Introduction

We built an 8x8 systolic array based AI accelerator. Our architecture supports SIMD PEs with support for weight stationary setup and output stationary setup. We ran inference for VGG-16 for (1) 4-bit activation, 4-bit weight quantization (4A4W) scheme, (2) 2A4W scheme, and we also implemented 4 alphas. To verify the correctness of our implementation, we replaced layer 27 of VGG with an 8x8 convolution layer and tested this layer on our systolic array. We also replaced the immediate surrounding layers to transition into the 8x8 convolution layers. Our evaluation is primarily focused on the 4 alphas: Alpha 1 is concerned with a software optimization to make training converge up to 5x faster without sacrificing accuracy. Alpha 2 is an implementation of BERT on our accelerator. We split the matmuls in the BERT encoder blocks into 8x8 matmuls and evaluated PTQ with the 4A4W scheme. We also compute the Pareto optimal frontier for finding the optimal energy savings to accuracy tradeoff. Alpha 3 is a hardware alpha where we implemented support for a multicore setup, effectively cutting our inference latency by half, and finally Alpha 4 is a hardware alpha of FPGA synthesis of the multicore alpha.

Baseline Software Desc.

We trained VGG-16 with custom 8x8 and 16x16 convolution layers for Part I and Part II. To support convergence, we trained the model in two distinct steps; **Step 1:** Training the model in Quantization Aware Training (QAT) scheme with trainable α parameter with the 4A4W scheme. Our optimizer was SGD with lr = 10^{-2}, decay = 10^{-5}, max epoch = 300, momentum = 0.9. We used *CosineAnnealing* as our scheduler which is industry standard for QAT. For **Step 2**, we took our previously trained 4A4W model, and fine tuned it for the 2A4W scheme. For fine tuning, we maintained the same parameters, except for the quantization parameters which were trained with lr = 10^{-3} and decay = 0. We were able to achieve 91% and 88% accuracy resp.

Baseline Hardware Desc.

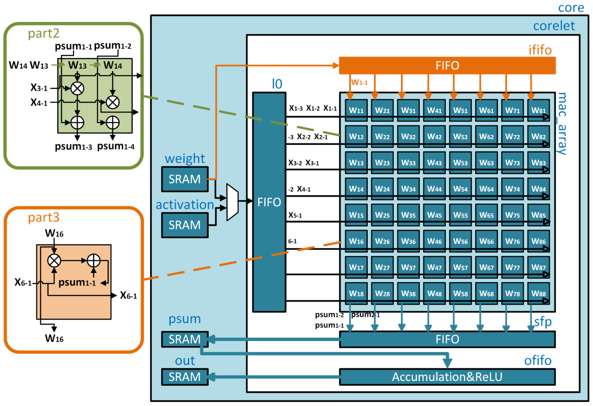

Fig 1: Systolic Array architecture

Our accelerator features an 8×8 2D systolic MAC array backed by dedicated SRAM blocks for weights, activations, psums, and outputs. An L0 FIFO supplies PEs with weights and streamed activations, while an output FIFO aligns partial sums before they are written to a double-buffered SRAM for accumulation by a special function processor. The core is runtime-configurable to operate in 2A4W or 4A4W precision modes, and support both WS and OS mapping.

In the 2A4W precision mode, each PE stores two distinct weights. As a result, kernel loading takes twice as long as in the 4A4W mode, while the western input (in_w) remains 4-bit wide. For the output stationary (OS) mapping, an input FIFO is introduced to load weights through the northern input (in_n).

Alpha 1 (Software Alpha 1)

Custom VGG-16 Architecture

The VGG-16 architecture (augmented with the 8,8 convolution block) can be split into "domains" defined by the major convolution dimensions. The newly added block with 8x8 convolutions would be an 8x8 domain. The block just before the 8x8 block needs to transition from the previous domain into 8x8 and the layer after the 8x8 chokepoint needs to transition into the rest of the 512 domain. Considering this, the architecture could be split into 5 sections: The initial 64, 128, 256 domains, the 256→8 transition domain, the 8x8 "chokepoint" domain, 8→512 transition domain, and the rest of the model's 512 domain.

Training Difficulties

The initial round of training (without any software optimizations) took more than 250 epochs to converge to a 90% and 85% for 4A4W (4-bit activation, 4-bit weight) quantization and 2A4W respectively. We hypothesized that the problem was caused by a constriction in the gradient flow caused by the 8x8 chokepoint domain. We tested this hypothesis by including a simple residual connection between the 256 domain and the 512 domain, jumping over the 8x8 chokepoint. This change markedly improved the training performance but the weight update did not progress as expected.

SkipBlock Optimisation

To alleviate this issue, we introduce a new block called the *SkipBlock*. The SkipBlock is made of an internal 256→512 transition domain that's also quantised with the same quantisation scheme as the rest of the model. This SkipBlock is connected between the 256 domain and the 512 domains. Initially, we trained the entire model without chokepoints or the SkipBlock.

