CS498 Bachelor's Thesis Project
Mid Semester Report
Accelerator Design for Machine Learning

Supervisor: Prof. Hemangee K Kapoor

Saurabh Baranwal (180101072) Amey Varhade (180101087)

Introduction

Hardware acceleration refers to the process by which an application will offload certain computing tasks onto specialized hardware components within the system, enabling greater efficiency than is possible in software running on a general-purpose CPU alone. They usually have novel designs and typically focus on low-precision arithmetic, better dataflow architectures or in-memory computing capability.

Hardware acceleration has many advantages, the main being speed.
Machine Learning Accelerators are the specialized hardware components designed to improve the efficacy of machine learning-based applications including but not restricted to computer vision, artificial neural networks, and other data-driven or sensor-driven tasks.

Convolutional neural networks (CNNs) have proven to be a disruptive technology in most vision, speech and image processing tasks.
However, the ever-increasing complexity of these algorithms poses challenges in achieving real-time performance. Specifically, CNNs have prohibitive costs in terms of computation time, throughput, latency, storage space, memory bandwidth, and power consumption.
Additionally, these networks require millions of parameters for their functioning, which translates to very high memory storage and bandwidth requirements.

The amount of data that needs to be processed is increasing at the rate of 1000x per decade. Also the number of parameters for these networks has been increasing at an exponential rate. Since 2012, 3000X increase in the computational time, and 10X increase in memory requirements. Hence, the need to efficiently run these networks is urgent.

In CNNs, more than half of the computation time is contributed by the convolutional layers. The pooling and Fully Connected layers take relatively less time.

Hence, in the last 5 years, a lot of work has been done by the scientific community to mitigate these costs.

Overview

1 Reduction in Time

1.1 Exploiting Inherent Parallelism

There are nested loops, we can pick up any axes and parallelize them specifically out of the 6 available axes.

1.2.1 Pattern based Computation reduction

Number of unique weights is limited by the maximum number of bits.
Here we exploit the associativity in multiplication There is a time vs space tradeoff.

1.2.2 Cavoluche [Wang et. al.]

This is a Software and Hardware based solution which uses Principal Component Analysis (PCA) to reduce the size of the weights and stores the intermediate results
Implementation involves NVIDIA TX1 vs RTL synthesized Cavoluche (Synopsys Design Compiler, 65nm TSMC)

1.3 Reducing inefficient computations

Pruning values below a certain threshold (or zero) helps in efficient computation
Saves time and energy both. Redundancy filters are used to remove the ineffectual computations.

1.4 Prediction Driven Computation Reduction

Computations filtered by ReLu and the Pooling layer are ineffectual. Different techniques for their prediction have been proposed.

1.4.1 Prediction based execution [Song et. al.]

High order bits passed through the comparator and sign unit to predict computations rendered ineffective by pooling and Relu layers. Predicted computations are marked as zero in both tables. But this doesn’t reduce computation time but only reduces power (as PEs are parallel

and this leads to only idleness of PEs as one PE sits idle till its neighbouring PEs complete their computation)

How to reduce computation time?
Allow data fetching in only one direction (either activation or weights). Data from other directions can be fetched from memory. This reduces computation time but increases pressure on the memory system.

1.4.2 SNAPEA

We rearrange weights in such a way to know if the computation overall results are negative at the earliest.

The problem we have chosen to work on is reducing the computation time of CNN's in general. We would be working on reducing the Computation Time involved in the Inference.

To reduce these computations, this paper offers a solution that leverages a combination of runtime information and the algorithmic structure of CNNs.

This technique cuts the computation of convolution operations short if it determines that the output will be negative.

Predictive Early Activation Technique has two modes, namely Exact mode and Predictive mode.

These cut the computation of convolutional operations short if it predicts final output of a computation to be negative during runtime. In this way we prune the ineffectual computations

1. Perform computation of add and multiply with positive weights
2. Sort the negative weights in decreasing order of their absolute values
3. In each cycle, perform computation using one negative weight in order. If at any stage, overall computation becomes negative then terminate.

Above mode is known as **exact mode.**
Exact Mode
We rearrange weights in such a way to know if the computation overall results negative at the earliest. Perform computation of add and multiply with positive weightsSort the negative weights in decreasing order of their absolute value
In each cycle, perform computation using one negative weight in order.
If at any stage, overall computation becomes negative then terminate.

Predictive mode
Threshold value predetermined along with no. of MAC operations for that threshold. Weights are divided into 3 classes: predictive, positive and negative. Computations are done on

predictive weights and when compared with threshold, we give prediction if computation will be ineffective or not.
Some false positives may occur. The advantage is that computations are reduced to size K*K - N where N is the number of predictive weights.

