and inference.

Trends in DNN Models for Inference

Traditionally, machine learning research was focused on improving the accuracy of the models without particular regard to the cost of inference. This is evident in the older ImageNet-winner networks like AlexNet and VGG, which are now considered large and over-parametrized [Ref 2]. However, as machine learning and DNNs move into practical applications, compute and memory requirements become a major concern. This has motivated a flurry of recent research on how DNN inference can be made more efficient, taking both the accuracy and computational complexity into account.

Methods for Creating More Efficient DNNs

This white paper provides a brief overview of several types of methods proposed to make DNNs more efficient. For the most part, these methods can be considered orthogonal to each other and can be combined, although some DNNs might be less amenable to certain techniques.

Efficient Topologies

The topology of a DNN defines how many layers it contains, the type and size of each layer, and how the layers are connected. Some topologies are defined by a construction rule that defines their size and number of layers according to a topology parameter. A large volume of recent research has proposed DNN topologies that achieve high accuracy with a compact topology—i.e., a small number of parameters, a small number of multiply-accumulate (MAC) operations, or both. Recent examples include MobileNets [Ref 3], ShiftNet [Ref 4], ShuffleNet [Ref 5], and Deep Expander Networks [Ref 6]. These typically have a topology parameter that controls accuracy/computation trade-offs. FPGAs can provide unique advantages here, as new types of operators—such as shift and shuffle, for example—require almost zero compute resources, because they can be implemented through reconfiguration of the programmable interconnect in the device.

Quantization

DNNs are commonly trained with floating-point arithmetic, but they can be made to use a limited set of values. They are typically amenable to direct quantization down to 8 bits (see Figure 3), or retrained to use even fewer bits (trained quantization), yielding quantized neural networks (QNNs). The quantization scheme can be uniform or non-uniform, and different quantization schemes can be used for different parts of the network. Using fewer bits requires much less compute and memory, but might result in less accuracy. Many recent publications [Ref 7] [Ref 8] [Ref 9] have proposed better quantized training techniques. Recent methods such as LQ-Nets [Ref 10] have reduced the accuracy gap between floating point and 4-bit QNNs to less than 1%.

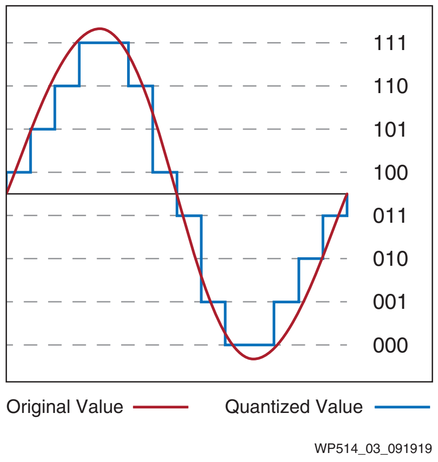

Figure 3: **3-Bit Quantization Function of a Sine Wave**

The programmable logic inside an FPGA offers the unique capability to customize the resources of an arithmetic operator with very fine granularity, to exact numbers of bits that are required by the application. Thereby, the application can exploit the potential to reduce the hardware cost for the compute along with the memory.

Pruning

Parts of a neural network can be pruned away without any significant impact on accuracy, as much as 90% for some layers [Ref 11]. See Figure 4. Pruning techniques differ in how the pruned parts are chosen (e.g., by weight magnitude or second-order derivatives), and in the granularity at which pruning is performed (e.g., individual synapses, adjacent groups of synapses, or entire feature maps for convolutions). Pruning individual synapses results in irregular structures that can only be processed efficiently by specialized hardware [Ref 12]. While coarser-granularity pruning is typically preferred, finely granular pruning offers further performance scalability that can be exploited with FPGAs, tailoring memory subsystems to efficiently store sparse representations while supporting the implementation of computing engines.

