Algorithm-Hardware Co-Design of Adaptive Floating-Point Encodings for Resilient Deep Learning Inference

Thierry Tambe[1], En-Yu Yang[1], Zishen Wan[1], Yuntian Deng[1]
Vijay Janapa Reddi[1], Alexander Rush[2], David Brooks[1] and Gu-Yeon Wei[1]
[1]Harvard University, Cambridge, MA, [2]Cornell Tech, New York, NY

ABSTRACT

Conventional hardware-friendly quantization methods, such as fixed-point or integer, tend to perform poorly at very low precision as their shrunken dynamic ranges cannot adequately capture the wide data distributions commonly seen in sequence transduction models. We present an algorithm-hardware co-design centered around a novel floating-point inspired number format, *AdaptivFloat*, that dynamically maximizes and optimally clips its available dynamic range, at a layer granularity, in order to create faithful encodings of neural network parameters. AdaptivFloat consistently produces higher inference accuracies compared to block floating-point, uniform, IEEE-like float or posit encodings at low bit precision ($\leq$ 8-bit) across a diverse set of state-of-the-art neural networks, exhibiting narrow to wide weight distribution. Notably, at 4-bit weight precision, only a 2.1 degradation in BLEU score is observed on the AdaptivFloat-quantized Transformer network compared to total accuracy loss when encoded in the above-mentioned prominent datatypes. Furthermore, experimental results on a deep neural network (DNN) processing element (PE), exploiting AdaptivFloat logic in its computational datapath, demonstrate per-operation energy and area that is 0.9× and 1.14×, respectively, that of an equivalent bit width NVDLA-like integer-based PE.

1 INTRODUCTION

In order to exact higher DNN compute density and energy efficiency at all computing scales, a plethora of reduced precision quantization techniques have been proposed. In this line of research, a large body of work has focused on fixed-point encodings [3, 5, 20] or uniform quantization via integer [21, 27]. These fixed-point techniques are frequently evaluated on shallow models or on CNNs exhibiting relatively narrow weight distributions. However, as shown in Figure 1, sequence transduction NLP models [11] with layer normalization such as the Transformer [28] contain weights more than an order of magnitude larger than those from popular CNN models with batch normalization such as ResNet-50. The reason for this phenomenon is that batch normalization effectively produces a weight normalization side effect [25] whereas layer normalization adopts invariance properties that do not reparameterize the network [2].

In the pursuit of wider dynamic range and improved numerical accuracy, there has been surging interest in floating-point based [6, 18, 26], logarithmic [12, 19] and posit number systems [8], which also form the inspiration of this work.

AdaptivFloat improves on the aforementioned techniques by dynamically maximizing its available dynamic range at a neural network layer granularity. And unlike block floating-point (BFP) approaches with shared exponents that may lead to degraded rendering of smaller magnitude weights, AdaptivFloat achieves higher inference accuracy by remaining committed to the standard floating-point delineation of independent exponent and mantissa bits for each tensor element. However, we break from IEEE 754 standard compliance with a unique clamping strategy for denormal numbers and with a customized proposition for zero assignment, which enables us to engineer leaner hardware.

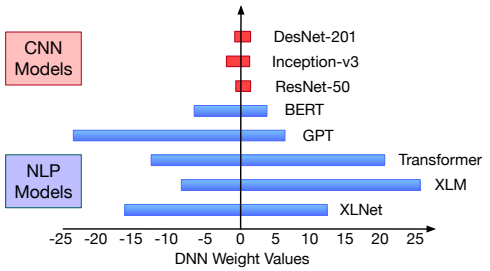

Figure 1: Range of weights from popular CNN and NLP models. Weights in NLP models [11] can be more than 10× larger than the maximum absolute weight value of common CNNs.

Rather than proposing binary or ternary quantization techniques evaluated on a group of carefully selected models, through AdaptivFloat, we aim to inform a generalized floating-point based mathematical blueprint for adaptive and resilient DNN quantization that can be easily applied on neural models of various categories (CNN, RNN or MLP), layer depths and parameter statistics.

By virtue of an algorithm-hardware co-design, we also propose a processing element implementation that exploits AdaptivFloat in its computational datapath in order to yield energy efficiencies that surpass those of integer-based variants. Furthermore, owing to the superior performance of AdaptivFloat at very low word sizes, as it will be shown, higher compute density can be acquired at a relatively lower penalty for computational accuracy compared to block floating-point, integer, posit and non-adaptive IEEE-like float encodings. Altogether, the AdaptivFloat algorithm-hardware co-design framework offers accuracy and energy advantages over integer or fixed-point solutions.

Finally, we note that the AdaptivFloat encoding is self-supervised as it only relies on unlabeled data distributions in the network.

This paper, therefore, makes the following contributions:

- We propose and describe a novel floating-point based data encoding algorithm for deep learning, *AdaptivFloat*, which maximizes its dynamic range at a neural network layer granularity by dynamically shifting its exponent range and by optimally clipping its representable datapoints.
- We evaluate AdaptivFloat across a diverse set of DNN models and tasks and show that it achieves higher classification and prediction accuracies compared to equivalent bit width uniform, block floating-point and non-adaptive posit and float quantization techniques.