Selecting the weights with the larger magnitude ignores the contributions of input values which are, to a large degree, random and data dependent
SNAPEA exact mode may not be effective for models where there are many negative weights with similar magnitude.

$$\mathrm{Op}(o_{l,k}^{d},\mathrm{Th}_{l}^{k},\mathrm{N}_{l}^{k})=\begin{cases}\mathrm{N}_{l}^{k}, & \text{if } \mathrm{PartialSum}_{\mathrm{N}_{l}^{k}}\leq\mathrm{Th}_{l}^{k},\\ \mathrm{Idx}_{w^{-}}, & \text{if } \mathrm{PartialSum}_{\mathrm{N}_{l}^{k}}>\mathrm{Th}_{l}^{k} \textit{ and } \mathrm{PartialSum}_{w^{-}}\leq 0,\\ \mathrm{C}_{in,l}\times\mathrm{D}_{l}^{k}\times\mathrm{D}_{l}^{k}, & \text{otherwise}\end{cases}$$

$$\min_{\mathrm{Th},\mathrm{N}} \sum_{d\in\mathscr{D}}\sum_{l\in L}\sum_{k\in K_l}\sum_{o\in O_{l,k}^{d}}\mathrm{Op}(o,\mathrm{Th}_{l}^{k},\mathrm{N}_{l}^{k})$$

$$\text{subject to} \quad \mathrm{Accuracy}_{CNN}-\mathrm{Accuracy}_{SnaPEA}\leq\varepsilon$$

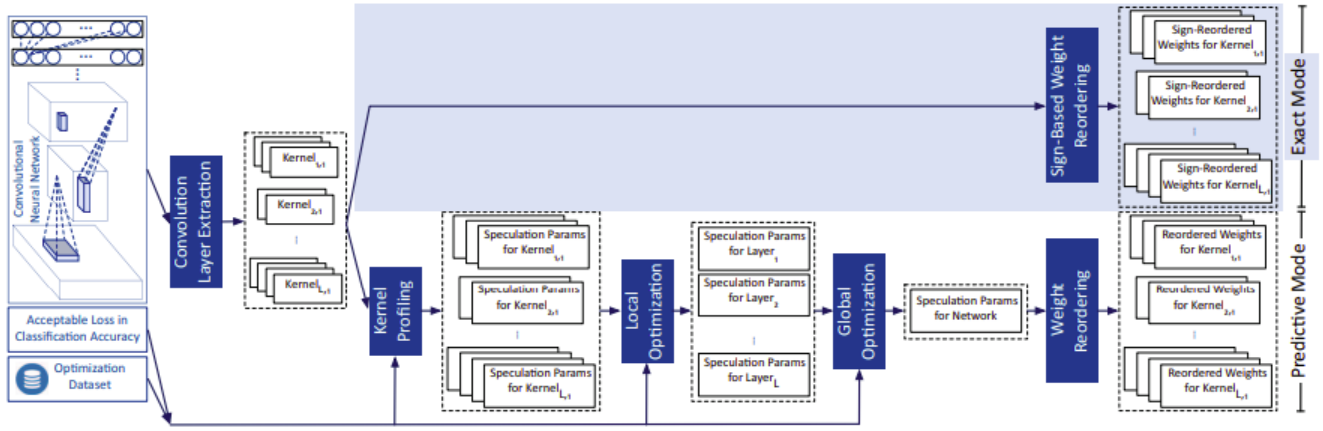
Sort the weights in ascending order, partition them into a number of smaller groups, and selects the weight with the largest magnitude from each group (called **predictive weights)**

Then keep the remaining positive and negative weights in order. Computations are first done on predictive weights and then compared with a threshold value. If less than threshold, predict negative output and terminate.

If greater than threshold, do further computations over all positive values and then over negative values. If at any negative weight index, partial computation comes negative, terminate. Else, terminate after computing for every weight value.

Note: SNAPEA model may not be effective for models where there are many negative weights with similar magnitude.

Implementation involves GPU and Custom hardware synthesized on TSMC 45nm standard cell library using Synopsys Design Compiler (SP5)



1.5 Improving the PE Utilization

PEs may be rendered idle, so we can perform future computations on idle PEs improving the time. Delmas et. al. exploited computation cycles involving zero weights to perform computation cycles for non zero weights that are scheduled for a later time instant.

For this, he proposed two designs

Weight Lookahead technique
It has a lookahead window. The technique promotes non zero weights from the left in the lookahead window. For this promotion, we also require activations corresponding to these weights stored in input buffer.Since rows are executed parallely in PEs, row having max effectual weights will decide execution time.

Now it might be the case that some rows have higher no of effectual weights while other rows have zero. We can balance the load using **Weight Lookaside** technique. It allows weight stealing (or computation stealing) from the neighboring rows (within the lookaside window). If at the current timestamp, a row has an ineffectual weight it might steal some effectual weight of a later time stamp from another row in the same lookaside window.

2 Reduction in Memory Footprint

Large memory due to intermediate values, high precision of input and filter weights and storing both effectual and ineffectual data.

