Dynamic deep bitwidth quantization of Neural Networks.

Rohit Gupta, Sreekrishna Ramaswamy
University of California San Diego,
sramaswa@eng.ucsd.edu, r4gupta@eng.ucsd.edu

Abstract— **Due to the diminishing nature of returns offered by Moore's law, computer architects are currently relying on the design of hardware accelerators to satisfy their relentless quench for better performance at lower power. With the advent of compute intensive workloads such as Deep Neural Networks, we are currently witnessing a paradigm shift in the nature of workloads that computer architects have normally seen. To support high performance with enough energy efficiency, architecture needs to be tightly correlated with algorithms. This paper builds upon the insight that the bitwidth of Neural Networks could be reduced without worrying too much about the accuracy. However, to alleviate the loss of accuracy, the bitwidth requirements for these networks may vary from layer to layer. Designing hardware accelerators with fixed bit-widths for these networks would lead to non-idealistic design for other networks and thus, yield a sub-optimal point on the Pareto curve. Thus, it is mandatory to support multiple bit-width quantization for deep bit-width quantization of neural networks. This work enhances upon the previous work conducted by Hardik et. al [1]. Their architecture primarily focuses on the design of powers of 2 bit-width accelerators using a dynamically composable architecture. Unfortunately, a lot of audio applications require support for a 6 bit-width operation. Our work primarily focuses on modification of the baseline architecture for support of 6 bit quantization along with 2, 4 and 8 bits.**

I. INTRODUCTION

With the increasingly extensively range of neural network applications, there's been a rise in the design of accelerators that could cater to these high performance and energy efficiency needs. Every neural network undergoes two phases namely training and inference. Nvidia's GPUs have been the traditional architecture used for the same. The main reason behind the success of Nvidia is related to the Compute Unified Device Architecture (CUDA). They have designed their entire computing stack for accelerating High Performance Computing (HPC) applications. As we transition on the move from data-centers to the edge, we need to ensure that hardware processing of neural networks happens at the cost of low power dissipation (lower energy requirements) .

Approximate computing is a paradigm of computing wherein the hardware being designed is based on the inherent accuracy-energy tradeoffs that exist in these workloads. Hardik et. al [1] have leveraged the following three algorithmic properties of DNNs to introduce a novel acceleration architecture (called BitFusion).

1) Most of the operations inside a DNN are Multiply-Adds.
2) The bitwidth of the operations can be reduced with no loss in accuracy.
3) However, to preserve accuracy, the bitwidth varies significantly across DNNs and may even be adjusted for each layer

To solve these deficiencies, this work introduces the concept of runtime bit-level fusion/ decomposition of a new dimension in the design of DNN accelerators. We explore this dimension by designing a bit-flexible DNN accelerator, which comprises an array of processing engines that fuse at the bit-level granularity to match the bitwidth of the individual DNN layers. The bit-level flexibility in the architecture enables to reduce the computation and communication at the finest granularity level with no loss in accuracy. The following insights have motivated the work on deep bitwidth quantization for acceleration of Deep Neural Networks.

First, the number of bit-level operations required for the multiply operator is proportional to the product of the operands' bitwidths and scales linearly for the addition operator. Therefore, matching the bitwidth of the multiply-add units to the reduced bandwidth of the DNN layers, almost quadratically reduces the bit-level computations. This strategy will affect the acceleration since the large majority of DNN operations (¿ 99%) are multiply-adds.

Second, energy consumption for DNN acceleration is usually dominated by the data accesses to on-chip storage and off-chip memory. Therefore, Bitfusion comes with encoding and memory access logic that stores and retrieves the values with the lowest required bandwidth.

Third, this work takes inspiration from the previous work that DNNs could be quantized using without degradation in classification accuracy. This opportunity exists across different network architectures.

Fourth, audio applications in specific require bit-width quantizations of 6 bits each for inputs and weights for their accurate computation. This is a case that's missing from the paper proposed by Hardik et al. This work actually supports a 6-bit width architecture along with the support for 2-bit, 4-bit and 8-bit quantization support.