978-1-7281-1085-1/20/$31.00 ©2020 IEEE

Authorized licensed use limited to: STAATS U UNIBIBL BREMEN. Downloaded on October 09,2021 at 16:06:08 UTC from IEEE Xplore. Restrictions apply.

- We propose a hybrid float-integer (HFINT) PE implementation that exploits the AdaptivFloat mechanism and provides a cost-effective compromise between the high accuracy of floating-point computations and the greater hardware density of fixed-point post-processing. We show that the HFINT PE produces higher energy efficiencies compared to NVDLA-like monolithic integer-based PEs.

2 RELATED WORK

Adaptive Precision Techniques. [24] noticed that large magnitude weights bear a higher impact on model performance and proposed outlier-aware quantization, which requires separate low and high bitwidth precision hardware datapaths for small and outlier weight values, respectively. [26] observed that the cell state variation in LSTM networks can be used to switch between high and low precision during inference, yielding greater energy efficiency as a result. [22] performs adaptive fixed-point quantization in search for a minimum energy network. Stripes [14] performs bit-serial computing to enable configurable per-layer adaptive precision. And several quantization techniques such as [16] aim to find the suitable per-layer bitwidth. In this work, rather than physically changing the bitwidth to adapt to the network precision requirement, we seek to obtain the best numerical representation out of a chosen bitwidth regimen by dynamically adapting, per-layer, the floating-point exponent range.

Floating-point Inspired Data Types. Aware of the dynamic range limitation of fixed-point encoding, we have seen block floating-point data types, such as Flexpoint [18] and similar variants employed in the Brainwave NPU [7]. Block floating-point's appeal stems from its potential to achieve floating-point-like dynamic range with hardware cost and implementation comparable to fixed-point. However, by collapsing the exponent value of each tensor element to that of the element with the highest magnitude, elements with smaller magnitudes will be more prone to data loss. Floating-point quantization by dividing a symmetrically-thresholded DNN tensor by a scale factor has been reported [26]. Our quantization process formulates a floating-point exponent bias from the maximum absolute value in the DNN tensor.

Finally, we note that *Deep Compression* techniques [9] such as pruning and weight sharing can be used in combination to this work, which evaluates instead against mutually-exclusive numerical data types prominently used in deep learning hardware acceleration.

3 METHODOLOGY

This section articulates the AdaptivFloat format and describes the inner workings behind the adjustment of the available dynamic range of representable values in order to best fit the weights of neural network layers.

3.1 The AdaptivFloat Format

The AdaptivFloat number representation scheme generally follows the IEEE 754 Standard floating-point format that includes a sign bit, exponent bit, and mantissa bit fields. In order to efficiently encode in hardware all the representable datapoints, we avoid the computation of denormal values in the AdaptivFloat number system.

The main problem arising as a result of not using denormals in floating-point is the lack of representation for the "zero" point,

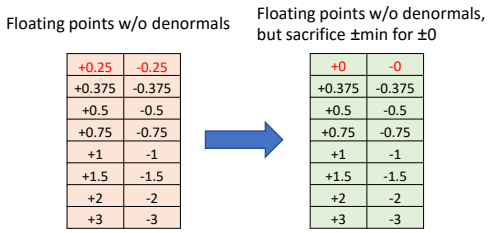

Figure 2: Illustration of the 0 value representation in AdaptivFloat

which is essential to DNN computations. We solve this constraint by sacrificing the positive and negative minimum values to allocate the slot for "zero" as shown in Figure 2. If both exponent bits and mantissa bits are zeros, the bit vector should be interpreted as zero. Otherwise, the bit vector is converted by the following equation:

$$sign * 2^{\,exponent\ value} * mantissa\ value \tag{1}$$

where the *exponent value* is the addition of exponent bits and exp_{bias}, and the *mantissa value* is calculated by appending an implied "one" as the MSB and follow the same format as standard floating point.

Moreover, at very low bit compression, customization based on the value range of a neural network layer can greatly reduce the quantization error with little overhead on the shared extra parameters. Thus, similar to integer quantization that uses a quantization scale (or step), we formulate a bias value, exp_{bias}, to dynamically shift the range of exponent values at a layer granularity. The calculation of exp_{bias} is described in Section 3.2. A benefit of using exp_{bias} is the simplicity of the hardware logic required to perform the adaptive operation compared to the multiplying quantization scale used in integer quantization. This contrast is discussed in detail in Section 5.

3.2 Quantization

Having defined the AdaptivFloat data format, the quantization problem is simply mapping a full precision value to the nearest representable datapoint. For this purpose, we need to determine the optimal AdaptivFloat exp_{bias} which will maximize the available dynamic range of the encoding format in order to provide the most accurate rendering of a particular matrix (or NN layer). This is analogous to determining the quantization scale for integer quantizations, but exp_{bias} is a small, typically negative, integer value rather than the high-precision floating-point scaling factor needed in integer quantization [21].