2.1 Data and weight compression

Wang designed a memory architecture called memsqueezer. It has an active weight buffer set, which contains all the weights. Base delta partitioning method enables to store a subset of weights by decomposing them into two values: one mean value (usually base), and the other are a set of differences. We save some bits if numbers are stored as offset values. Two tables: Base table and Delta Table, which are indexed by index buffer.
Limitation: It cannot be used for intermediate data in CNNs since the intermediate data changes dynamically.

For intermediate data, Wang proposed a data buffer set. Frequent pattern mining method is used for compression. Frequently seen patterns are identified and replaced with shorter identifiers (like Huffman coding). Area overhead is reduced due to storage of encoded data but it introduces additional computational overheads.

2.2 Reduction of ineffectual bits in Binary representation

There are two types of ineffectual bits: static and dynamic. Static ineffectual bits are those bits which might not be necessary for representation. Dynamic ineffectual bits are the zero bits that are necessary for representation but are useless with respect to computation operations (like multiplication).
Eg: 0010010 has 2 dynamic ineffectual bits and 3 static ineffectual bits.

2.2.1 Removal of static ineffectual bits

STR Proposal. Precision of each CNN layer is defined as maximum static effectual bits required to store all activations for that layer. Since each layer would've required a different multiplier unit, they introduced a serial multiplier unit. It increases multiplication latency by serial data bit-width (b) , thus b such parallel units are used which improves performance by f/b (where f is fixed bit-width). Also, since each layer has a different desired precision, buffer size at output can be customized to save further space.

Dynamic STR determines precision of a group of activations at runtime. Thus, only those activations are considered that are currently being executed. Dynamic approaches are usually more effective but they have more overheads.

2.2.2 Removal of dynamic ineffectual bits from activations

Pragmatic exploits dynamic ineffectuality. Mahmoud further proposed another extension over pragmatic by another convolution operation. For windows other than the first one, differences of pixel values with respect to first window are stored. This reduces memory.
In precision, computation involving dynamic ineffectual bits are not calculated. This improves the latency further.

2.3 Removal of ineffectual bits from weights

We also use bit multipliers for this purpose. LOOM exploits serial processing for both activation and weights, unlike previously which only exploited it for activation and had fixed the weight bit width. Thus, latency may be more but memory used is less (since operations are performed bit by bit for both activation and weights).

2.4 Optimizing performance of effectual bits

Bit serial multipliers have a limitation; hence a different architecture is proposed. 2*2 multiplier units called Bitbrick. It can be fused with other bitbricks to create larger multipliers.
How to fuse databricks?
Spatial
An n bit multiplication can be broken down into several 2*2 bitbricks
Temporal
Multiple multiplications can be performed on same cluster of bitbricks at different times.

LUT

Assume that you have 1 bit activations: X,Y,Z. Now, we multiply them with weights wx,wy,wz (MSB: LSB). There will be 8 combinations of wx,wy,wz and hence they are stored in a Lookup Table for reuse.
Details such as the number of memory buffers, and the nature of the interconnects are very crucial, and also are the key determinants of the performance
When we reuse partial results across filters. This reduces the amount of computation at the cost of increased storage space. There is a time and space tradeoff.

The advantage of reducing precision is a higher computational performance, and a significantly reduced memory footprint but the cost is accuracy. We need to balance this tradeoff well.
For CNNs we can make these adjustments according to layers or type of layers.
The SNAPEA model may not be effective for models where there are many negative weights with similar magnitude.

Threshold acts as a control knob for the accuracy(performance) and the computation involved. A stronger relationship for tradeoff between these needs to be established.

Some false positives may occur. This will directly affect the accuracy, precision, recall and other metrics.

References

SNAPEA https://ieeexplore.ieee.org/document/8416863 (2018)

https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8192499
(FPGA Based)

https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9195436
(Software + Hardware based)

https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8060431
(28nm CMOS, FPGA based, entire analysis in frequency domain) (Fourier transforms)(2017)

https://dl.acm.org/doi/pdf/10.1145/2786763.2694358
(General Accelerator for all ML Algorithms)

[SWAN] https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9045580
(Implementation (RTL synthesized, Synopsys Design Compiler, 65nm TSMC
2020 Paper, improves state of the art performance)

https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9138986

https://www.linleygroup.com/mpr/article.php?url=mpr/h/2020/12245/12245.pdf

https://habana.ai/wp-content/uploads/pdf/habana_labs_goya_whitepaper.pdf

SCNN https://arxiv.org/abs/1708.04485

[Cavoluche] Wang et.al https://dl.acm.org/doi/10.1145/3316781.3317749 (2019)
(Software + Hardware based)

https://ieeexplore.ieee.org/document/8416870

https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8585656

https://arxiv.org/pdf/1912.03413.pdf

https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9062984

https://dl.acm.org/doi/pdf/10.1145/3360307

https://dl.acm.org/doi/10.1145/3079856.3080246

https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9371592

https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9373953