Hardware-Software Codesign of DNN Accelerators using Approximate Posit Multipliers

Tom Glint*, Kailash Prasad*, Jinay Dagli, Krishil Gandhi#, Aryan Gupta, Vrajesh Patel, Neel Shah, Joycee Mekie
IIT Gandhinagar, India | #SVNIT, India
{tom.issac,kailash.prasad,joycee}@iitgn.ac.in

Abstract

Emerging data intensive AI/ML workloads encounter memory and power wall when run on general-purpose compute cores. This has led to the development of a myriad of techniques to deal with such workloads, among which DNN accelerator architectures have found a prominent place. In this work, we propose a hardware-software co-design approach to achieve system-level benefits. We propose a quantized *data-aware* POSIT number representation that leads to a highly optimized DNN accelerator. We demonstrate this work on SOTA SIMBA architecture, extendable to any other accelerator. Our proposal reduces the buffer/storage requirements within the architecture and reduces the data transfer cost between the main memory and the DNN accelerator. We have investigated the impact of using integer, IEEE floating point, and posit multipliers for LeNet, ResNet and VGG NNs trained and tested on MNIST, CIFAR10 and ImageNet datasets, respectively. Our system-level analysis shows that the proposed approximate-fixed-posit multiplier when implemented on SIMBA architecture, achieves on average ∼2.2× speed up, consumes ∼3.1× less energy and requires ∼3.2× less area, respectively, against the baseline SOTA architecture, *without* loss of accuracy (∼ ±1%)

CCS Concepts

• **Hardware → Emerging architectures**; • **Computer systems organization → Neural networks**; **Data flow architectures**; • **Computing methodologies → Neural networks**.

Keywords

DNN accelerators, neural networks, co-design

ACM Reference Format:

Tom Glint*, Kailash Prasad*, Jinay Dagli, Krishil Gandhi#, Aryan Gupta, Vrajesh Patel, Neel Shah, Joycee Mekie. 2023. Hardware-Software Codesign of DNN Accelerators using Approximate Posit Multipliers. In *28th Asia and South Pacific Design Automation Conference (ASPDAC '23), January 16–19, 2023, Tokyo, Japan.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3566097.3567866

*Both authors contributed equally to this research.

ASPDAC '23, January 16–19, 2023, Tokyo, Japan

ACM ISBN 978-1-4503-9783-4/23/01…$15.00
https://doi.org/10.1145/3566097.3567866

1 Introduction

Artificial Intelligence (AI) and Machine Learning (ML) are fast penetrating every aspect of human life and have been used to provide near accurate solutions for various predictive and classification tasks such as image classification, object detection, image segmentation, text-to-speech, and language processing [9]. In view of this, hardware solutions that can accelerate these AI-ML workloads have emerged as promising solutions to deal with the power[4] and memory wall [15] inherent to the von-Neumann based general purpose computing in handling massive data manipulations. DNN hardware accelerators (HA) are optimally designed for multiply-and-accumulate (MAC) operation, which is the core compute operation in DNNs. The main advantage is gained by operating thousands of MAC units in parallel to reduce the latency of carrying out the task. Specifically, with technology scaling, there is potential to add a large number of MAC units on a large die area. However, two main issues practically limit these DNN accelerators' real-world applications: (a) Limited memory bandwidth [2, 16] that restricts data to be fetched from main memory into the DNN accelerators, rendering many MAC units idle as not enough data can be cached from the main memory into the accelerator buffer. (b) A stringent energy budget is allocated to the hardware accelerator, which is often due to the application type. This energy budget needs to include the energy required for data movement within the HA, between the main memory and HA, and for MAC and other compute operations. Special emphasis is placed on reusing data cached into the DNN accelerator to reduce main memory access. This is done by optimally mapping the neural network architecture layers [11].

In this work, we present a unique data quantization approach in error-resilient pre-trained ML workloads to deal with the energy and bandwidth constraints in DNN hardware accelerators without incurring any loss in accuracy. We use the SIMBA architecture [16] as the baseline architecture, but our technique itself can be applied to any DNN accelerator. Data quantization in the DNN accelerator has three-fold benefits, which impact energy and bandwidth. We propose using a quantized fixed posit number system as an alternative to conventional floating point number representation [5, 13]. This enables us to reduce the number of bits (quantization) required to represent numbers which, in turn, (a) reduces the MAC size and its energy and latency, (b) reduces the storage requirement within the hardware accelerator, and thus both storage and transfer energy reduces, and (c) allow us to fetch more operands with the given bandwidth, thus reducing the energy of data transfer between main memory and the hardware accelerator and increasing the throughput (or reducing the latency). Our quantization technique is unique in two aspects compared to the conventional techniques [2]. Firstly, we have optimized the bit requirement by using POSIT number