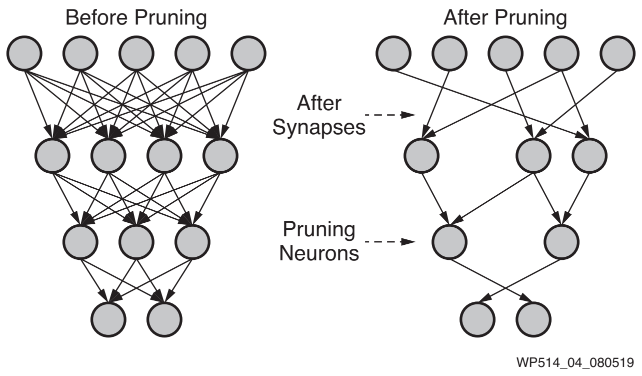

Figure 4: **Synaptic Pruning**

Layer Fusion and Decomposition

Mathematical equivalences or approximations can be used to decrease the compute and memory requirements of DNN layers. For instance, batch normalization operations can be fused into the preceding linear transform layer (convolutional or fully connected). Convolutions can be approximated by depth-wise separable filters [Ref 13], fully-connected layers by singular value decomposition [Ref 14].

Other techniques: Knowledge distillation [Ref 15] can be used to make the training of efficient models easier. Hoffer et al. [Ref 16] propose to fix the final classification layer of a DNN to a Hadamard matrix with a fixed pattern of +1/-1 values, and show that this results in no loss of accuracy for several ImageNet networks.

Accuracy-Computation Trade-Offs: Examples from Quantization

Neural networks are function approximators, and high-quality approximators cost more than low-quality ones. Specifically, a trade-off exists between how much memory and compute resources are required to perform inference with a neural network, and the quality of the resulting inference (e.g., how accurately the network predicts the class of an unseen input image). Although the exact relationship between resources and accuracy is difficult to determine, the design space can be empirically explored by training neural networks that have different compute requirements and observing the obtained accuracy (Figure 5).

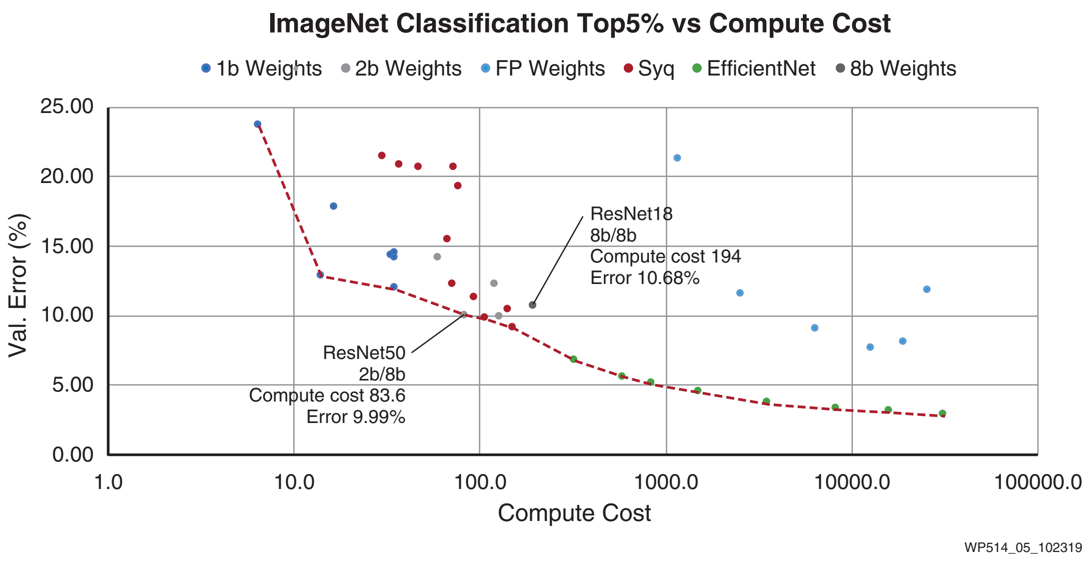

Figure 5: **Compute Cost vs. Top-5 Classification Error on ImageNet Using a Variety of Quantized Networks**

Figure 5 illustrates the results of one such design space exploration. (In Figure 5, lower is better for both axes.) Different quantization schemes are used to produce networks of different compute cost (x-axis, expressed as a number roughly indicating how many FPGA LUTs and DSP slices would be used for the entire computation) and accuracy (y-axis). The red line is the Pareto frontier, containing the design points, which are best-in-class for compute cost and accuracy. In this case, a lower-precision deep network (ResNet-50 with 2-bit weights, 8-bit activations) outperforms a higher-precision shallow network (ResNet-18 with 8-bit weights, 8-bit activations), both in terms of lower compute cost and lower error rate.

