WHITE PAPER

FPGA Inline Acceleration

Harnessing Numerical Flexibility for Deep Learning on FPGAs

Authors

Andrew C. Ling
andrew.ling@intel.com

Mohamed S. Abdelfattah
mohamed.abdelfattah@intel.com

Andrew Bitar
andrew.bitar@intel.com

David Han
david.han@intel.com

Roberto Dicecco
roberto.dicecco@intel.com

Suchit Subhaschandra
suchit.subhaschandra@intel.com

Chris N Johnson
chris.n.johnson@intel.com

Dmitry Denisenko
dmitry.denisenko@intel.com

Josh Fender
josh.fender@intel.com

Gordon R. Chiu
gordon.chiu@intel.com

Intel® Corporation
Programmable Solutions Group

Table of Contents

Abstract 1
Introduction 1
Deep Learning Accelerator 2
Compute Precision and Minifloat 2
Block Floating Point and Memory Impact 2
Block Floating Point vs Fixed Point 3
Conclusion 3
References 4

Abstract

Deep learning has become a key workload in the data center and the edge, leading to a race for dominance in this space. FPGAs have shown they can compete by combining deterministic low latency with high throughput and flexibility. In particular, FPGAs bit-level programmability can efficiently implement arbitrary precisions and numeric data types critical in the fast evolving field of deep learning.

In this paper, we explore FPGA minifloat implementations (floating-point representations with non-standard exponent and mantissa sizes), and show the use of a block-floating-point implementation that shares the exponent across many numbers, reducing the logic required to perform floating-point operations. The paper shows this technique can significantly improve FPGA performance with no impact to accuracy, reduce logic utilization by 3X, and memory bandwidth and capacity required by more than 40%.†

Introduction

Deep neural networks have proven to be a powerful means to solve some of the most difficult computer vision and natural language processing problems since their successful introduction to the ImageNet competition in 2012 [14]. This has led to an explosion of workloads based on deep neural networks in the data center and the edge [2].

One of the key challenges with deep neural networks is their inherent computational complexity, where many deep nets require billions of operations to perform a single inference. To mitigate the computational burden of deep nets three methods are often used:

1. Skipping redundant operations (e.g., multiply by 0) and modifying training algorithms or post-processing weights to lead to sparse connectivity in the network [9], [16].
2. Removing redundancy in the deep net by either trimming layers or connections [12-13].
3. Reducing the complexity of each operation by reducing their precision and bit width [4], [7-8], [10-11].

Because of their flexibility, FPGAs are perfect candidates to take advantage of all of these approaches. In this work, we will explore the third approach in detail.

Specifically, we show how you can efficiently map minifloat operations onto the FPGA fabric leading to a significant reduction in resource utilization with negligible degradation in accuracy on GoogLeNet, a common network used in image classification. Additionally, we will show how using a block-floating-point based approach can significantly increase the number of operations that can fit on a single FPGA, reducing the memory bandwidth and footprint required to store intermediate data and filter weights.

Finally, we show how our block floating-point implementation on Intel® Arria® 10 FPGAs has low overhead compared to fixed-point equivalent operations, and can be trivially converted to fixed-point operations if necessary.

Deep Learning Accelerator

We implemented a highly efficient deep learning inference engine [1], where a convolutional core, consisting of an array of processing elements, reads input image and filter data from external memory (DDR), and stores the data in caches built from on-chip block RAMs. The processing elements consist of highly efficient dot product kernels executing in parallel: one of the key operations in deep neural nets. We will explore how minifloat implementations can significantly reduce both logic utilization and memory bandwidth usage.

Compute Precision and Minifloat

Deep learning applications often have large memory and compute requirements, leading to exploration in reducing precision and complexity of each individual operation. Fixed-point representation has been employed by NVIDIA* [6], Xilinx* [8], and Google's TPU* [17] for convolutional neural network (CNN) and recurrent neural network (RNN) acceleration, while Microsoft* has recently announced the use of a reduced-precision floating-point on Intel® Stratix® 10 FPGAs in their acceleration of gated recurrent units (GRUs) [5].