representation and using an optimally designed multiplier for the same [5, 13]. Secondly, from our analysis of the filters and inputs used in different ML workloads (LeNet, VGG, ResNet), we observe that the regime bits remain constant for most of them. Thus, we do not need to store these values, which significantly improves the quantization. Thirdly, and most importantly, unlike most of the existing works, we show the impact of this quantization on the full system of DNN hardware accelerator and main memory interfaced with it after optimally mapping the different machine learning workloads on this system. While we obtain large benefits in computation energy and latency, we also see significant system-level benefits of quantization without loss of accuracy, which is the most important contribution of this work.

The main contributions of this work are as follows: (a) The most significant contribution of this work is the system-level study of the impact of quantization in DNN accelerators. We obtain ~2.2× speedup and ~3.1× reduction in energy compared to the baseline SIMBA architecture. Our proposal reduces the chip area of the baseline architecture by ~3.2×. (b) We show the usefulness of alternative fixed POSIT number representation to improve quantization. Further, our data-aware approach allows 25% more quantization by approximating the regime bits in posit number representation. (c) We have implemented exact and approximate (quantized) multipliers using fixed posit and floating point representations to obtain the energy and latency values used in the system-level analysis. These are applied to pre-trained image recognition CNN workloads such as VGG [12], Lenet [8] and ResNet [6] and ensure that quantization has minimal accuracy drop during inferencing.

2 Background

DNN workloads consisting of convolution layers (CL) and fully connected layers (FCL) can be processed using DNN accelerators [1, 2, 16]. They have specialized compute elements and spatial arrangements for efficient and fast processing of DNNs. The accelerators can also be further optimized with design techniques, such as compression, quantization and pruning, that exploit the robustness of specific neural networks to errors. Further, different data encoding schemes can be used in DNNs to represent data using less number of bits.

2.1 Simba architecture

Simba [16] is a state-of-the-art data agnostic DNN accelerator. As shown in Fig. 1, each Simba chip has a global buffer, 16 Processing Elements (PE) arranged in mesh Topology and a RISC V processor, and is connected to an external memory. The external memory stores the inputs, outputs and weights used in CLs and FCLs. The RISC-V processor co-ordinates the flow and execution of data to each PE. Each PE has separate buffers for input, weight and partial sums. As shown in Fig. 1, eight input words are commonly shared across eight Vector MAC units. Each vector MAC unit has eight multipliers whose products are summed up into a single word using an adder tree and stored at the accumulation buffer after adding to the previous partial sum. Each Vector MAC units has a dedicated weight buffer. Inputs and weights of each layer are fetched from the external memory and buffered at the various buffer levels for optimum data reuse and the final outputs of each layer are written back to the memory.

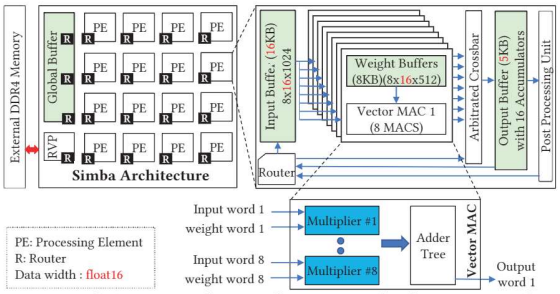

Figure 1: Left Top: Simba Architecture [16] connected to an external memory. The architecture consists of 16 Processing Elements (PE) that are connected with each other and the global buffer over a Network-on-Chip (NoC). Right Top: shows the internal components of a PE. Bottom: Vector MAC

2.2 Hardware-software co-design

DNN accelerators can process agnostic of the data to keep the accelerator design simple, or data-dependent techniques can be applied to make the storage and compute requirements less but at the expense of more complicated hardware that supports these techniques. The significant ones are compression, pruning quantization, and approximation among these data-dependent techniques [2]. In compression, similar data and zero data are compressed to reduce data volume, and multiplication with zero is skipped [2]. In pruning, the resiliency of DNN algorithms is exploited, and random or otherwise informed computations and the associated data are skipped when processing the DNN workload [2]. In quantization, the precision of the data is altered, giving both storage and computational benefits [2]. In approximation, the compute units are modified to provide approximate results but at a fraction of the compute cost and latency [2].