Trends in Inference Accelerator Architectures

As mentioned earlier, compute and memory requirements in neural networks can be very large; for instance, image classification with a popular DNN such as ResNet-50 requires 7.7 billion operations for every single input image. However, on the upside, DNNs are highly parallel in nature, which can be exploited. As such, numerous forms of customized hardware architectures are evolving to make the deployment of these algorithms a reality.

The inference computation for a DNN contains multiple levels of parallelism, as illustrated in Figure 6. These can be summarized as follows:

- Coarse-grain topology parallelism between consecutive layers, and parallel branches, such as those found in GoogLeNet or in DNN ensembles
- Neuron and synapse parallelism inside a layer, such as multiple input/output feature map (IFM/OFM) channels and pixels in convolutional layers
- Bit-level parallelism inside arithmetic, when individual bits of weights and activations are viewed separately

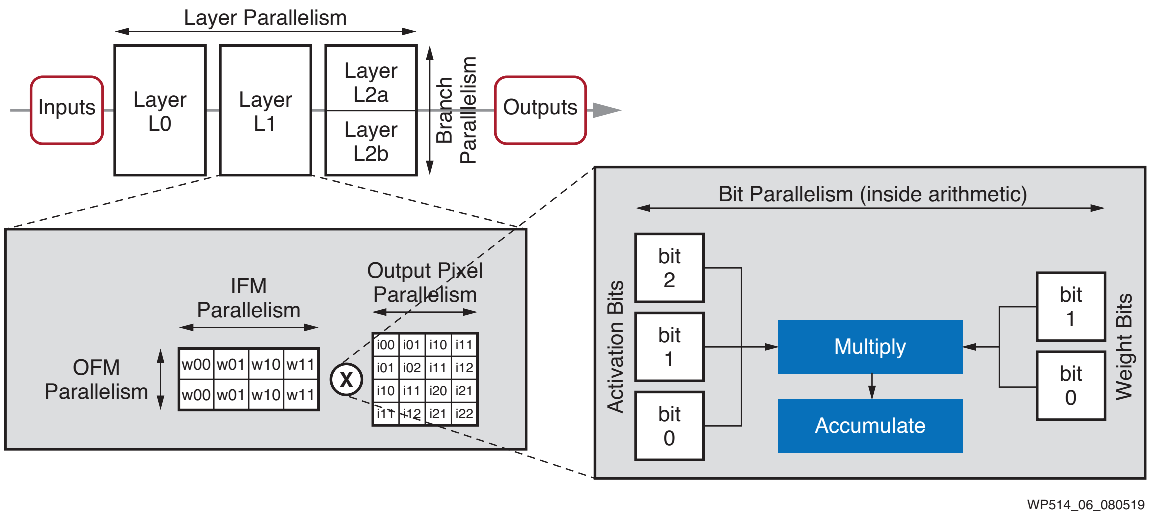

Figure 6: **Levels of Parallelism Available for Computing DNN Inference**

The Landscape of Inference Accelerator Architectures

When optimizing a hardware architecture for these highly compute and memory intensive algorithms, the following questions arise:

- How is this best loop-transformed and unfolded to maximize data reuse and compute efficiency to minimize memory bottlenecks?
- How can performance scalability be provided with limited gains coming from shrinking technology nodes?
- And how can real-time responses be achieved, limiting power consumption to enable deployment within embedded application scenarios where energy consumption is at a premium?

Besides standard CPUs, numerous specialized hardware architectures are trying to optimize these applications for specific application constraints, including GPUs, FPGAs, and AI ASICs. Microsoft has coined the term "DNN Processing Unit," [Ref 16] or DPU for short, as an umbrella term for these customized architectures. Figure 7 depicts a generic DPU architecture, with typical "pain points" highlighted in red.

