BitFuSCNN: A flexible bitwidth inference accelerator for Compressed-sparse Convolutional Neural Networks

Max, Edwin, Arpan

March 2021

1 Introduction

Deep Learning has emerged to be an important contributor in solving problems in a variety of domains like speech, vision etc and Convolutional Neural Networks (CNNs) form a large proportion of the Deep Learning architectures deployed for some of these tasks. The availability of large amounts of data and advances in high performance computing have given rise to several novel applications. Developing a CNN based deep Learning Application has two parts - Training and Inference. During training the input is fed to a network and after computation with each layer weights, the output is generated. This is called a forward pass. The output is then compared to the expected output from the training data and the weights are updated. This is called a backward pass. After the weights are learned after several iterations, the trained network is deployed into production and used to make predictions on new data which is called inference. Inference only entails a forward pass and generating the output. Typically training is done on large amounts of data on massive server farms with GPUs or ASICs, inference is often done on both low-powered devices and in the cloud. There is a need to have on-device inference to enable real-time inference without the network latency and worrying about data being sent to a remote server for inference. This project proposes an accelerator to improve CNN inference performance.

2 Quantized Neural Network Models with variable bitwidths for inference and Sparsity in CNNs

We use quantized networks to achieve a speedup over the SCNN [3] implementation. This means we also used the sparsity outlined in the SCNN paper. We assumed a quantized network to 2, 4 or 8 bits and did all of our accumulation

at 4, 8, or 16 bits respectively. We expect a reasonable network would work well with the chosen quantitization levels as illustrated by the SCNN, EIE [2], and Bitfusion [4] works. We also assume that our network would have similar activation and weight densities as AlexNet in SCNN. So 60% of weights are zero.

3 BitFuSCNN Processing Element Architecture

BitFuSCNN strives to combine the benefits of both sparsity and quantization in CNNs. The goal is to use the SCNN architecture and add BitFusion to it to support variable bitwidth multiplication for faster inference of quantized models. The variable bitwidth multiplication follows the concept of BitFusion to enable fusing of bit bricks to perform multiple lower precision multiplications from one 8bit by 8bit multiplier. We have implemented the cycle accurate simulator in python for one processing element of SCNN with BitFusion incorporated to gauge the performance, power and area of our approach. A verilog implementation has been completed and used for synthesis.

The design is similar to that of SCNN, except all of the bitwidths are variable. Instead of a fixed 4x4 multiplier operating on 16 bit values we have a variable bitwidth multiplier operating on 16x16, 8x8, or 4x4 input either 2, 4, or 8 bits wide. This is similar to the implementation in Bitfusion.

4 Experiment

We wanted to measure the time it takes to process the two AlexNet layers four and five. This would be our benchmark to compare against SCNN to see if adding the Bitfusion low bitwidth multiplier provided a tangible benefit. We used a python cycle accurate simulator that passed the same unit tests as the verilog code. We then generated a set of input and weight matrices for one output channel with the given sparsity parameter of zeros. So a sparsity of 0.7 would have on average seven zeros for every ten values. We then run through the three layers for the last two AlexNet layers: Convolve layer 4, exchange layer 4 while convolving layer 5. Exchange layer 5. The input activations for layer 5 were randomly generated as we only compute one output channel. Figure 1 shows the absolute performance in clock cycles. We are assuming that we can achieve a similar cycle time as SCNN of 1 GHz but have not evaluated this.

The best comparison to SCNN is the multiplier idle time. If we have similar cycle times we can compare the stall count or multiplier utilization for the last two layers of AlexNet. We have 4.66 stall cycles inserted yielding a 17% multiplier utilization while SCNN manages a 50% utilization. This means SCNN process on average 8 16-bit outputs per cycle while we manage 10 2-bit outputs. See table 2 for other bitwidths. The improvement is somewhat constrained by the number of nonzero weights, however bank conflicts dominate multiplication throughput.

Table 1 shows the scaling of stalls with respect to bank size and sparsity.

Since we plateau after 64 banks we chose to limit to 64 banks instead of the expected 512 as extrapolated from SCNN. This is almost certainly due to a poorly designed mapping function. The area overhead is shown in table 3.

The code can be found in the repository [1].