2.3 Multipliers for accelerating DNN workloads

Due to the low-precision requirements of DNNs efficient implementations of small multipliers have recently gained growing interest. Today, virtually all computer systems use IEEE-754 floating point representation to represent real numbers. For applications requiring lower precision, FP 16 and FP 8 are proposed. The bfloat16 (Brain Floating Point) is another floating-point format occupying 16 bits in computer memory. Using a floating radix point, it represents a wide dynamic range of numeric values. This format is a truncated (16-bit) version of the 32-bit IEEE 754 single-precision floating-point format with the intent of accelerating DNNs.

Despite the popularity of IEEE 754 representation, it has many shortcomings. It breaks the mathematical laws of commutativity and associativity and suffers from underflow/overflow while performing arithmetic operations. Further, since it has a fixed number of precision bits, it suffers from rounding errors when representing real numbers. These complexities lead to complex hardware design.

Gustafson and Yonemoto [5, 13] proposed a new format called the Posit number system to overcome the shortcomings of the IEEE-754 format. The posit number representation consists of a sign, one or multiple regime bits, multiple optional exponent bits, and multiple optional fraction bits. The sign bit is 1 for negative numbers and 0 for positive number. The number of regime bits is dynamic that follows a unique encoding which is a series of all 0's or all 1's that ends with an opposite bit. Similarly, the number

of bits for the fraction and exponent is dynamic. A posit number generally includes the exponent and fraction only if required.

One of the drawbacks of the Posit number system is that it consumes more area and power. To reduce the power and area requirements, the fixed-posit representation was introduced [13]. In the fixed-posit format, [13], in addition to the value of N and es, the length of regime is also fixed.

3 Proposed DNN accelerator

At the system level in DNN accelerators, we have three critical aspects for low energy and latency. One is the compute unit (blue box in Fig. 1) within the HA, the Second is the storage within the HA (green box in Fig. 1), and the third is memory access within the HA and with the DRAM chip (blue and red arrows in Fig. 1). In this proposal, we propose a method to quantize the compute unit. Our initial experiment suggests that the multiplier is the main bottleneck for energy and performance. A detailed analysis is presented in Section 5. We analyze existing multipliers, namely, IEEE 754, bFloat, Posits, and Fixed posit at the system level and quantize them until there is less than ±1 % accuracy drop. Because of that, the multiplication operation takes lesser energy and becomes faster. Second, because of the lesser bit, the storage space requirement reduces, resulting in energy and area reduction. The third is the memory access within the accelerator and from the main memory reduced because of the use of lesser bits. Each of the components providing system level benefits is highlighted for Simba Architecture in Fig 1. Further, the system-level analysis of all the multipliers suggests that the Fixed Posit multiplier gives the maximum benefit in terms of energy, area, and latency.

Beyond this, we analyze computation data of the DNNs for fixed posit representation. For all the DNNs under consideration in this work, we observe that the regime bits remain the same for 50% - 95% of the cases. Based on this understanding, we propose an approximate fixed posit multiplier where we do not store the regime bit. Thus we reduce the number of bits further and then analyze the system-level benefit of the proposed multiplier on the Simba accelerator.

Table 1: Specification of Simba (baseline)

Property	Value (baseline)
External memory used	DDR4 DRAM
Baseline Architecture	Simba
Bandwidth to DRAM from chip	25GBps
Access energy to DRAM from logic die	46 pJ/bit
Global buffer size	64 KB
No. of PEs	16
MAC Units per PE	64
PE weight buffer size	65 KB
PE input buffer size	16 KB
PE output buffer size	5 KB
MAC latency	979 ps
Word size	Float16

4 Experimental setup

The experimental framework used for a fair and detailed comparison has four major aspects: (i) The baseline hardware (Simba) and the proposed hardware are modeled, and its hardware-bound aspects are captured using a framework made up of Timeloop [11], Accelergy [14], and Cacti [10]. The parameters for multipliers and adders are obtained from Cadence Genus Synthesis at 65 nm technology node (ii) Both older NN, like Lenet and VGG, and newer deep NN like ResNet are considered as the workload for analysis. (iii) The optimal workload mapping is found for each hardware, rather than a static mapping policy, for a fair comparison of accelerators. The configuration of the Simba (baseline) used in this work is given in Table 1. (iv) Inference accuracy analysis

4.1 Hardware model and optimal mapping

4.1.1 *Timeloop* is used for modeling the DNN accelerator. It captures the spatial organization of a DNN accelerator, including the hierarchy of different storage classes like DRAM, SRAM buffers, registers, and the placement of MAC units in relation to the memory hierarchy. Further, it captures the bandwidth available for communication between each hardware unit. *Timeloop Mapper* identifies all possible permutations of mapping a workload to hardware and calculates the latency and energy associated with each mapping based on the Timeloop and Accelergy model. Further, it optimizes for a parameter (least latency followed by least energy) and outputs the optimal mapping based on the optimization criteria.