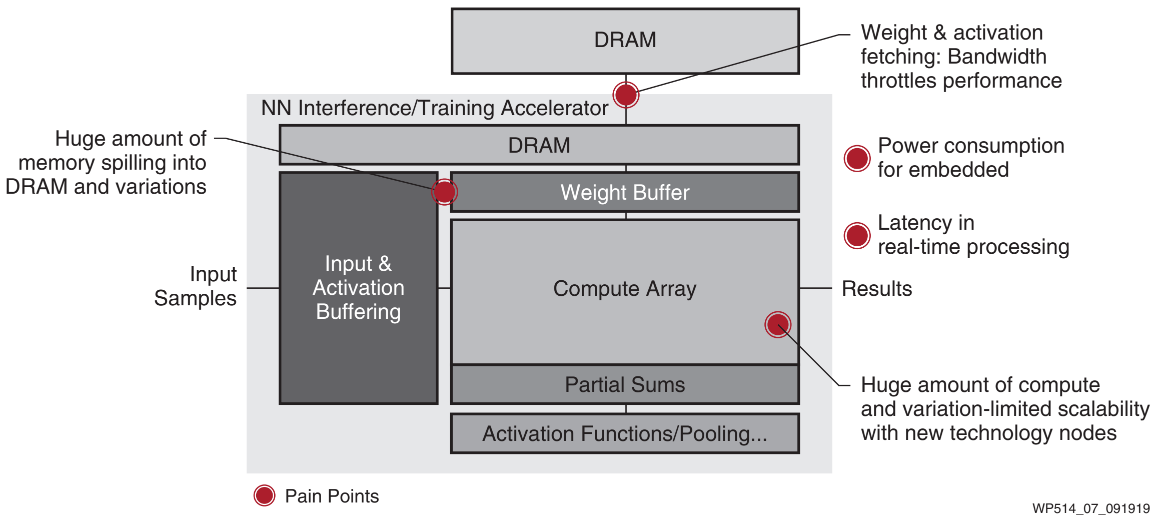
DRAM
Weight & activation fetching: Bandwidth throttles performance
NN Interference/Training Accelerator
DRAM
Huge amount of memory spilling into DRAM and variations
Power consumption for embedded
Weight Buffer
Latency in real-time processing
Input Samples
Input & Activation Buffering
Compute Array
Results
Partial Sums
Huge amount of compute and variation-limited scalability with new technology nodes
Activation Functions/Pooling...
● Pain Points
WP514_07_091919

Figure 7: **Typical “Pain Points” in a Generic DPU Architecture**

Architectures can broadly be classified by the basic type of compute operation, memory bandwidth, level of parallelism, degree of specialization, and inherent precision support. While GPUs initially had a different focus—targeting gaming and graphics processing, and are gradually being adopted for high-performance computing—they also have an increasing focus for AI and are the *de facto* standard for training acceleration. GPUs are considered to be vector-based SIMD processors, increasingly customized for Deep Learning with introduction of Tensor Cores in Nvidia’s Volta family [Ref 17] and fixed-point integer arithmetic, specifically INT4 and INT8 with Nvidia’s most recent Turing architectures [Ref 18]. Custom ASIC implementations of DPUs aim to minimize hardware cost and maximize performance, such as Google's Tensor Processing Unit (TPU) [Ref 19]. The TPU operates specifically on tensors rather than just vectors, and boasts customized memory hierarchies and customized arithmetic to take advantage of quantization, as described earlier. Besides the TPU, an increasing number of companies are building custom hardware, including Arm, Intel’s Nervana, MobilEye, and Movidius acquisitions, and numerous startups such as GraphCore, Cerebras, Groq, and Wave Computing, to name a few. In summary, the landscape is rapidly changing.

FPGA Implementation Advantages for Efficient DNNs

The diversity of DNNs is also reflected in how much parallelism is available at each level mentioned earlier. Thus, any fixed hardware architecture that instantiates a fixed number of parallel compute elements communicating in a fixed fashion has limitations as to how efficiently it can execute a DNN. For instance, if a fixed architecture is built to take advantage of input feature map and output feature map (IFM-OFM) parallelism, it might exhibit low utilization for depth-wise separable convolutions [Ref 2]. Especially considering the rapidly-advancing techniques for creating efficient DNNs, adaptability is key to staying efficient in the changing DNN inference landscape.

In this context, the key advantage of Xilinx FPGAs is the adaptable, fine-grained, massively parallel nature of the compute and memory resources offered. The Xilinx devices facilitate a wide range of DPU architectures that can take advantage of the multiple levels of parallelism mentioned and tailor to the specific requirements of a given DNN topology and the application-specific design constraints. Soft DPUs implemented on FPGAs can support any of the above configurations with explicitly managed memory and arithmetic customized for every specific neural network.