Fifth, 6-bit width implementations could be converted to a 8-bit quantization, but because of the wastage in area, power and latency. Using our customized 6-bit quantization based architecture and compiler, we are able to generate savings on power, area and compute resources.

This paper has been divided into four major sections. Section II covers the Compiler and ISA, section III details the Micro-architecture of the design, section IV explains the methodology adopted for benchmarking and section V concludes the work by proposing possible future enhancements.

II. Architecture

The baseline architecture designed and simulated by Hardik et. al [1] is shown as part of left side of figure 1. Whereas, the one on the right depicts the designed architecture. As one could notice, there are three fusion units that are present in the figure on the right, whereas there are four of them in the fusion units that are present in the figure on the left. As it can be observed from the baseline architecture

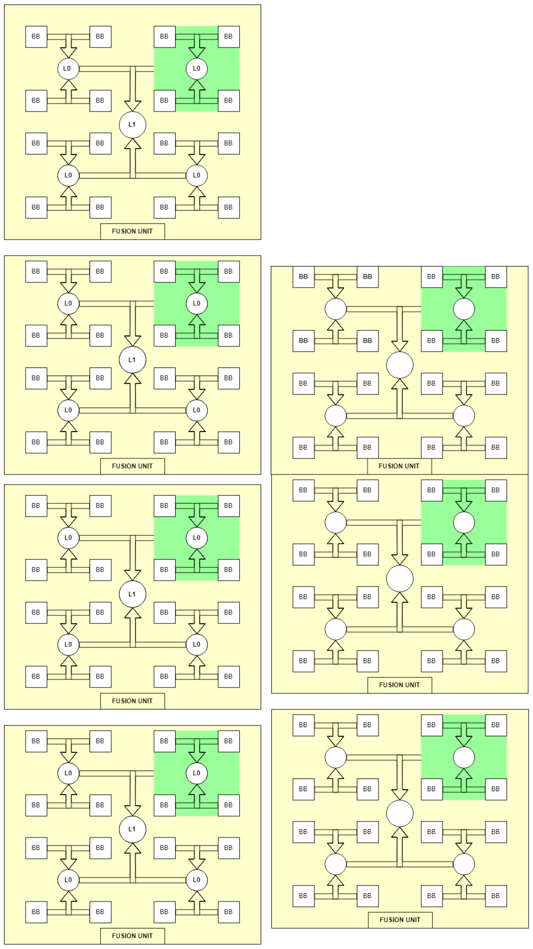

Fig. 1. Baseline and modified versions of the architecture

that the fusion unit and the updated fusion unit have a minor difference. This benefit could be seen when we include the support for the 6 bit quantization within the baseline architecture. If we use the 6 bit quantized values within the bit-fusion unit, we could see that the 6b x 6b computations could be done more efficiently using just 3 units with our modified architecture. Figure 1 right describes the allocation of resources for computation of 6b x 6b products using the compiler. For instance, let's assume a computation of 6b x 6b in both our architectures. For the baseline architecture, the most significant two bits would have been padded with zeros for computing the effective product. Whereas, in the modified architecture, we have issued those bits to the bit-bricks that were previously calculating zeros. There would be a minor hardware change in the architecture that would be required to support this change. We need to introduce a 4:1 multiplexer to support this change in the architecture. The inputs to the multiplexers would be the outputs of the the fusion units highlighted in green. These outputs would be connected to four shifters, each shifting by 0, 2, 4 and 8 respectively. These signals would then be sent to the multiplexer as inputs and thus, we would achieve the desired result. Using our custom compiler that we have designed to support this architecture, we are able to support the 6b x 6b architecture that has been designed to support quantization for audio applications. In the case of the baseline architecture, the authors do not support 6 bit quantization and every 6 bit quantized weight would have to be converted from a 6 bit to an 8 bit quantized weight. Thus, we waste the available compute resources, power and latency required in computing the final output. To avoid this wastage of the power, latency and compute resources, we have designed the updated fusion unit architecture, depicted in figure 1 (right) and the compiler is more complex than it was previously to support the same.