4.1.2 *Accelergy* is in charge of finding the area and energy associated with sub-components of the DNN accelerator. The energy is calculated on a per-action basis.

4.1.3 *Cacti* is used for finding the area and energy of SRAMs and MAC units. The area, energy and latency values of the multiplier and adder in the MAC unit are obtained from Cadence Genus synthesis for each design. This work uses the 65 nm node for evaluation across the system.

4.2 Workloads

This work primarily focuses on Convolutional Neural Network-based DNNs and their inference. We use them to represent scenarios where NN can not be re-trained due to privacy and data access limitations [7]. We use LeNet and VGG to establish a baseline as prior works were optimized for these older networks. However, we use ResNet, as it has variations that are 151 layers deep and is part of the widely adopted MLPerf inference benchmark suite [9]. VGG is a DNN with substantially more layers and has 13 convolutional layers, and the largest and most common filter size is 3×3. For ResNet, we consider only the unique layers, and results from these individual layers can be extrapolated to any ResNet combination.

4.3 Accuracy analysis

To obtain the baseline accuracy LeNet, VGG and ResNets are implemented in the Pytorch framework with 32-bit floating point data words. We use MNIST (LeNet), CIFAR-10 (ResNet) and ImageNet (VGG) data sets for training and calculating accuracy. For calculating the inference accuracy of various multipliers, the Pytorch framework is modified to redirect all MAC operations internal to the DNN to the corresponding software model of the multiplier. The multiplier is modeled at an event level and is precise to the bit level.

4.4 Choice of baseline design

Simba, the state-of-the-art data agnostic accelerator, is used as a baseline to identify and quantify the system level benefit of applying hardware-software co-design techniques such as quantization and approximation. Thus area, energy, and latency benefit of using a particular multiplier, quantization, and number system can be obtained. We note that further benefits might be obtained, on top of the observed results, by using compression, pruning, and other dictionary-based hardware-software co-design approaches, which are beyond the scope of this work.

5 Methodology

This section discusses the detailed methodology for the proposed DNN accelerator. A DNN accelerator should process the workload with the least latency and energy. To analyze the same, we have studied the critical timing path and energy consumption of each component.

5.1 Critical timing path in accelerator

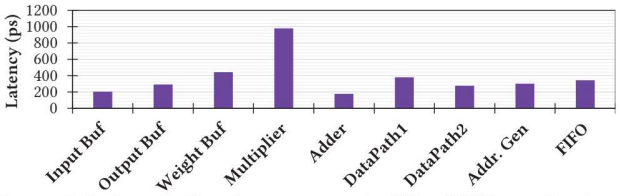

Figure 2: Latency of each component of the DNN accelerator

To determine the critical timing path in the DNN accelerator, each component of the DNN accelerator is synthesized separately using Cadence Genus. Fig. 2 shows the zero slack latency of each component in the baseline accelerator. Here, the multiplier has the highest latency, and any decrease in latency of the multiplier will directly result in a reduction of clock cycle time for the entire accelerator. Hence, multiplier latency needs to be brought down.

5.2 Energy expenditure at each component

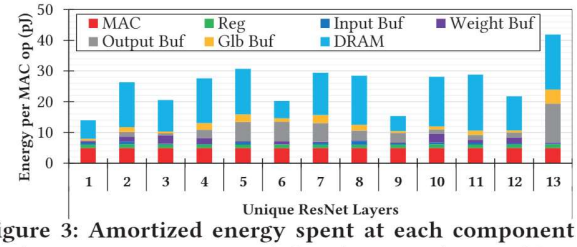

Figure 3: Amortized energy spent at each component of Simba per MAC operation while inferencing layers of ResNet

Fig. 3 shows the amortized energy spent per MAC operation at each component of the Simba accelerator when inferencing each layer of ResNet. It can be observed that external memory access dominate the total energy per MAC operation at ~49% followed by buffer operations at ~31%. The volume of data access to the external memory and the buffers must be decreased to reduce the overall system energy, along with any optimization to the MAC unit where quantization is a potential method.