Examples of Soft DPUs

Figure 8, Figure 9, and Figure 10 provide examples of Soft DPUs that illustrate the diversity of possible Soft DPU architectures. The key characteristics of each architecture are as follows:

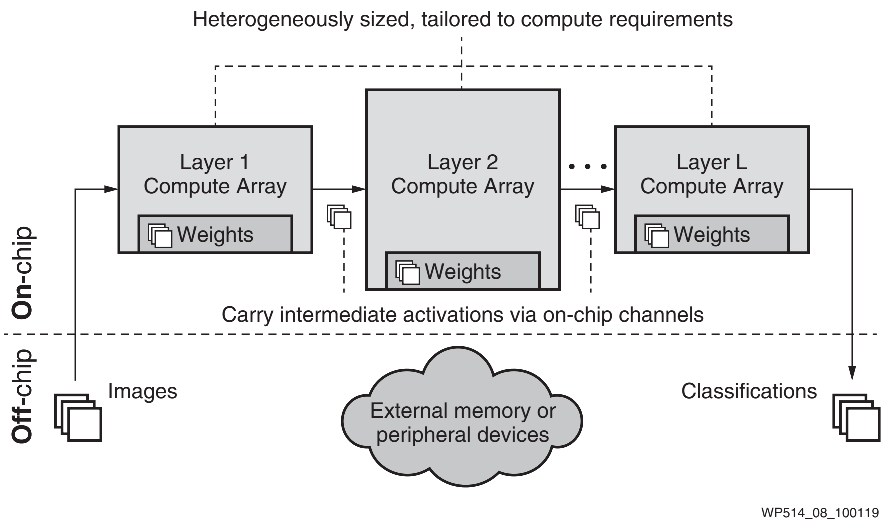
Heterogeneously sized, tailored to compute requirements
Layer 1 Compute Array
Weights
Layer 2 Compute Array
Weights
Layer L Compute Array
Weights
Carry intermediate activations via on-chip channels
On-chip
Off-chip
Images
External memory or peripheral devices
Classifications
WP514_08_100119

Figure 8: **FINN, a Soft DPU with Dedicated Per-layer Compute Resources and On-chip Data Flow between Layers**

- **Custom Architecture, Multi-layer Parallelism**

  For a given QNN, FINN [Ref 20] [Ref 21] generates a custom DPU where each layer is implemented with dedicated hardware and connected to the next layer with an on-chip channel, subject to device size limitations. This enables tailoring the precision and compute resources for each layer, resulting in an efficient design. Data flow parallelism between layers can be beneficial for achieving low latency and high throughput. FINN is available as open-source [Ref 22].

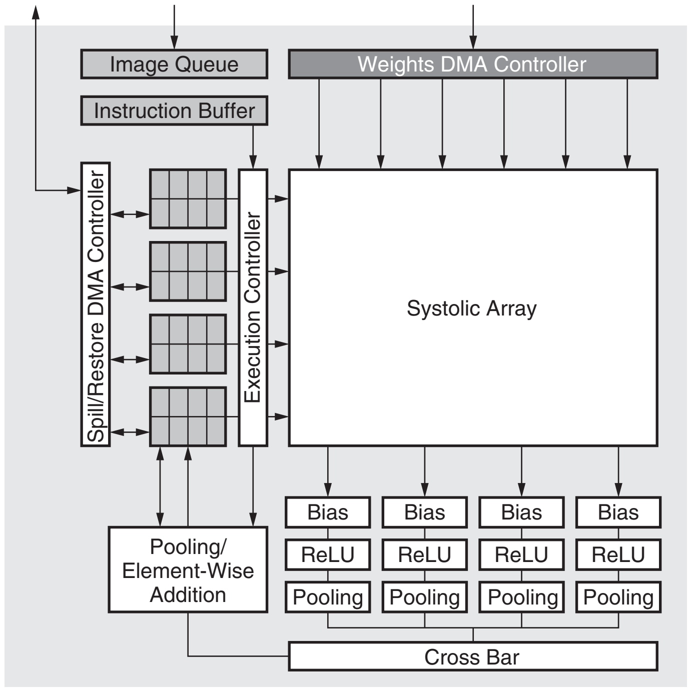

Figure 9: **xDNN, a Soft DPU with a High Degree of Programmability and Performance Optimization**

- **Programmable Generic Architecture, Single-Layer Parallelism, Fixed Precision**

  xDNN [Ref 23] is a programmable overlay architecture with a fixed-precision systolic array. The regular structure of this array enables a high degree of performance optimization. A tool flow is provided to map any DNN to this architecture, eliminating the need for new bitstream generation or FPGA expertise. xDNN is available for evaluation [Ref 24].