To minimize the computation and communication at the finest granularity, this architecture dynamically matches the architectures of the accelerator to the bitwidth required for the DNN, which may vary layer by layer, without any loss in the accuracy. Architecture implements a collection of bit-level computational elements, called BitBricks, which dynamically compose to logically construct Fused Processing Engines (Fused-PE) that execute DNN operations with the required bitwidth. Currently, this architecture only supports 2 bit, 4 bit, 6 bit and 8 bit quantizations. Specifically, Fused-PEs provide bit-level flexibility for multiply-adds, which are the dominant operations across all types of DNNs. Below, we discuss how BitBricks can be dynamically fused together to support a range of bit-widths, yet provide a significant increase in parallelism when operating at lower bitwidths.

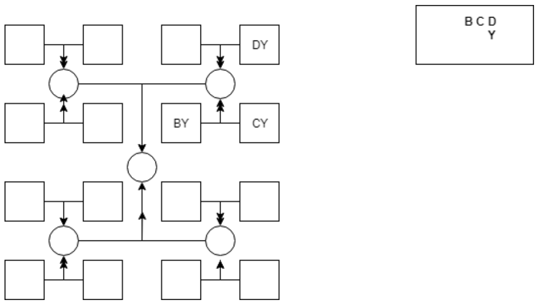

Fig. 2. 6 x 2 bit quantization bit-brick distribution

Figures 2 and 3 describe different variants for quantizing the weights and inputs for 6 bit and 8 bit configurations. (a) shows the configuration for A x 8, (b) depicts the configuration for A x 4 and (c) portrays the configuration for A x 8 respectively, where A represents the bit-width and A could be either 6 and 8. We aren't displaying the outputs for A = 2 and 4 primarily because, there won't be any difference between the baseline and the proposed architectures for those neural networks. As portrayed in the figure shown above, we see that the fusion units arrange the bit-bricks in a 2-dimensional physical grouping, called Fusion Unit. Each Bit-Brick (BB) in a Fusion Unit can perform individual binary (0,

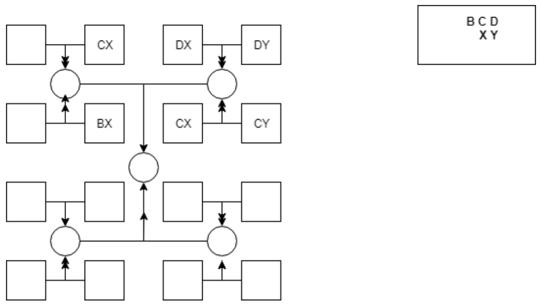

Fig. 3. 6 x 4 bit quantization bit-brick distribution.

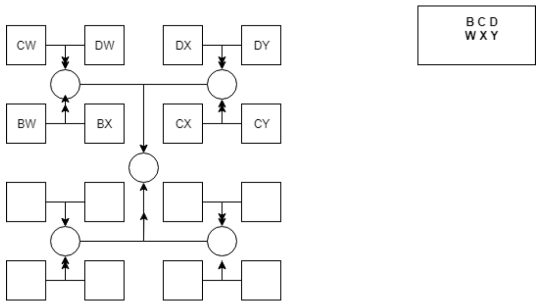

Fig. 4. 6 x 6 bit quantization bit-brick distribution.

+1) and ternary (-1, 0, +1) multiply-add operations. As figure 1 shows, the Bit-bricks logically fuse together at run-time to form Fused Processing Engines (Fused-PEs) that match the bitwidths required by the multiply-add operations of a DNN layer.