Algorithm 1 describes how to find the most suitable exp_{bias} in order to faithfully encode the AdaptivFloat-quantized weight matrix. We first compute the sign matrix, W_{sign}, and the matrix of absolute values, W_{abs}, from the full precision weight matrix W_{fp}. Then, the algorithm finds the maximum absolute value from W_{abs} to determine the exp_{bias} corresponding to a suitable range of representable datapoints for the weight matrix to quantize. Before doing quantization on W_{abs}, we first round the values smaller than the AdaptivFloat minimum representable value to zero or $value_{min}$ at a halfway threshold. Then, we clamp values larger than the max value, $value_{max}$, to $value_{max}$. The quantization involves rewriting W_{abs} into normalized exponent and mantissa form with an exponent matrix W_{exp} and a mantissa matrix W_{mant}. The mantissa matrix is quantized by the quantization scale calculated by

Authorized licensed use limited to: STAATS U UNIBIBL BREMEN. Downloaded on October 09,2021 at 16:06:08 UTC from IEEE Xplore. Restrictions apply.

Algorithm 1: AdaptivFloat Quantization

Input: Matrix W_{fp}, bitwidth n and number of exponent bits e

// Get Mantissa bits

$m := n - e - 1$

// Obtain sign and abs matrices

$W_{sign} := sign(W_{fp})$; $W_{abs} := abs(W_{fp})$

// Determine exp_{bias} and range

Find normalized exp_{max} for $max(W_{abs})$ such that

$2^{exp_{max}} \leq max(W_{abs}) < 2^{exp_{max}+1}$

$exp_{bias} := exp_{max} - (2^e - 1)$

$value_{min} := 2^{exp_{bias}} * (1 + 2^{-m})$

$value_{max} := 2^{exp_{max}} * (2 - 2^{-m})$

// Handle unrepresentable values

Round $value < value_{min}$ in W_{abs} to 0 or $value_{min}$

Clamp $value > value_{max}$ in W_{abs} to $value_{max}$

// Quantize W_{fp}

Find normalized W_{exp} and W_{mant} such that

$W_{abs} = 2^{W_{exp}} * W_{mant}$, and $1 \leq W_{mant} < 2$

$W_q :=$ quantize and round W_{mant} by $scale = 2^{-m}$

// Reconstruct output matrix

$W_{adptiv} := W_{sign} * 2^{W_{exp}} * W_q$

return W_{adptiv}

the number of mantissa bits. W_q indicates the quantized mantissa matrix. Finally, the AdaptivFloat-quantized matrix is reconstructed by multiplication of W_{sign}, $2^{W_{exp}}$, and W_q.

We use the notation $AdaptivFloat < n, e >$ to indicate a n-bit AdaptivFloat number with e exponent bits. Figure 3 provides an illustration showing how exp_{bias} is chosen to best fit the range of values in a weight matrix and the resulting quantized datapoints adhering to the $AdaptivFloat < 4, 2 >$ format. Generally, the narrower the datapoints are, which indicates a smaller absolute maximum value in the weight tensor, the more negative exp_{bias} gets.

4 EXPERIMENTAL RESULTS

For bit compression evaluation, we select three popular DNN models of distinct neural types and applications, and exhibiting relatively narrow to wide spread in their weight distributions. The models considered, as shown in Table 1, are: (1) Transformer [28], which made a very consequential impact in the field of machine translation and question answering; (2) a 4-layer LSTM encoder, 1-layer LSTM decoder, attention-based sequence-to-sequence (seq2seq) network [4] commonly used in speech recognition; and (3) ResNet-50, a well-known image classification CNN [10]. The Transformer and the LSTM-based Seq2Seq networks are trained on the OpenNMT platform [17] using the WMT'17 English-to-German and LibriSpeech datasets, respectively. And, ResNet-50 is trained on the Pytorch framework using the ImageNet dataset.

We compare the efficacy of AdaptivFloat along with numerical data types frequently employed for deep learning acceleration, namely block floating-point (BFP), IEEE-like float, posit, and uniform representations. We created templates of these data types in Python to be run within the PyTorch framework. The AdaptivFloat, uniform, and block floating-point quantization schemes are self-adaptive in the sense that their dynamic range auto-adjusts,

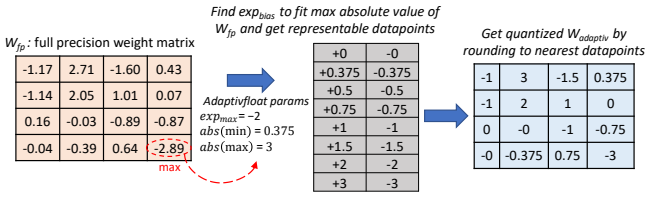

Figure 3: Illustration of $AdaptivFloat < 4, 2 >$ quantization from a full precision weight matrix

layer-by-layer, based on the distribution of the tensor. The number of exponent bits in the AdaptivFloat, IEEE-like float, and posit formats is set evenly for all the layers in the network to the value yielding the highest inference accuracy after doing a search on the exponent width. Generally, the best inference performance was obtained with the exponent space set to 3 bits for AdaptivFloat, 4 bits for float (3 bits when the word size becomes 4 bits), and 1 bit for posit (0 bit when the word size becomes 4 bits).