5.3 Alternative multipliers

Table 2 shows the baseline multiplier's (FP16) area, energy, and latency along with alternative multipliers that can be used to implement the MAC unit. The values are obtained from Cadence Genus synthesis at 65 nm with the least cycle time possible. The table also shows each multiplier's corresponding accuracy when inferencing using LeNet and ResNet18. It is observed that fixed posit multipliers can maintain higher accuracy for newer DNNs at lower bit-width compared to IEEE 754 implementation (see FP8 and FPOS8). The metrics related to the proposed multiplier (AFPOS), given in Section 6, is also added at the end of Table. 2 for conciseness.

5.4 Data distribution in posit encoding

When fixed posit number systems (N, es, regime) are used for encoding data of DNN, it is observed (as shown in Fig. 4) that the regime bits values are not uniformly distributed, and for the majority of the operations, the resultant value of k (as in Section 6) is −1. This insight may be used to implement an approximate encoding scheme that omits the regime bits of the data, thereby further quantizing the data that needs to be stored.

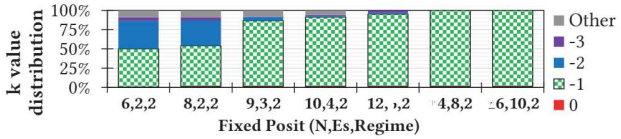

Figure 4: Observed distribution of value of k (Section 2.3) for fixed posit encoding when inferencing LeNet

6 Proposed multiplier

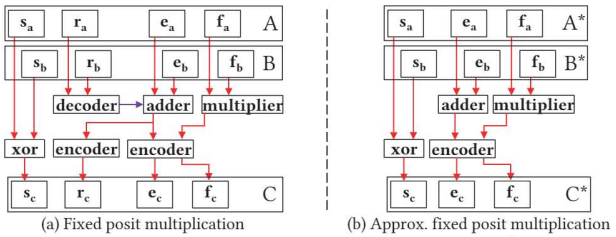

Figure 5: Posit and Proposed approximate posit multiplier

The posit number system is represented in the form of (N,es), where N represents the total number of bits and es represents the maximum exponent size. A posit number is made up of four major parts: the sign bit(0 for a positive number, while 1 for a negative number), the regime bits (a series of all 0's or all 1's that ends with an opposite bit), the exponent bits, and the mantissa. Another parameter, k is also being used, the value of which is determined by the regime bits. If the regime consists of $m-1$ zeros, k equals $-m$, and if the regime consists of $m-1$ ones, k equals $m-1$. An important point to note is that there is a unique representation of regime bits for any value of the parameter k. Eq. 1 shows the encoded resultant value and relationship with bit representation.

$$value = (-1)^{sign} \times 2^{2^{es}k} \times 2^{exp} \times (1+f) \quad (1)$$

One of the drawbacks of the Posit number system is that it consumes more area and power. To reduce the power and area requirements, the fixed-posit representation was introduced [13]. In the fixed-posit format, [13], in addition to the value of N and es, the length of regime is also fixed. That is, the fixed-posit format is represented by (N,es,regime), where N is the total number of bits, es represents the maximum exponent size, and regime represents the length of regime bits. The circuit diagram of the already proposed fixed-posit multiplier is as shown in Fig. 5(a).

The sign bit of the final result, s_c, is obtained by XOR-ing the sign bits of the two operands, s_a and s_b. The final mantissa bits, f_c, are obtained after multiplying the two mantissa bits, f_a and f_b. The decoder shown in the figure extracts the value of parameter k. The final exponent, e_c, is obtained by adding the two exponent bits, e_a and e_b, the shifted value of k, and the carry from mantissa multiplication. The value of regime, r_c, is obtained using an encoder, as shown in the figure. Another encoder is used to take care of underflow and overflow conditions in the case of fixed-posit.

Based on the observations from the previous section, we propose an approximate fixed posit multiplier for use in DNN accelerators, as shown in Fig. 5 (b). The multiplier takes two operands whose

Table 2: Alternative multipliers that can be implemented at the MAC unit of DNN accelerators