We explore an approach similar to Microsoft's, taking advantage of minifloat representations that reduce the mantissa and exponent from IEEE 754 fp32. In our approach, we explore different mantissa sizes from 2 to 5 bits, which we refer to as fp8 to fp11 respectively. For all of our representations, we keep one bit for the sign value and five bits for the exponent.

Table 1 shows the relative impact of reducing the precision against fp32. Here we show that peak tera floating-point operations per second (TFLOPS) can increase up to 8X by moving to lower minifloat representations.†

	FP32	FP16	FP11	FP9
Relative TFLOPS	1.0X	2.0X	3.8X	8.0X

Table 1. Relative TFLOP Increase of Minifloat Precisions When Compared Against fp32 on the Intel® Stratix® 10 FPGA

The ability to take advantage of mixed precision networks makes FPGAs particularly attractive for deep networks that have different layers and operations with varying influence on the final accuracy of the results. Table 2 compares the accuracy of fp11 against fp32 for GoogLeNet using two approaches: changing the precision to fp11 for the dot product only in the Convolutional and InnerProduct layers and changing the entire design to fp11 for all operations in all layers. As Table 2 shows, changing all operations to fp11 has a significant degradation to overall accuracy, while changing only the dot product operations has little to no impact. The benefit is, since over 80% of the FPGA resources are dedicated to dot products, lowering only the dot products precision yields the majority of the logic reduction and most of the performance benefits.

	ALL	DOT PRODUCT ONLY
Top-1 Accuracy Drop	2.31%	0.04%
Top-5 Accuracy Drop	1.22%	0.20%

Table 2. Accuracy Impact of Reducing All Operations to fp11 vs. Dot Product Operations Only.

Block Floating Point and Memory Impact

Although Intel FPGAs support fp32 natively in hard DSP blocks, variable precision minifloating-point operations cannot be fully implemented in hard DSP bocks, and take a significant amount of resources to implement each multiply and add in soft logic [15]. However, in [3], we illustrated how we can implement the majority of dot products in block-floating-point form, where a group of operations are clustered to form a "block" of operations and the exponent is shared across all numbers in the block. An illustration of this is shown in Figure 1, for a conversion of fp11 (1-bit sign, 5-bit exponent, 5-bit mantissa) to block fp11 with a block size of four.

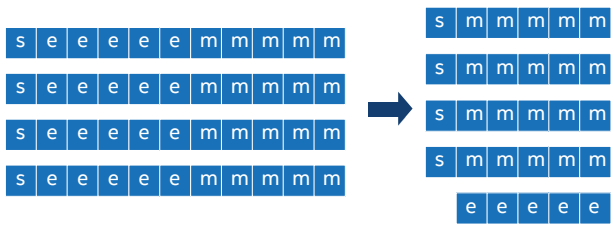

Figure 1. Illustration of Block-Floating-Point for fp11 with a Block Size of 4 (s=sign bits, e=exponent bits, m=mantissa bits).

Within each block, the mantissa is shifted such that all numbers in the block will have the same exponent and can be factored out. Any resulting multiplies or adds can then be applied directly on the resulting mantissas, which are equivalent to fixed-point operations in terms of cost on the FPGA. This can lead to over 3X reduction in required logic to implement when a block size of 8 is used, as described in [1]. In general, the larger the block size, the more resources can be saved; however, this leads to reduced accuracy, since as more numbers are shifted to align to a single exponent value, more bits may be shifted off in the mantissas found within the block. In practice, we find that a block size of 8 or 16 provide a good tradeoff between accuracy and resources [3]. However, in the event that accuracy is impacted, previous work has shown that top up training can successfully recover the accuracy loss incurred by large block sizes and lower precisions [4].

In addition to implementing dot products in block-floating-point form, we can store the data in block-floating-point form. This can lead to a significant reduction in both memory bandwidth to fetch data, and memory capacity to store the data either on or off chip. To illustrate this, Table 3 shows the compression ratio (lower is better) achieved by storing fp9, fp11, and fp16 in block-floating-point form, with a block size of 2 to 32 versus no blocking (i.e., block size of 1).