Finally, we note that the following results are generated by quantizing all of the layers in the DNN models in order to capture the whole-length performance of these five numerical data types unlike several works [3] that intentionally skip quantization on the sensitive first and last layers to escape steeper accuracy loss.

4.1 Root Mean Squared Error

We begin by quantifying the quantization error of the number formats with respect to baseline FP32 precision. The boxplots depicted in Figure 4 show the distribution of the root mean squared (RMS) quantization error emanating from the data types and computed across all layers of the three models under evaluation. AdaptivFloat consistently produces lower average quantization error compared to uniform, BFP, posit, or IEEE-like float encoding. Furthermore, among the self-adaptive data types, AdaptivFloat exhibits the tightest error spread for all bit widths of the Transformer and seq2seq networks, while BFP's error spread is thinnest for the 6-bit and 8-bit versions of the ResNet-50 model – although with a higher mean compared to AdaptivFloat. This suggests that BFP would fare best in networks with slimmer weight distribution such as ResNet-50. Among the non-adaptive data types, we see that posit generally yields both a lower average RMS quantization error and a narrower interquartile error range compared to Float. These results provide important insights to quantized DNN performance as we dive into the bare inference accuracy results in the next subsection.

4.2 Inference Performance

Tables 2 shows the resiliency of the data types under study as they are put to the test under varying weight bit compression on the Transformer, sequence-to-sequence, and ResNet-50 models. The inference results are tabulated post-training quantization (PTQ) and after quantization-aware re-training (QAR) from the plateaued FP32 baseline. The training setup and the hyper-parameter recipe are kept the same for all five data types under evaluation in order to impose a fair comparison.

The key observation we can distill is that AdaptivFloat demonstrates much greater resiliency at very low precision ($\leq$ 6-bit) compared to the other four data formats. Notably, at 4-bit encoding, AdaptivFloat can still yield, after retraining, a decent BLEU score of 25.5 on the Transformer model while the impact from the other

Authorized licensed use limited to: STAATS U UNIBIBL BREMEN. Downloaded on October 09,2021 at 16:06:08 UTC from IEEE Xplore. Restrictions apply.

Table 1: DNN models under evaluation

Model	Application	Dataset	Structure	Number of parameters	Range of weights	FP32 performance
Transformer	Machine translation	WMT'17 En-to-De	Attention, FC layers	93M	[-12.46 , 20.41]	BLEU: 27.40
Seq2Seq	Speech-to-text	LibriSpeech 960h	Attention, LSTM, FC layers	20M	[-2.21 , 2.39]	WER: 13.34
ResNet-50	Image classification	ImageNet	CNN, FC layers	25M	[-0.78, 1.32]	Top-1 Acc: 76.2