Multiplier	Abbrv.	Type	Bit Width	Bit Arrangment	Latency (ps)	Energy (pJ)	Area (um^2)	LeNet Acc.	ResNet Acc.
IEEE 754 Multiplier(32,8)	FP32	Floating Point	32	(N,es)	1299	22.5	13830.84	98.49%	82.21%
IEEE 754 Multiplier(16,5)	FP16	Floating Point	16	(N,es)	979	4.55	3095.28	98.49%	82.02%
IEEE 754 Multiplier(10,5)	FP10	Floating Point	10	(N,es)	579	1.17	964.08	98.43%	63.81%
IEEE 754 Multiplier(8,5)	FP8	Floating Point	8	(N,es)	334	0.217	362.88	93.28%	23.54%
Bfloat16 Multiplier(16,8)	BF16	Floating Point	16	(N,es)	832	2.75	2148.12	98.52%	79.19%
Array Multiplier	AM16	Integer	16	-	1780	8.81	6015.24	98.55%	53.18%
Posit Multiplier(16,3)	POS16	Floating Point	16	(N,es)	1984	10.8	7660.44	98.49%	82.16%
Posit Multiplier(10,2)	POS10	Floating Point	10	(N,es)	1183	3.39	2202.12	98.53%	81.6%
Posit Multiplier(8,2)	POS8	Floating Point	8	(N,es)	685	0.903	790.56	98.45%	74.56%
Posit Multiplier(6,2)	POS6	Floating Point	6	(N,es)	483	0.397	582.12	97.38%	13.36%
Fixed Posit Multiplier(16,4,2)	FPOS16	Approximate	16	(N,es,regime)	1234	3.98	3859.56	98.49%	82.16%
Fixed Posit Multiplier(10,4,2)	FPOS10	Approximate	10	(N,es,regime)	709	0.985	1243.44	98.62%	80.73%
Fixed Posit Multiplier(8,2,2)	FPOS8	Approximate	8	(N,es,regime)	663	0.963	997.92	98.62%	80.73%
Fixed Posit Multiplier(6,2,2)	FPOS6	Approximate	6	(N,es,regime)	581	0.536	477	98.14%	60.11%
Approximate Fixed Posit Multiplier(10,4)	AFPOS10	Approximate	8	(N,es)	602	0.607	959.04	98.62%	80.73%
Approximate Fixed Posit Multiplier(8,2)	AFPOS8	Approximate	6	(N,es)	563	0.463	776.88	98.62%	80.73%
Approximate Fixed Posit Multiplier(6,2)	AFPOS6	Approximate	4	(N,es)	321	0.0794	285.12	98.14%	60.11%

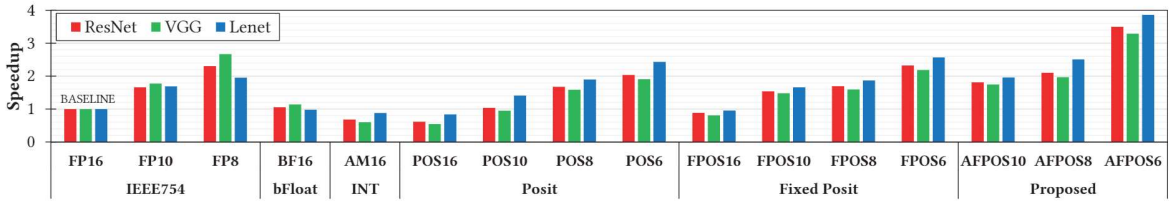

Figure 6: Relative speedup of DNN accelerators with various multipliers compared to baseline FP16

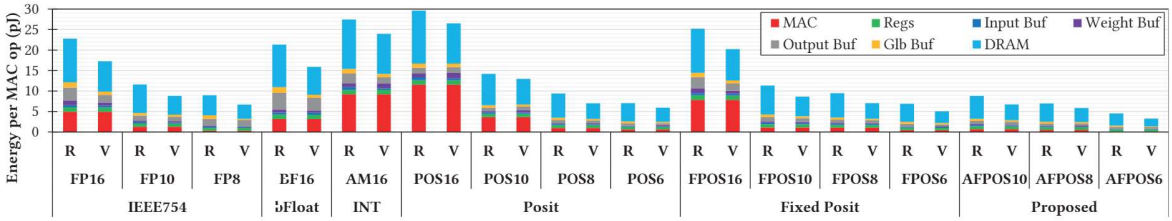

Figure 7: Split of average energy per MAC operation for ResNet (R) and VGG (V)

regime bit are excluded, but the multiplier internally uses −1 as value of k for calculation. Further, the encoding state of the product is also optimized as the regime bits are omitted. This results in a more simplified design that might have lower area, energy, and latency.