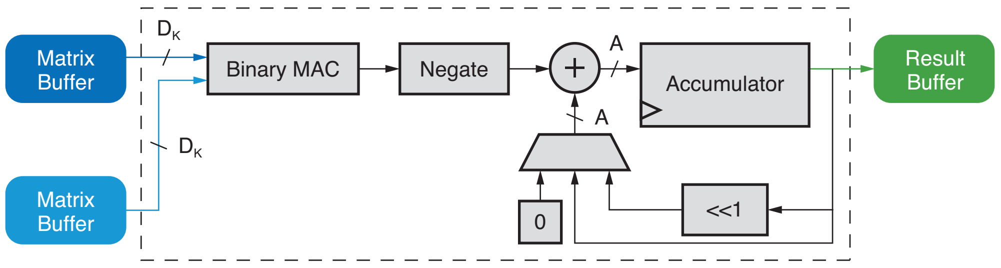

Figure 10: **BISMO, a Soft DPU that Supports Different Arithmetic Precisions without Reconfiguration**

- **Programmable Generic Architecture, Single-Layer Parallelism, Runtime-Variable Precision**

  BISMO [Ref 25] [Ref 26] is a programmable overlay for bit-serial matrix multiplication. By serializing the bit precision dimension but parallelizing across the other dimensions, it offers a fixed architecture that can take advantage of variable precision at runtime while still offering high performance. Higher-precision layers take more clock cycles to execute. BISMO is available as open-source [Ref 27].

Driven by the end of Moore’s law and in line with general trends of the semiconductor industry, Xilinx has developed increasingly specialized devices for specific vertical markets. This is achieved by a novel compute fabric for AI, consisting of software programmable AI engines with a custom instruction set. In addition, NoC-based interconnectivity provides a new level of flexibility in routing resources, which is vital to achieving high device utilization. Furthermore, beyond the neural network itself, FPGAs can offer sensor fusion and flexible I/O; they can add computer vision pre- and post-processing; and they can provide other functionality required to integrate as intelligence-on-the-wire and adapt the device perfectly to the customer’s environment.

Summary

The increasing adoption of machine learning algorithms to an ever-growing spectrum of applications creates an immense computational burden on traditional compute architectures. The semiconductor industry rises to the challenge through many innovative architectures code-named DPUs. FPGAs can play a fundamental role, offering the ultimate flexibility in adjusting compute architectures not only to machine learning tasks in general, but to the specifics of given neural network topologies. The programmable devices offer customized arithmetic to minimize storage and compute resources, provide further performance scalability or optimization for stringent latency requirements. Finally, FPGAs can provide flexibility in the I/O and sensor fusion, as well as pre- and post-processing for computer vision like no other device types. This combination of features allows the FPGA to adapt perfectly to the customer's requirement.

References

1. N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, et al., “In-Datacenter Performance Analysis of a Tensor Processing Unit,” in *Computer Architecture (ISCA), 2017 ACM/IEEE 44th Annual International Symposium*, 2017.
2. Y.-H. Chen, J. Emer, and V. Sze, “Eyeriss v2: A Flexible and High-Performance Accelerator for Emerging Deep Neural Networks,” arXiv preprint arXiv:1807.07928, 2018.
3. A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, “Mobilenets: Efficient Convolutional Neural Networks for Mobile Vision Applications,” arXiv preprint arXiv:1704.04861, 2017.
4. B. Wu, A. Wan, X. Yue, P. Jin, S. Zhao, N. Golmant, A. Gholaminejad, J. Gonzalez, and K. Keutzer, “Shift: A Zero FLOP, Zero Parameter Alternative to Spatial Convolutions,” arXiv preprint arXiv:1711.08141, 2017.
5. X. Zhang, X. Zhou, M. Lin, and J. Sun, “ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices. arXiv 2017,” arXiv preprint arXiv:1707.01083, 2017.
6. A. Prabhu, G. Varma, and A. Namboodiri, “Deep Expander Networks: Efficient Deep Networks from Graph Theory,” arXiv preprint arXiv:1711.08757, 2017.
7. A. Mishra, E. Nurvitadhi, J. J. Cook, and D. Marr, “WRPN: Wide Reduced-Precision Networks,” arXiv preprint arXiv:1709.01134, 2017.
8. J. Faraone, N. Fraser, M. Blott, and P. H. W. Leong, “SYQ: Learning Symmetric Quantization For Efficient Deep Neural Networks,” in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2018.
9. J. Choi, P. I.-J. Chuang, Z. Wang, S. Venkataramani, V. Srinivasan, and K. Gopalakrishnan, “Bridging the Accuracy Gap for 2-bit Quantized Neural Networks (QNN),” arXiv preprint arXiv:1807.06964, 2018.