BLOCK SIZE	2	4	8	16	32
fp16	0.84	0.77	0.73	0.71	0.70
fp11	0.77	0.66	0.60	0.57	0.56
fp9	0.72	0.58	0.51	0.48	0.46

Table 3. Compression Ratio of Different Block Size Storage Requirements. Block size vs. No Blocking Used to Store Weights and Intermediate Data Feature Maps.

Block Floating Point vs Fixed Point

When implementing operations in block-floating-point, the majority of the operations are applied directly to the mantissas, which effectively converts floating-point operations into fixed-point operations leading to an implementation that is as efficient as fixed-point. For example, in fp11, the mantissa plus sign is 6 bits in width. This allows us to map two fp11 multiplies as two INT6 operations in the native 18x18 multiplier as illustrated in Figure 2.

In the event that fixed-point operation is desired, the block-floating-point dot products in our architecture can be trivially converted to fixed-point operations simply by removing the exponent and shifts required to convert to the block-floating-point format.

Once converted to naive fixed-point, scaling and quantization of weights would be required to account for the loss in dynamic range by moving to true fixed-point operations.

CONCLUSION

In this work we have demonstrated how using minifloat representations can have a significant impact to the over-all performance of the FPGA for deep learning inference applications. Using block-floating-point, we show how we can both reduce the logic utilization and memory footprint of the design. Additionally, we describe how block-floating-point efficiency is similar to fixed-point implementations with little overhead over fixed-point designs.

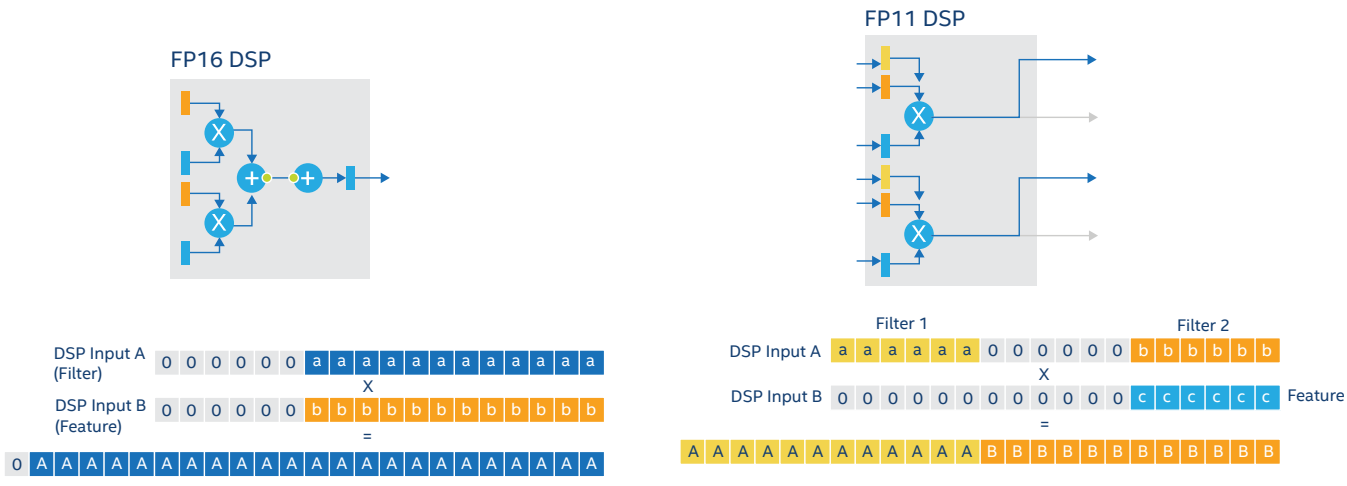

Figure 2. DSP Packing Technique of fp16 and fp11 Block-Floating-Point Multiplications into 18x18 Integer Multiplier DSP Block.

References

[1] Utku Aydonat, Shane O'Connell, Davor Capalija, Andrew C. Ling, and Gordon R. Chiu. An OpenCLTM Deep Learning Accelerator on Arria 10. In Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays. ACM, New York, NY, USA, 55–64.