7 Results and analysis

7.1 Approximate fixed posit multiplier

The proposed approximate fixed posit multiplier (AFPOS) given in Section 6 is synthesized using Cadence Genus using 65 nm node. The area, energy and latency obtained for the AFPOS8 (see Table 2 design is 776.88 um^2, 0.463 pJ and 563 ps. Hence, the proposed multiplier has 22.15% less area, 51.92% less energy and 15.08% less latency compared to the fixed posit multiplier (FPOS8). We note that the number of bits required to store that data is also reduced by 25% for AFPOS8 encoding scheme compared to FPOS8. Further, the proposed multiplier has 3.9× less area, 9.8× less energy and 1.7× less latency compared to the quantized 16 bit floating point multiplier (baseline). At the same time, the inference accuracy remains similar to the FP32 (71.36% vs 72.28% for VGG on the ImageNet dataset and 80.73% vs 82.21% for ResNet on the CIFAR10 dataset) and has no loss of accuracy compared to baseline FP16. Futher, we note that quantized versions of IEEE float number (FP10 and FP8) achieve much less accuracy compared to baseline. We derive the system-level benefits of using various multipliers in next sub-sections.

7.2 System latency

As seen in Section 5.1, the latency of the multiplier will directly affect the cycle time of the DNN accelerator. AFPOS8 with a lower cycle time can have a higher compute throughput for a given time, provided the external memory can supply enough data for computation. Further, due to the omission of the regime bits, the effective bandwidth from the external memory also increases compared to the baseline. Thus the system level speedup needs to be identified. Fig. 6 shows the speedup of DNN accelerators with various multipliers compared to the baseline when inferencing ResNet, VGG, and LeNet. The system with AFPOS8 obtains the highest speedup of 2.2× compared to the baseline system without loss of accuracy for ResNet, VGG and LeNet, on average. We like to point out that the increase in frequency was only 1.7× AFPOS8 and the difference in speedup is due to increase in effective external bandwidth. In other multipliers, FP8 based system achieves higher speedup than FPOS8, but at *significant* loss of accuracy.

7.3 System energy

Quantization reduces the number of bits that are required to be read and written into memory and buffers and, therefore, will directly impact the energy for inference. Fig. 7 shows the energy spend at each component per MAC operation while processing ResNet and VGG (newer NNs). The system with AFPOS8 obtains ~3.1× less energy compared to the baseline system. This reduction

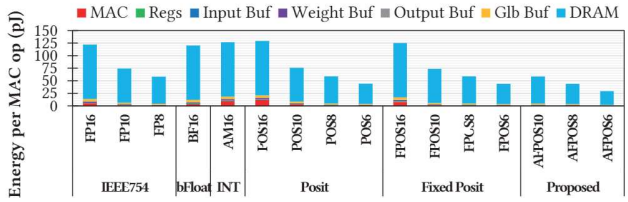

Figure 8: Amortized energy spent at each component of Simba while inferencing LeNet (old workload)

in newer DNNs can be mainly attributed to reduction in MAC energy and memory access energy. On the otherhand, Fig. 8 shows the component wise energy per MAC operation for LeNet (older NN). In this NN, both AFPOS8 and AFPOS6 have near baseline accuracy and have ~2.8× and ~4.2× less energy than baseline FP16. In LeNet, the major reduction in energy is due to the reduced reads and writes to the memory as bits are approximated. In Figures 7 and 8, the system level energy benefits for AFPOS8 are not equivalent to the multiplier level benefits as it is not the major energy drawing component in a DNN accelerator (as seen in Fig. 3).

7.4 System area

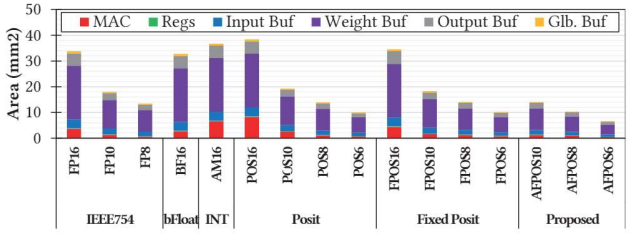

Figure 9: Area of Simba Chip with various encoding schemes

The area of the chip directly relates to the cost of manufacturing the chip, and a lower area reduces the chances of defects appearing in a chip sample. Fig. 9 shows the area of each Simba chip using the corresponding multiplier and encoding scheme. In these designs, the number of words that can be stored at each buffer remains the same, but the total number of bits stored at each buffer varies due to the encoding scheme. It can be observed that the implementation of the DNN accelerator with AFPOS8 multiplier has the lowest area of 10.36 mm^2 without loss of accuracy on all DNNs. Thus AFPOS8 reduces chip area by ~3.2× compared to baseline. For LeNet (less number of layers), AFPOS6 can maintain baseline accuracy and therefore a dedicated systems for this NN with AFPOS6 will reduce area by ~5.1×.

