Hybrid dot-product design for FP-enabled FPGAs

Bogdan Pasca
Programmable Solutions Group
Intel Corporation
France
bogdan.pasca@intel.com

I. Introduction

Machine learning (ML) is a high-profile application area for FPGAs. While initially FPGAs were only used as inference accelerators - novel research on using smaller precisions for training neural networks have allowed FPGAs to broaden their applicability to accelerating neural network training. Many training algorithms are composed of forward and back propagations through the network - with each step requiring many matrix-matrix multiplications. Current research focuses on the use the bfloat16 [1] format for the dot-product multiplications, while the reduction operation is implemented using single-precision (SP) arithmetic. The bfloat16 format is an IEEE-754-like 16-bit format having 8 exponent bits and 7 fraction bits. The interest of using this format on the dot-product entries, as opposed to using SP, is the 2x bandwidth usage - 16 vs 32-bits.

Modern FPGAs have hardened support for SP arithmetic [2], [3] in the DSP Blocks. Each DSP Block implements a multiply-add structure and inter-block routing allows for efficient implementation of dot-products - requiring minimal logic resources. The SP multipliers can act-down as bfloat16 multipliers which allows for a trivial dot-product mapping on the DSP-Block columns [4]. In the absence of bandwidth constraints, the bfloat16+SP dot-product compute density of a device will be directly dependent on total DSP Block count.

In this work we discuss one alternative for increasing the dot-product compute density by using a hybrid dot-product. The top-level diagram is shown in Figure 1. The hybrid dot-product combines DSP Blocks configured in FP mode (right) with a more traditional logic and fixed-point DSP-based dot-product (left). It increases the compute density of the device by generating a customized core that has a logic-to-DSP ratio close to that of the target device (or target deployment region).

II. Hybrid Dot Product

Modern FPGAs have varying ratios of DSPs and logic elements - that we denote for simplicity by $r_{\text{ALM to DSP}}$. Our goal is to generate a dot-product implementation that has a logic to DSP ratio (r_{dot}) less than or equal to $r_{\text{ALM to DSP}}$, and that provides a high system compute density (total number of multiplications).

The first step is to determine a splitting of n (the size of the dot-product) into α and β such that $n = \alpha + \beta$. The α-element dot product will be implemented using a more traditional approach, while the β-element dot product will be using the DSP Blocks in FP mode.

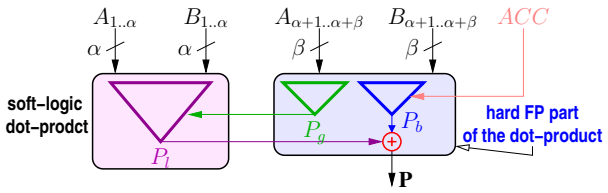

Fig. 1. Hybrid dot-product structure

Given a valid splitting, the total number of DSP Blocks used by the dot product is $C_{\text{DSP}} = \alpha/4 + \beta$. One DSP Block implements either 4-bfloat16 multiplications or 1 SP multiplication.

A user-visible knob w controls the accuracy of the dot-product by controlling the internal datapath size of the α-element **logic**-based dot-product P_l. We use a resource model for each component to estimate the number of logic elements of for an α-element dot-product.

A. Hard FP

The *Hard FP* part of the dot-product incorporates the addition of the SP input ACC, used when assembling the final result of wider dot-products. In order to accommodate for the addition without using an additional DSP Block, the Hard FP part of the dot-product is split into two sub dot-products: P_g (green in Figures 1 and 2) and P_b (blue); the ACC value is added to P_b. The dot-product result P_g merges with the soft-logic adder tree, to produce the result P_l. One carefully selected spare SP FP adder (in a used DSP) produces $P = P_l + P_b$. An example mapping of the hard FP part of the dot-product, for $\beta = 4$, $\beta_g = 2$, $\beta_b = 2$ is given in Figure 2 (color code matches Figure 1).

B. Soft FP

The α-element dot-product is fused - as opposed to being assembled out of IEEE-754 like operators. Two 8x8 mantissa multipliers (required for the bfloat16 multipliers) are mapped to a single 18x18-bit multiplier (half-DSP) and some logic [5]. The *normalization* stage is skipped, thus saving one multiplexer level. Rounding-to-nearest, which requires an addition that potentially updates the exponent, is replaced by rounding towards zero on a user-defined w-bit wide fraction. The local

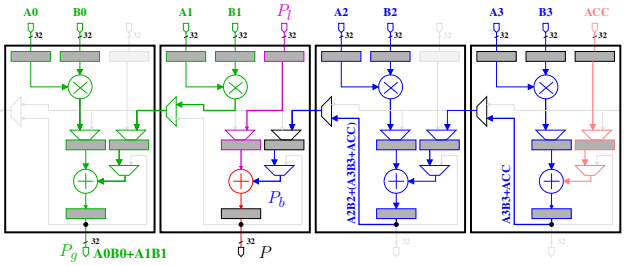

Fig. 2. Hard-FP part of the hybrid dot-product

TABLE I
RELATIVE ERROR COMPARISON BETWEEN THE PROPOSED HYBRID DOT-PRODUCT AND A TYPICAL AI BFLOAT16+SP IMPLEMENTATION FOR $n = 16$, $\alpha = 8$, $\beta = 4$, $\beta_g = 2$, $\beta_b = 2$

Config	Param	Proposed	AI
$e_c = 0$, $e_s = 5$	$w = 7$	1.287601e-02	4.570449e-03
	$w = 8$	6.172194e-03	
	$w = 9$	2.935275e-03	
$e_c = 0$, $e_s = 10$	$w = 7$	7.934867e-03	3.402314e-03
	$w = 8$	4.120781e-03	
	$w = 9$	1.864206e-03	
$e_c = 0$, $e_s = 20$	$w = 7$	6.672454e-03	2.996574e-03
	$w = 8$	3.161355e-03	
	$w = 9$	1.588372e-03	

overflow-underflow at the level of the multipliers is handled using an extended exponent. The behavior of the adders in the adder tree is similar, with the exception that the returned mantissa is in two's complement. Each adder level extends the input mantissa by 2 bits: one to capture a potential overflow, and another to compensate for rounding-towards-zero. The hard FP P_g merges in the adder tree. At the merge point the most accuracy is kept for P_g during the conversion to the internal adder format. The output of the soft FP adder tree passes through a normalization stage and keeps the maximum amount of accuracy before populating the SP operand, P_l.