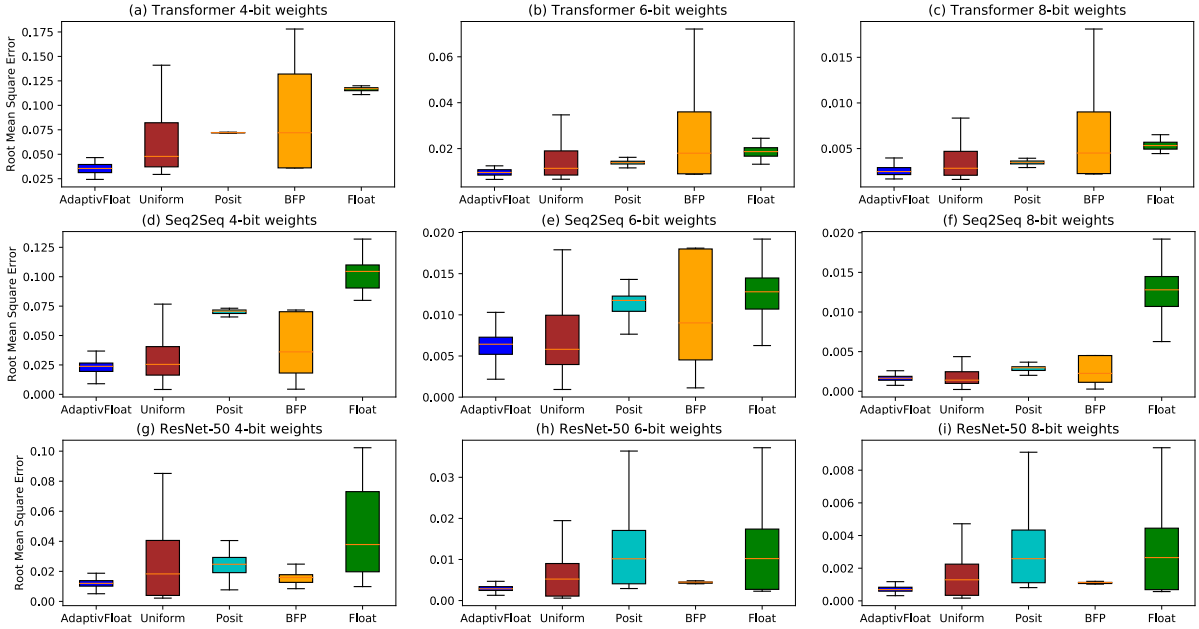

Figure 4: Root mean square of the quantization error w.r.t. FP32 at 4-bit, 6-bit and 8-bit weight precision across the layers of the Transformer, Seq2Seq and ResNet-50 models. AdaptivFloat produces the lowest mean error compared to the other number systems.

Table 2: Impact of weight bit compression post-training quantization / post-quantization aware retraining

	BLEU Score of Transformer (BLEU @ FP32=27.4)					Word Error Rate of Seq2Seq (WER @ FP32=13.34)					Top-1 Accuracy of ResNet-50 (Top-1 Acc. @ FP32=76.2)				
#Bits	Float	BFP	Uniform	Posit	AdaptivFloat	Float	BFP	Uniform	Posit	AdaptivFloat	Float	BFP	Uniform	Posit	AdaptivFloat
16	27.4 / 27.4	27.4 / 27.4	27.4 / 27.4	27.4 / 27.5	27.4 / 27.6	13.40 / 13.07	13.30 / 13.14	13.27 / 12.82	13.29 / 13.05	13.27 / 12.93	76.1 / 76.3	76.2 / 76.3	76.1 / 76.3	76.1 / 76.3	76.2 / 76.3
8	27.2 / 27.5	26.3 / 27.3	27.3 / 27.4	27.3 / 27.5	27.3 / 27.7	14.06 / 12.74	13.23 / 13.01	13.28 / 12.89	13.24 / 12.88	13.11 / 12.59	75.4 / 75.9	75.7 / 76.0	75.9 / 76.1	75.4 / 76.0	75.7 / 76.3
7	27.1 / 27.5	16.9 / 26.8	26.0 / 27.2	27.3 / 27.4	27.3 / 27.7	13.95 / 12.84	13.54 / 13.27	13.45 / 13.37	13.36 / 12.74	13.19 / 12.80	73.8 / 75.6	74.6 / 75.9	75.3 / 75.9	74.1 / 75.8	75.6 / 76.1
6	26.5 / 27.1	0.16 / 8.4	0.9 / 23.5	26.7 / 27.2	27.2 / 27.6	15.53 / 13.48	14.72 / 14.74	14.05 / 13.90	15.13 / 13.88	13.19 / 12.93	65.7 / 74.8	66.9 / 74.9	72.9 / 75.2	68.8 / 75.0	73.9 / 75.9
5	24.2 / 25.6	0.0 / 0.0	0.0 / 0.0	25.8 / 26.6	26.4 / 27.3	20.86 / 19.63	21.28 / 21.18	16.53 / 16.25	19.65 / 19.13	15.027 / 12.78	16.1 / 73.6	13.2 / 73.4	15.1 / 74.0	33.0 / 73.9	67.2 / 75.6
4	0.0 / 0.0	0.0 / 0.0	0.0 / 0.0	0.0 / 0.0	16.3 / 25.5	inf / inf	76.05 / 75.65	44.55 / 45.99	inf / inf	19.82 / 15.84	0.5 / 66.3	0.5 / 66.1	2.6 / 67.4	0.7 / 66.7	29.0 / 75.1

four number formats is catastrophic due to insufficient dynamic range or decimal accuracy. We can make similar observations on the seq2seq and ResNet-50 models as AdaptivFloat show modest retrained performance degradation at 4-bit and 5-bit weight precision. For instance, only a 1.2 Top-1 accuracy drop is seen with a weight width of 4-bit. When the weights of the seq2seq model are quantized to 4-bit, the non-adaptive data types (Float and Posit) are essentially unable to provide expressible transcription. This suggests that, for resilient performance at very low word size, it is critical to have a quantization scheme that can adjust its available dynamic range to encode the compressed weights as faithfully as possible. AdaptivFloat's marked robustness at very low precision enables higher compute density into reconfigurable architectures at a relatively lower penalty for computational accuracy.

Adding noise to weights when computing the parameter gradients has been shown to produce a regularization effect that can improve generalization performance [23]. This effect can be seen in all data types and is particularly pronounced in AdaptivFloat with performance exceeding FP32 by up to +0.3 in BLEU score, -0.75 in word error rate and +0.1 in Top-1 accuracy.

4.3 Effect of both Weight and Activation Quantization

Tables 3 reports the inference performance from reducing the word size on both weights and activations. Wn/An signifies a quantization of n-bit weight and n-bit activation.