8 Related work

Deng et al. [2] have surveyed all recent works on hardware-software co-design of DNN accelerators. Still, most of these methods require retraining the DNN model to achieve accuracy, the same as the baseline results. Joydeep et al. [3] have mathematically shown the necessary number of bits for exponent and mantissa in a floating point number system for representing trained DNNs without loss of accuracy when inferencing (requires no retraining). Sumit et al. [13] have shown the distribution data in trained DNNs and have shown the multiplier level benefits of using a fixed posit number system. However, [3] and [13] do not identify and quantify the system-level benefits of the same. This work fills that gap, proposes a new approximate encoding scheme, and shows the system-level benefits when deployed in DNN accelerators.

9 Conclusion

We have experimentally shown the system level area, energy, latency, and accuracy benefits of using various data encoding schemes (such as floating point, integer, and posit number systems) and their corresponding multipliers in DNN accelerators. When storing data of DNNs, we bring out the distribution of regime bits in a posit number encoding scheme and show that $k = -1$ is the most common case. Based on this distribution, this work proposes an approximate fixed posit encoding scheme and a corresponding multiplier that results in 3.1× less energy and 3.2× less area at the system level while increasing the speedup by 2.2× compared to state-of-the-art Simba architecture with a 16-bit floating point scheme, without loss of accuracy (~ ±1%).

10 Acknowledgement

This work is supported through grants received from Prime Minister Research Fellowship, Intel India Research Fellowship, Science and Engineering Research Board (SERB), Government of India, under CRG/2018/005013, MTR/2019/001605, SPR/2020/000450, SERB SUPRA, YFRF Visvesvaraya PhD fellowship and Semiconductor Research Corporation (SRC) through contracts 2020-IR-3005 and 2020-IR-2980. This work is partially supported through Ashoka University startup and Huawei Technologies India grants.

References

[1] Yiran Chen, et al. 2020. A survey of accelerator architectures for deep neural networks. *Engineering* 6, 3 (2020), 264–274.
[2] Lei Deng, et al. 2020. Model Compression and Hardware Acceleration for Neural Networks: A Comprehensive Survey. *Proc. IEEE* 108, 4 (2020), 485–532. https://doi.org/10.1109/JPROC.2020.2976475
[3] Joydeep Kumar Devnath, et al. 2020. A mathematical approach towards quantization of floating point weights in low power neural networks. In *2020 33rd International Conference on VLSI Design and 2020 19th International Conference on Embedded Systems (VLSID)*. IEEE, 177–182.
[4] Xiaochen Guo, et al. 2010. Resistive computation: Avoiding the power wall with low-leakage, STT-MRAM based computing. *ACM SIGARCH computer architecture news* 38, 3 (2010), 371–382.
[5] John L Gustafson and Isaac T Yonemoto. 2017. Beating floating point at its own game: Posit arithmetic. *Supercomputing frontiers and innovations* 4, 2 (2017), 71–86.
[6] Kaiming He, et al. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.
[7] Navjot Kukreja, et al. 2019. Training on the Edge: The why and the how. In *2019 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. 899–903. https://doi.org/10.1109/IPDPSW.2019.00148
[8] Yann LeCun, et al. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 11 (1998), 2278–2324.
[9] Peter Mattson, et al. 2019. MLPerf training benchmark. *arXiv preprint arXiv:1910.01500* (2019).
[10] Naveen Muralimanohar, et al. 2009. CACTI 6.0: A tool to model large caches. *HP laboratories* 27 (2009), 28.
[11] Angshuman Parashar, et al. 2019. Timeloop: A systematic approach to dnn accelerator evaluation. In *2019 IEEE international symposium on performance analysis of systems and software (ISPASS)*. IEEE, 304–315.
[12] Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014).
[13] Sumit Walia, et al. 2021. Fast and low-power quantized fixed posit high-accuracy DNN implementation. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 30, 1 (2021), 108–111.
[14] Yannan Nellie Wu, et al. 2019. Accelergy: An architecture-level energy estimation methodology for accelerator designs. In *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 1–8.
[15] Wm A Wulf and Sally A McKee. 1995. Hitting the memory wall: Implications of the obvious. *ACM SIGARCH computer architecture news* 23, 1 (1995), 20–24.
[16] Brian Zimmer, et al. 2020. A 0.32–128 TOPS, scalable multi-chip-module-based deep neural network inference accelerator with ground-referenced signaling in 16 nm. *IEEE Journal of Solid-State Circuits* 55, 4 (2020), 920–932.