10. D. Zhang, J. Yang, D. Ye, and G. Hua, "LQ-Nets: Learned Quantization for Highly Accurate and Compact Deep Neural Networks," arXiv preprint arXiv:1807.10029, 2018.
11. S. Han, H. Mao, and W. J. Dally, "Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding," arXiv preprint arXiv:1510.00149, 2015.
12. S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally, "EIE: efficient inference engine on compressed deep neural network," in *Computer Architecture (ISCA), 2016 ACM/IEEE 43rd Annual International Symposium*, 2016.
13. J. Guo, Y. Li, W. Lin, Y. Chen, and J. Li, "Network Decoupling: From Regular to Depthwise Separable Convolutions," arXiv preprint arXiv:1808.05517, 2018.
14. E. L. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus, "Exploiting Linear Structure within Convolutional Networks for Efficient Evaluation," in *Advances in Neural Information Processing Systems*, 2014.
15. G. Hinton, O. Vinyals, and J. Dean, "Distilling the Knowledge in a Neural Network," arXiv preprint arXiv:1503.02531, 2015.
16. E. Hoffer, I. Hubara, and D. Soudry, "Fix Your Classifier: The Marginal Value of Training the Last Weight Layer," arXiv preprint arXiv:1801.04540, 2018.
17. Nvidia, *Nvidia Volta and Tensor Core GPU Architecture*. Available: https://www.nvidia.com/en-us/data-center/volta-gpu-architecture/.
18. Nvidia, *Nvidia Turing*. Available: https://www.nvidia.com/en-us/geforce/turing/.
19. Google, *Cloud TPU*. Available: https://cloud.google.com/tpu/.
20. 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*, 2017.
21. M. Blott, T. Preusser, N. Fraser, G. Gambardella, K. O'Brien, and Y. Umuroglu, "FINN-R: An End-to-End Deep-Learning Framework for Fast Exploration of Quantized Neural Networks," arXiv preprint arXiv:1809.04570, 2018.
22. Xilinx Research Labs, *Machine Learning on Xilinx FPGAs with FINN*, [Internet]. Available: https://xilinx.github.io/finn.
23. Xilinx White Paper WP504, *Accelerating DNNs with Xilinx Alveo Accelerator Cards*, 2018.
24. Xilinx, *Xilinx ML Suite*, [Internet]. Available: https://github.com/Xilinx/ml-suite.
25. Y. Umuroglu, L. Rasnayake, and M. Sjalander, "BISMO: A Scalable Bit-Serial Matrix Multiplication Overlay for Reconfigurable Computing," arXiv preprint arXiv:1806.08862, 2018.
26. Y. Umuroglu, D. Conficconi, L. Rasnayake, T. B. Preusser, and M. Sjalander, "Optimizing Bit-Serial Matrix Multiplication for Reconfigurable Computing," in *ACM Transactions on Reconfigurable Technology and Systems*, 2019.
27. NTNU, Xilinx Research Labs, *BISMO: A Scalable Bit-Serial Matrix Multiplication Overlay for Reconfigurable Computing*, [Internet]. Available: https://github.com/EECS-NTNU/bismo.

Further Reading

1. C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going Deeper with Convolutions," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015.
2. S. Lee, S. Purushwalkam, M. Cogswell, D. Crandall, and D. Batra, "Why M Heads Are Better than One: Training a Diverse Ensemble of Deep Networks," arXiv preprint arXiv:1511.06314, 2015.

Revision History

The following table shows the revision history for this document:

Date	Version	Description of Revisions
10/28/2019	1.0	Initial Xilinx release.

Disclaimer

Automotive Applications Disclaimer