5 Future Work based on Results

As demonstrated by the above experiments the increase in performance is not proportional to the increase in area. This due to the fact that we have to use a 256 bank buffer to completely process all the outputs of the multiplier when configured to 2-bit mode, ie, 256 parallel multiplications. This causes the area of the accumulator bank to increase quadratically to accommodate the worst case of all outputs mapping to different banks. But the worst case would rarely occur in practice especially since we are using sparsity in our operations too. Hence, reducing the bank size to 64 only results in a 4% increase in stall cycles. That's 4x reduction in area for the banks while a minuscule impact to performance.

Another approach to resolve the area bloat is to make the accumulator adders and the bank itself Bit Fused, ie, dynamically change the precision of the adders for parallel addition. The area impact in this case is smaller than having 4 separate adders and you also get the parallelism throughout the chain of operation. This functionality is yet to be explored and a good place to extend this work.

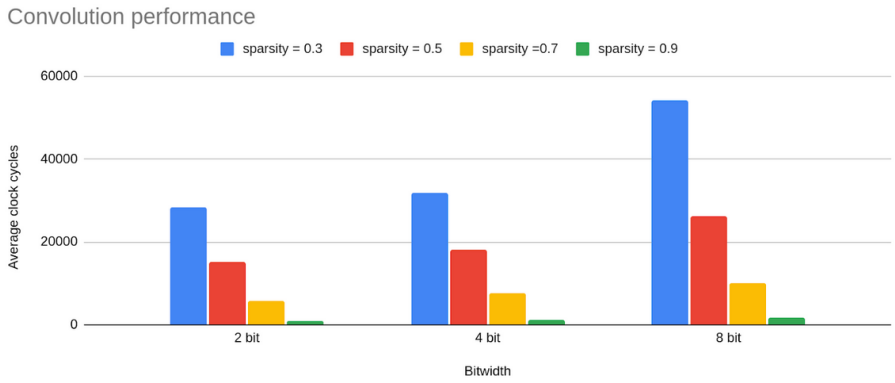

Figure 1: Comparison of clock cycles for a given bitwidth and sparsity.

[1] On average 3.6 weights in each kernel

[2] Based on linear scaling of SCNN result in terms of bitwidth.

Bank Count	Stall Cycles / Multiply 90% Sparsity	Stall Cycles / Multiply 70% Sparsity	Stall Cycles / Multiply 60% Sparsity
16	1.63	3.61	4.78
32	1.10	3.43	4.66
64	1.05	3.40	4.66
128	1.06	3.39	4.69
256	1.08	3.37	4.66
512	1.03	3.40	4.66
1024	1.03	3.39	4.68

Table 1: Number of stalls per multiply of 16 x 16 2 bit values.

Activation Width	Stall Cycles / Multiply	Multiplier Size	Multiplies / Cycle
2	4.66	16x16[1]	10
4	2.72	8x8[1]	7.7
8	1.06	4x4[1]	7.0

Table 2: Number of stalls per multiply of 16 x 16 2 bit values.

PE Component	BitFuSCNN		SCNN	
	Size	Estimated Area[2]	Size	Area
IARAM + OARAM	Off Chip	0	20 KB	0.031
Weight FIFO	Off Chip	0	0.5 KB	0.004
Multiplier array	16 ALUs	0.008	16 ALUs	0.008
Scatter Network	512bit x 64 Crossbar	0.052	256bit x 32 Crossbar	0.026
Accumulator Buffers	32 KB	0.192	6 KB	0.036
Other		Not Estimated		0.019
Total		≥ 0.252		0.019

Table 3: BitFuSCNN area estimates.

References

[1] *BitFuSCNN*. https://github.com/Arpan-Shankar-Dutta/bitfuscnn. Accessed: 2021-03-21.

[2] Song Han et al. "EIE: Efficient Inference Engine on Compressed Deep Neural Network". In: *Proceedings of the 43rd International Symposium on Computer Architecture*. ISCA '16. Seoul, Republic of Korea: IEEE Press, 2016, pp. 243–254. ISBN: 9781467389471. DOI: 10.1109/ISCA.2016.30. URL: https://doi.org/10.1109/ISCA.2016.30.

[3] Angshuman Parashar et al. "SCNN: An Accelerator for Compressed-sparse Convolutional Neural Networks". In: *CoRR* abs/1708.04485 (2017). arXiv: 1708.04485. URL: http://arxiv.org/abs/1708.04485.

[4] Hardik Sharma et al. "Bit Fusion: Bit-Level Dynamically Composable Architecture for Accelerating Deep Neural Networks". In: *CoRR* abs/1712.01507 (2017). arXiv: 1712.01507. URL: http://arxiv.org/abs/1712.01507.