We observe that AdaptivFloat's 8-bit performance is as good as, if not better than, the baseline FP32 result on all three DNN models while the degradation at 6-bit is still modest. Interestingly, in the case of the seq2seq model, the 6-bit AdaptivFloat weight and activation quantization generates regularization effective enough to exceed the FP32 baseline. At 4-bit weight and activation precision, the performance degradation of AdaptivFloat is steeper on the sequence models than on ResNet-50 as many of the activations from the attention mechanism fall outside of the available dynamic range of the number format.

5 PE ARCHITECTURE

AdaptivFloat's superior bit compression ability paves the way to efficient bit packing into resource-constrained accelerators. In this section, we describe the design of a hybrid Float-Integer (HFINT) PE that exploits the AdaptivFloat logic in its computational datapath

Authorized licensed use limited to: STAATS U UNIBIBL BREMEN. Downloaded on October 09,2021 at 16:06:08 UTC from IEEE Xplore. Restrictions apply.

Table 3: Impact of both weight and activation quantization, measured after quantization-aware retraining

	BLEU Score of Transformer (BLEU @ FP32=27.4)					Word Error Rate of Seq2Seq (WER @ FP32=13.34)					Top-1 Accuracy of ResNet-50 (Top-1 Acc. @ FP32=76.2)				
#Bits	Float	BFP	Uniform	Posit	AdaptivFloat	Float	BFP	Uniform	Posit	AdaptivFloat	Float	BFP	Uniform	Posit	AdaptivFloat
W8/A8	27.4	27.4	10.1	26.9	27.5	12.77	12.86	12.86	12.96	12.59	75.7	75.7	75.9	75.8	76.0
W6/A6	25.9	0.0	5.7	25.7	27.1	14.58	14.68	14.04	14.50	12.79	73.5	73.4	74.1	73.6	75.0
W4/A4	0.0	0.0	0.0	0.0	0.3	inf	78.68	48.86	inf	21.94	63.3	63.0	64.3	63.0	72.4

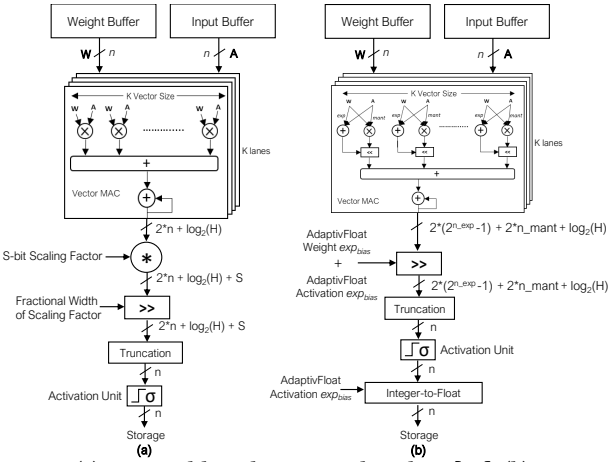

Figure 5: (a) NVDLA-like n-bit Integer-based PE [29], (b) Proposed n-bit Hybrid Float-Integer PE.

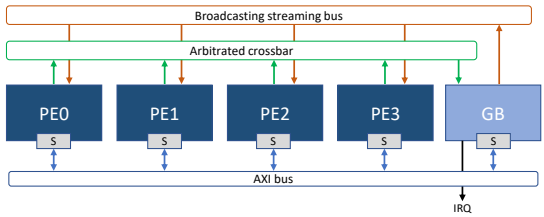

Figure 6: Accelerator system with 4 PEs and a global buffer (GB) targeting sequence-to-sequence networks.

and provides an efficient compromise between the high accuracy of floating-point computations and the greater hardware density of fixed-point post-processing. We contrast the proposed PE architecture against that of a NVDLA-like PE [29] implementing monolithic integer arithmetic, which is also observed in the plurality of commercial ML accelerators (TPU [13], ARM ML NPU [1], etc.).

5.1 NVDLA-like Integer PE

The micro-architecture of a NVDLA-like n-bit integer-based PE is shown in Figure 5a. It contains fixed-point vector MAC units receiving n-bit integer weight and activation vectors. The MAC partial sums are stored in $2 * n + log_2(H)$-bit registers in order to accumulate up to H values without overflow. A high-precision scaling factor is typically used to dequantize the computation with high accuracy [21]. Using a S-bit scaling factor requires the scaled results to be stored in registers of width $2{*}n{+}log_2(H){+}S$, which later are bit-shifted right by the fractional value of the scaling. Then, the data is clipped and truncated back to n bits before being modulated by the neural network activation function. As an example, an 8-bit integer-based PE architecture will be referred later in the document as INT8/24/40 to designate a datapath with 8-bit MAC operands, accumulated into 24-bit (to add up to 256 values without overflow) and then scaled to 40-bit using a 16-bit scaling factor.