C. Accuracy

The accuracy of the custom implementation of the hybrid dot product and be adjusted accordingly using the *w* parameter. We give in Table I an average accuracy comparison between the dot-product implementation using bfloat16 for multiplications and SP for the addition tree and our proposed hybrid implementation for different values of the *w* parameter. It can be observed that the parameter offers the flexibility required to match the average accuracy of the typical bfloat16+SP implementation.

D. Density

Stratix 10 devices are currently the largest produced by Intel [3] and make good candidates for neural network training accelerators. Within the Stratix 10 device family, there are multiple combinations of DSP blocks and logic: for instance Stratix 10-MX $r_{\text{ALM to DSP}} = 177$ and Stratix 10-GX has $r_{\text{ALM to DSP}} = 161$. The logic/DSP ratio for our architectures can be changed by modifying the splitting between $n = \alpha + \beta$, and by changing the value of the parameter w. Both changes also influence the accuracy of the solution: a large β may compensate for a smaller w. Table II shows the changes in the resource utilization ratio by changing both w and the ratio between α and β.

TABLE II
RESOURCE UTILIZATION RATIOS FOR TWO CONFIGURATIONS

Config	Param	ALMs	DSPs	r_{dot}
$n = 16$	$w = 7$	1030	7	147
$\alpha = 12, \beta = 4$	$w = 8$	1075		153
$\beta_g = 2, \beta_b = 2$	$w = 9$	1141		163
$n = 16$	$w = 7$	863	8.5	102
$\alpha = 10, \beta = 6$	$w = 8$	894		106
$\beta_g = 4, \beta_b = 2$	$w = 9$	948		112

III. CONCLUSION

The implementation of a seemingly simple dot-product operator exposes numerous potential tradeoffs for the user, from operator-specific to more general: (1) Particular reduction tree; was this operator inferred from an HLS design? Can the iterative accumulation nature be relaxed? (2) Accuracy; How is accuracy defined for your design? Do you care more about worst case or average accuracy? Relative or absolute? Can the implementation be validated with typical application data? Does it make sense to use system-level accuracy tests such as classification accuracy (for neural networks), or SNR (for filters)? (3) Ratio of resources; What is my exact deployment target, Stratix 10 or Arria 10? Is the core replicated thousands of times or only a few times per design? (4) Integration? Do all inputs need to be synchronized or can the top-level entity handle the synchronization? What are my available resources after the system plumbing is in place? (5) Design/Portability? What tools do we use to describe it (or parts or it)? How close to handwritten RTL (using low-level primitives) can HLS tools quality-of-results get? (6) Routability - how much of the local routing does my IP require?

The many tradeoffs make this type of core an excellent candidate for a arithmetic generator. Numerous switches allow modifying the ratio between α, β, and w depending on the exact deployment target, and accuracy criteria. Pipelining the design for a user-defined frequency can be automatized [6], [7], and even scheduling inputs at different cycles to further save area can be integrated. The challenging aspect that remains is choosing the level at which this generator operates: the higher the level the more high-level optimization opportunities (constant propagation, pruning etc.) and greater portability. The lower the level (essentially printing RTL) the more bitwidth-specific and FPGA-specific the design tends to get - lowering area at the expense of portability and design time.

REFERENCES

[1] Intel Corporation, "BFLOAT16 Hardware Numerics Definition," 11 2018. [Online]. Available: https://software.intel.com/sites/default/files/managed/40/8b/bf16-hardware-numerics-definition-white-paper.pdf

[2] *Intel Arria®10 Device Overview*, 2018, https://www.intel.com/content/dam/altera-www/global/en_US/pdfs/literature/hb/arria-10/a10_overview.pdf.

[3] *Intel Stratix®10 GX/SX Device Overview*, 2018, https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/hb/stratix-10/s10-overview.pdf.

[4] M. Langhammer and B. Pasca, "Floating-point DSP block architecture for FPGAs," in *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '15. New York, NY, USA: ACM, 2015, pp. 117–125. [Online]. Available: http://doi.acm.org/10.1145/2684746.2689071

[5] M. Langhammer, G. W. Baeckler, S. Gribok, D. N. Denisenko, and B. Pasca, "Methods for using a multiplier to support multiple sub-multiplication operations," Oct 2018. [Online]. Available: https://assignment.uspto.gov/patent/index.html#/patent/search/resultAssignment?searchInput=20190042198&id=47091-888

[6] A. J. Chung, K. Cobden, M. Jervis, M. Langhammer, and B. Pasca, "Tools and techniques for efficient high-level system design on FPGAs," *CoRR*, vol. abs/1408.4797, 2014. [Online]. Available: http://arxiv.org/abs/1408.4797

[7] F. de Dinechin and B. Pasca, "Designing custom arithmetic data paths with FloPoCo," *IEEE Design and Test*, 2011.