[2] Diane Bryant. Keynote, Intel Developer Forum 2016, San Francisco.

[3] Gordon R. Chiu, Andrew C. Ling, Davor Capalija, Andrew Bitar, and Mohamed S. Abdelfattah. Flexibility: FPGAs and CAD in Deep Learning Acceleration. In Proceedings of the 2018 International Symposium on Physical Design. ACM, New York, NY, USA, 34–41. https://doi.org/10.1145/3177540.3177561

[4] Eric Chung, Jeremy Fowers, Kalin Ovtcharov, Michael Papamichael, Adrian Caulfield, Todd Massengill, Ming Liu, Mahdi Ghandi, Daniel Lo, Steve Reinhardt, Shlomi Alkalay, Hari Angepat, Derek Chiou, Alessandro Forin, Doug Burger, Lisa Woods, Gabriel Weisz, Michael Haselman, and Dan Zhang. 2018. Serving DNNs in Real Time at Datacenter Scale with Project Brainwave. https://www.microsoft.com/en-us/research/publication/serving-dnns-real-time-datacenter-scale-project-brainwave/

[5] Eric Chung et al. 2017. Accelerating Persistent Neural Networks at Datacenter Scale. Hot Chips.

[6] NVIDIA Corporation. 2017. NVIDIA TensorRT. [7] Philippe Coussy, Cyrille Chavet, Hugues Nono Wouafo, and Laura Conde-Canencia. 2015. Fully Binary Neural Network Model and Optimized Hardware Architectures for Associative Memories. J. Emerg. Technol. Comput. Syst. 11, 4, Article 35, 23 pages. https://doi.org/10.1145/2629510

[8] Yao Fu et al. 2016. Deep Learning with INT8 Optimization on Xilinx Devices. Xilinx white paper

[9] Chang Gao, Daniel Neil, Enea Coelini, Shih-Chii Liu, and Tobi Delbruck. 2018. DeltaRNN: A Power-efficient Recurrent Neural Network Accelerator. In Proceedings of t he 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays. ACM, New York, NY,USA, 21–30. https://doi.org/10.1145/3174243.3174261

[10] Philipp Gysel. 2016. Ristretto: Hardware- Oriented Approximation of Convolutional Neural Networks. CoRR abs/1605.06402 arXiv:1605.06402 http://arxiv.org/abs/1605.06402

[11] Philipp Gysel, Mohammad Motamedi, and Soheil Ghiasi. 2016. Hardware-oriented Approximation of Convolutional Neural Networks. CoRR abs/1604.03168. arXiv:1604.03168 http://arxiv.org/abs/1604.03168

[12] Song Han, Junlong Kang,Huizi Mao,Yiming Hu, Xin Li, Yubin Li, Dongliang Xie, Hong Luo, Song Yao, Yu Wang, Huazhong Yang, and William J, Dally. 2017. ESE: Efficicient Speech Recognition Engine with Sparse LSTM on FPGA. In Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA'17). ACM, New York, NY, USA, 75–84. https://doi.org/10.1145/3020078.3021745

[13] Forrest N. Iandola et al. 2016. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5 MB model size. arXiv:1602.07360.

[14] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2012. ImageNet Classification with Deep Convolutional Neural Networks. In Advances in Neural Information Processing Systems. 1097–1105.

[15] Amulya Vishwanath et al. 2016. Enabling High-Performance Floating-Point Designs. Intel white paper

[16] Shuo Wang, Zhe Li, Caiwen Ding, Bo Yuan, Qinru Qiu, Yanzhi Wang, and Yun Liang. C-LSTM: Enabling Efficient LSTM Using Structured Compression Techniques on FPGAs. In Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, ACM, New York, NY, USA, 11-20. https://doi.org/10.1145/3174243.3174253

[17] Yonghui Wu et al. 2016. Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation. arXiv:1609.08144

† Tests measure performance of components on a particular test, in specific systems. Differences in hardware, software, or configuration will affect actual performance. Consult other sources of information to evaluate performance as you consider your purchase. For more complete information about performance and benchmark results, visit www.intel.com/benchmarks.