5.2 Hybrid Float-Integer PE

Figure 5b illustrates the micro-architecture of our proposed n-bit Hybrid Float-Integer (HFINT) PE. The vector MACs units perform floating-point multiplications between a n-bit float weight vector and a n-bit float activation vector — and accumulate the result as integer. The weights and activations stored on-chip are quantized according to the AdaptivFloat algorithm described in Algorithm 1 in Section 3. The extracted AdaptivFloat exp_{bias} for weight and activation tensors are saved in allocated 4-bit registers and are used to shift the exponent range of the accumulated partial sums. We note that while the AdaptivFloat exp_{bias} for the static weights are extracted post-training, the exp_{bias} for the dynamic activations are informed from statistics during offline batch inference on the test dataset. The accumulation precision needs to be $2 * (2^{n_{exp}} - 1) + 2 * n_{mant} + log_2(H)$ -bit in order to accumulate up to H values without overflow. The accumulated partial sums are then clipped and truncated back to n-bit integer before being processed by the activation function. At the end of the PE datapath, the integer activations are converted back to the AdaptivFloat format. An 8-bit HFINT PE architecture will be referred to as HFINT8/30 to indicate an 8-bit MAC datapath with 30-bit accumulation.

A key contrast to note between the INT PE and the HFINT PE, apart from the differing data types employed in the vector MAC units, is that the INT PE requires a post-accumulation multiplier in order to perform the adaptive operation of the quantization. This in turn increases the required post-accumulation precision by S-bit before truncation. In the next section, we provide energy, performance, and area comparisons between the two PE topologies.

6 HARDWARE EVALUATION

6.1 Experimental Setup

In order to evaluate the hardware on a realistic DNN workload, we designed an accelerator system, depicted in Figure 6, targeted for RNN and FC sequence-to-sequence networks where we have seen wider parameter distributions compared to convolution networks. The accelerator is evaluated with four PEs that are integrated as either INT or HFINT. A global buffer (GB) unit with 1MB of storage collects the computed activations from the individual PEs via the arbitrated crossbar channel and then broadcasts them back to the four PEs, in order to process the next time step.

Each PE contains an input/bias buffer with sizes ranging from 1KB to 4KB and a weight buffer whose size ranges from 256KB to 1MB depending on the vector size and operand bit width. We also note here that the HFINT PE always uses MAC operands with 3 exponent bits which was found to yield the best inference accuracy across the ResNet-50, Seq2Seq, and Transformer networks.

The INT and HFINT accelerators were both designed in SystemC with synthesizable components from the MatchLib [15] library. Verilog RTL was autogenerated by the Catapult high-level synthesis (HLS) tool with HLS constraints uniformly set with the goal to achieve maximum throughput on the pipelined designs.

Authorized licensed use limited to: STAATS U UNIBIBL BREMEN. Downloaded on October 09,2021 at 16:06:08 UTC from IEEE Xplore. Restrictions apply.

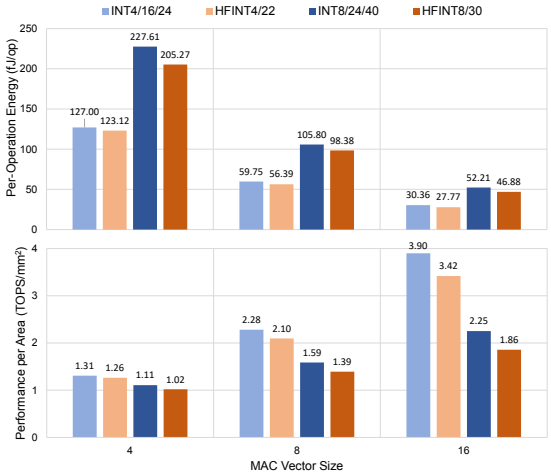

Figure 7: Per-operation energy (Top) and throughput per unit area (Bottom) of the INT and HFINT PEs across MAC vector sizes.

Table 4: PPA results of the 8-bit INT and 8-bit HFINT accelerators

	Power (mW)	Area (mm^2)	Computational Time for 100 LSTM Timesteps (μs)
Int Accelerator with 4 INT8/24/40 PEs	61.38	6.9	81.2
HFInt Accelerator with 4 HFINT8/30 PEs	56.22	7.9	81.2

For fair comparison, the two designs employ the same evaluation methodology. Energy, performance and area results are reported on the post-HLS Verilog netlists by the Catapult tool at 1GHz clock frequency using a commercial 16nm FinFET standard cell library. The simulated workload consists of 100 LSTM time steps with 256 hidden units operating in a weight stationary dataflow.

6.2 Energy, Performance and Area Analyses

We evaluate the effect of increasing throughput via the MAC vector size, K, which also equals the number of parallel MAC lanes, meaning that a single PE throughput equals $K^2 10^9 OPS$.

Figure 7 (Top) shows that larger vector sizes and operand bit widths benefits more the HFINT PE than the INT PE in terms of energy efficiency. Precisely, from 4-bit operands and vector size of 4 – to 8-bit operands and vector size of 16, the per-operation energy of the HFINT PE is 0.97× to 0.90× that of the INT PE. The smaller per-operation energy (i.e. higher energy efficiency) of the HFINT PE stems from the fact that its vector MACs contain smaller mantissa multipliers and exponent adders that consume less overall power than the full bitwidth multipliers used in the INT PE's MACs.