III. Instruction Set Architecture (ISA)

Our implementation consists of following instructions -

- ld-mem
  - Used for loading 4B from main memory to the buffers
  - Operands width depends on the memory supported and include the target buffer as well
- st-mem
  - Used for storing 4B from buffers to the main memory
  - Operands width depends on the memory supported
- ld-buf
  - Used for loading 1B from buffers - IBUF, WBUF
  - Operands width depends on the memory supported
- mul2
  - Used for multiplying 2bits in the *bit-brick*
  - Operands widths can vary depending on the size of IBUF, WBUF as they specify which address to get the bits from
- staddr
  - Used for storing the data from *column-based accumulator* to an address in OBUF
  - Operand can vary depending on size of OBUF
- shconfig *16-bit operands*
  - Used for configuring the Level 1 Shifters shown in figure 5

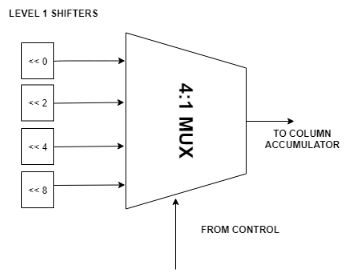

Fig. 5. Level 1 shifter structure

IV. COMPILER

In our approach, all the complexities of design have been shifted to the compiler. So, the compiler is responsible for scheduling instructions on to the *bit-bricks* for all configurations from table VI as well as generating instructions to control the flow of data to the IBUF/WBUF as well as from the *column-based accumulator* to the OBUF. Under the hood features are described in below sub-sections.

A. Fully parametrized

The compiler parses through a model designed using *keras* and figures out the specifications of the layer to simulate in the hardware simulator. It actively extracts the information mentioned in table I and generates the instructions accordingly. The current implementation does not generate encoded instructions but they can be easily integrated in future.

TABLE I
PARAMETERS EXTRACTED AND USED BY COMPILER

S. No.	Parameter	Example
1	Input image shape	60000 28 28 (for MNIST)
2	Kernel shape	3 3 3
3	Padding (input image)	0
4	Bitfusio Dimensions	16 16 (256 Fusion Units)
5	IBUF size	256 (in Bytes)
6	WBUF size	128 (in Bytes)
7	OBUF size	16384 (in Bytes)
8	Input Quantization	2 (in bits)

B. Micro-managing SRAM buffers

To emit instructions for a *bit-brick*, it is imperative for the compiler to manage the contents of the IBUF and WBUF. Thus, the compiler actively tracks the usage of input images and the kernels inside the buffers in chunks of 4B and sorts them using *Least Recently Used* policy. This policy is in-line with how a window traverses in a convolution (spatiality) and hence it helps the compiler in emitting *ld-buf* in case of a miss in the buffers along with replacing the least frequently used entry from the buffer. It is currently used in IBUF and WBUF only but can be implemented for OBUF as the input becomes longer.

V. MICRO-ARCHITECTURE

Often DNNs offer a high degree of parallelism and benefit significantly from the increasing number of Fusion units available within the accelerator's area budget. Therefore, it is essential to minimize the overhead of control in the accelerator by not only maximizing the number of Fusion Units but also minimizing the overhead of dynamically constructing Fused-PEs, thereby integrating the maximum number of BitBricks in the area budget. Second, on-chip SRAM and register-file accesses dominate the energy consumption while accelerating DNNs. Therefore, it is essential to reduce the number of bits exchanged with on-chip and off-chip memory while maximizing data reuse.

The enhanced version of the micro-architecture is very similar to the one designed by Hardik et. al [1] in their seminal paper. To support 6-bit quantization for weights and inputs, we have used the extra bit-bricks that would be available in the architecture had there not been padding of zeros to convert a 6-bit number to a 8 bit number. In other words, we have used the bit-bricks for useful computation instead of just computing zeros. To accommodate different possible combinations of weights and inputs, for instance, 6x4 or 6x8 or 6x2, we need to use the outputs from those units differently. Thus, we have used a 4:1 multiplexer at the outputs of those units which are then connected to the different shifter units present in the design. The control signal for the multiplexer would be provided based on the command and thus, the necessary output would basically be driven by software. This results in an extra area penalty as compared to the existing architecture, but provides other benefits such as reduction in effective utilization of resources, power and latency, along with a support for 6-bit quantization of inputs and weights in the neural network. The design for the additional support has also been shown in the figure.