Increasing the vector size is found to improve overall energy efficiency due to higher spatial reuse of the accumulated partial sums. On the other hand, the INT PEs exhibit 1.04× to 1.21× higher performance per unit area compared to the HFINT PEs due to its more compact and homogeneous logic in the vector MACs.

Table 4 reports the power, area, and compute time of the 8-bit INT and 8-bit HFINT accelerator systems with 4 PEs and a global buffer. The PEs here have a MAC vector size of 16 in both systems. While both accelerators achieve the same compute time (due to same aggregate pipelining generated by Catapult), the HFINT accelerator reports 0.92× the power and 1.14× the area of the integer-based adaptation, confirming the efficiency trends reported in Figure 7.

7 CONCLUSION

This paper presents an algorithm-hardware co-design centered around AdaptivFloat, a floating-point based encoding solution that dynamically maximizes and optimally clips its available dynamic range, at a layer granularity, in order to create robust encodings of neural network parameters from narrow to wide weight distribution spread. The proposed HFINT PE, leveraging AdaptivFloat, produces higher energy efficiencies than integer-based adaptations at varying vector sizes and MAC operand bit widths. Altogether, the AdaptivFloat algorithm-hardware co-design framework offers a tangible accuracy and energy advantage over fixed-point solutions.

8 ACKNOWLEDGMENTS

This work was supported by the Application Driving Architectures (ADA) Research Center, a JUMP Center cosponsored by SRC and DARPA

REFERENCES

[1] Arm. Arm machine learning processor. https://developer.arm.com/ip-products/processors/machine-learning/arm-ml-processor.
[2] L. J. Ba et al. Layer normalization. *ArXiv*, abs/1607.06450, 2016.
[3] J. Choi et al. Accurate and efficient 2-bit quantized neural networks. In *SysML*, 2019.
[4] J. Chorowski et al. Attention-based models for speech recognition. In *NIPS*, 2015.
[5] M. Courbariaux et al. Binaryconnect: Training deep neural networks with binary weights during propagations. In *NIPS*, 2015.
[6] M. Drumond et al. End-to-end DNN training with block floating point arithmetic. *arXiv*, abs/1804.01526, 2018.
[7] J. Fowers et al. A configurable cloud-scale dnn processor for real-time ai. In *ISCA*, 2018.
[8] J. Gustafson et al. Beating floating point at its own game: Posit arithmetic. *Supercomputing Frontiers and Innovations*, 2017.
[9] S. Han et al. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. *ICLR*, 2016.
[10] K. He et al. Deep residual learning for image recognition. In *CVPR*, 2016.
[11] Hugging Face. Transformers. https://github.com/huggingface/transformers.
[12] J. Johnson. Rethinking floating point for deep learning. *arXiv*, abs/1811.01721, 2018.
[13] N. P. Jouppi et al. In-datacenter performance analysis of a tensor processing unit. In *ISCA*, 2017.
[14] P. Judd et al. Stripes: Bit-serial deep neural network computing. In *MICRO*, 2016.
[15] B. Khailany et al. A modular digital vlsi flow for high-productivity soc design. In *DAC*, 2018.
[16] S. Khoram et al. Adaptive quantization of neural networks. In *ICLR*, 2018.
[17] G. Klein et al. OpenNMT: Open-source toolkit for neural machine translation. In *Proceedings of ACL*, 2017.
[18] U. Köster et al. Flexpoint: An adaptive numerical format for efficient training of deep neural networks. In *NIPS*, 2017.
[19] E. H. Lee et al. Lognet: Energy-efficient neural networks using logarithmic computation. In *ICASSP*, 2017.
[20] D. D. Lin et al. Fixed point quantization of deep convolutional networks. In *ICML*, 2016.
[21] S. Migacz. 8-bit inference with tensorrt. In *NVIDIA GPU Tech Conf*, 2017.
[22] B. Moons et al. Minimum energy quantized neural networks. In *ACSSC*, 2017.
[23] H. Noh et al. Regularizing deep neural networks by noise: Its interpretation and optimization. In *NIPS*, 2017.
[24] E. Park et al. Energy-efficient neural network accelerator based on outlier-aware low-precision computation. In *ISCA*, 2018.
[25] T. Salimans et al. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In *NIPS*, 2016.
[26] S. Settle et al. Quantizing convolutional neural networks for low-power high-throughput inference engines. *ArXiv*, abs/1805.07941, 2018.
[27] F. Silfa et al. Boosting lstm performance through dynamic precision selection. *ArXiv*, abs/1911.04244, 2019.
[28] A. Vaswani et al. Attention is all you need. In *NIPS*, 2017.
[29] B. Venkatesan et al. Magnet : A modular accelerator generator for neural networks. In *ICCAD*, 2019.

Authorized licensed use limited to: STAATS U UNIBIBL BREMEN. Downloaded on October 09,2021 at 16:06:08 UTC from IEEE Xplore. Restrictions apply.