VI. Benchmark Methodology

To evaluate the correctness of this design, we have designed our custom ISA, compiler and software simulation of the architecture, micro-architecture as part of our work. To simulate and verify the correctness of our architecture, we have taken the implementation of LeNet-5 architecture in Keras. We have designed our custom parser that converts this code into a format that's understandable by the compiler. The features of the parser are that it supports flexible layer conversion.

For proof of concept, our simulation infrastructure currently generates opcodes based on the input dimensions of the kernel image, input image size, number of filters in the kernel, input buffer size, output buffer size, weight buffer size. Based on these inputs, our compiler starts generating commands to the different fusion units and each bit-brick within each fusion unit. For simulation purposes, we have restricted our architecture to have a dimension of 16 x 16. To compare the efficiency of our approach to the one proposed by Hardik et. al [1], we have taken our architecture and simulated it with 8-bit quantization and 6-bit quantization. We also support other bit-width quantizations, but currently, we only support 4 bit and 6 bit quantizations. With the other hyper-parameters remaining fixed, we get an apples to apples comparison metric between the efficiency of the two architectures. The following metrics could collectively or individually be considered to be equivalent to the efficiency of our architecture - fraction of fusion units being utilized and total number of bit-bricks used across all the cycles. Table 1 captures these metrics for different quantizations and provides a good comparison between the efficiencies of the different architectures.

TABLE II
Comparison between effective utilization of fusion units

Bitwidths	Bit-bricks(%)	Fusion units (%)
6 x 2	18.7345	24.9799
6 x 4	37.4691	45.9588
6 x 6	56.2037	74.9383
6 x 8	74.9383	99.9177
8 x 2	24.9794	45.9588
8 x 4	45.9588	45.9588
8 x 6	74.9383	99.9177
8 x 8	99.9177	99.9177

A. Results

Table VI captures the resource utilization of fusion units and bit-bricks for different quantizations where our architecture performs better than the baseline architecture. Using the compiler, we have been able to issue instructions to the bit-bricks and the fusion units. The different values for bitwidths are 6 x 2, 6 x 4, 6 x 6 and 6 x 8. Whereas, the different values of bitwidths supported for the other side of the configuration are 8 x 2, 8 x 4, 8 x 6 and 8 x 8 respectively. As it could clearly be seen, we have the values in the top-half of the table being lesser than or equal to the corresponding values in the bottom half of the table. This clearly signifies that the modified architecture is better than the baseline architecture in that it support a 6 bit quantization and performs better than the traditional 8 bit architecture.

VII. CONCLUSIONS AND FUTURE WORK

Deep Neural Networks are highly parallelizable primarily due to the lack of data dependencies within a single layer of neuron. Traditional systems are good at sequential execution and as current workloads use very highly parallel execution models, we need to adopt a different approach to solve this problem. Our paper proposed an enhancement to the already existing architecture that was designed by Hardik et al in their seminal work. This work looked at an approach towards supporting 6-bit quantizations which are extremely crucial for audio neural network applications. Through this work, we have seen that we could achieve a better resource utilization and latency by using the extra available bit-bricks for designing systems.

In this paper, we proposed an effective enhancement to the work proposed by Hardik et. al [1] as part of their work. There are a few shortcomings such as support for inter-column support for addition. We wish to support the same as it would enable this architecture to perform more efficiently for much larger neural networks. As part of the future work, one could take this proposed architecture, implement it in RTL and run through the backend flow to generate the area, power numbers of a 65 nm technology node.

References

[1] Sharma, H., Park, J., Suda, N., Lai, L., Chau, B., Chandra, V. and Esmaeilzadeh, H., 2018, June. Bit fusion: Bit-level dynamically composable architecture for accelerating deep neural networks. In Proceedings of the 45th Annual International Symposium on Computer Architecture (pp. 764-775). IEEE Press.