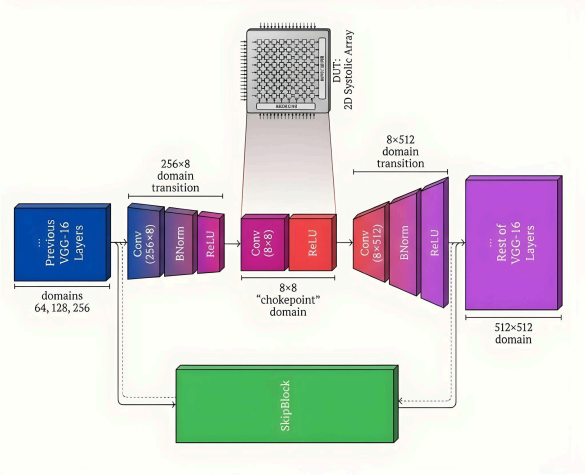

Fig 2: SkipBlock optimization

Once this training was complete, we introduced the SkipBlock and the chokepoint and froze the rest of the model. The SkipBlock provided an easier path for the gradients to flow reducing the "effective gradient resistance" along the chokepoint path. To verify that the chokepoint was still getting some gradient flow, we noticed the weights throughout different epochs and noticed that the weights changed appreciably.

Results

Compared to the baseline where the models took 250+ epochs to converge to the required accuracy requirements, the models now only took ~50 and ~75 epochs resp. for 4A4W and 2A4W quantization schemes. This dramatically improved the turnaround time for experimentation. The time taken for each epoch stayed relatively constant so the total training time was effectively cut down by 5x in the best case scenario.

The maximum achievable accuracy without the SkipBlock (with chokepoint) was 91% and 86% respectively for 4A4W and 2A4W. With the SkipBlock optimisation, the numbers jumped up to 94% and 91% resp.

showing that the SkipBlock not only improved the training time but also improved the best accuracy metrics.

Alpha 2 (Software Alpha 2)

BERT Architecture

For our second alpha, we implemented the transformer based BERT (Bi-directional Encoder Representations from Transformers) model. As the name suggests, it's an encoder only model with 110 million parameters. At a high level BERT is composed of 12 transformer based encoding blocks. Each block consists of two main parts: **Multi-Head Self Attention (MHA)** and **Feed Forward Network (FFN).** Both the MHA and FFN sub blocks are primarily made of matmuls. We evaluated this model with various quantization levels to find a suitable energy vs quantization tradeoff.

Energy Modeling

Since our accelerator only supports 8x8 matmuls/convolution operations, we decided to take a matmul in one or many of the encoding blocks, split them into many 8x8 matmuls, and run them on our accelerator. The goal of this experimentation was to see how sensitive the accuracy of the model is compared to **energy use** of the accelerator. If we consider the total energy use of the model to be ***E*** for full FP32 execution, then we can approximate the optimised energy use for the quantized version as (we're only quantizing the FFN blocks' matmuls):

$$E_Q = (1 - \alpha)E + N(\frac{\alpha E}{96}) + (12 - N)(\frac{\alpha E}{12})$$

where E_Q is the quantized energy of the model, $\alpha \leq 1$ is the *energy fraction* i.e. proportion of energy consumed by FFN compared to the total model, and $N \leq 12$ is the number of blocks quantized. For BERT α sits somewhere around 0.61 based on matmul size approximations; by utilising that, we arrive at:

$$E_Q \approx E(1 - 0.0445N)$$

Methodology

Since BERT is an encoder-only (non seq2seq) model, it doesn't have a notion of "accuracy". Our measure of correctness for the quantized model output was based on *cosine similarity* with the original FP32 outputs with **pre-trained weights** which we assumed to be the ground truth. This measure is known as **model fidelity**. For the quantization, we used our existing 4A4W scheme. For our sensitivity analysis, we performed 2 experiments: (1) Finding out the most sensitive blocks: we quantized one block at a time from 0 to 11 and found out which blocks affected the fidelity the most. (2) Finding the optimal quantization strategy: this experiment was split into 3 more phases where in (i) we quantized blocks 0 to N where N = [0,11] (Naive approach), (ii) sorted the blocks by sensitivity from experiment (1) and then quantized from (0' to N') where N' is the Nth block in the sorted list, and (iii) exhaustive search where for each 0 to N we tried all permutations of 0 to N (2^N possibilities) and found out the **pareto optimal frontier** for quantization.

Results

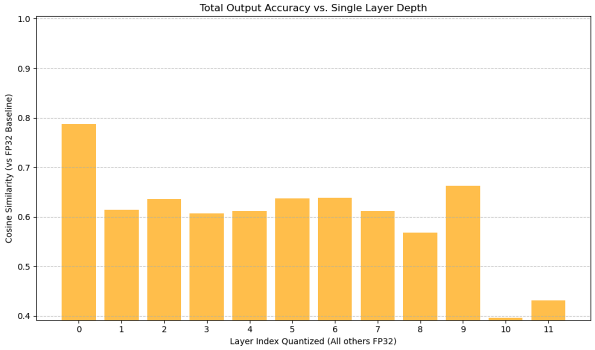

Fig 3: Exp (1) layer sensitivity to quantization

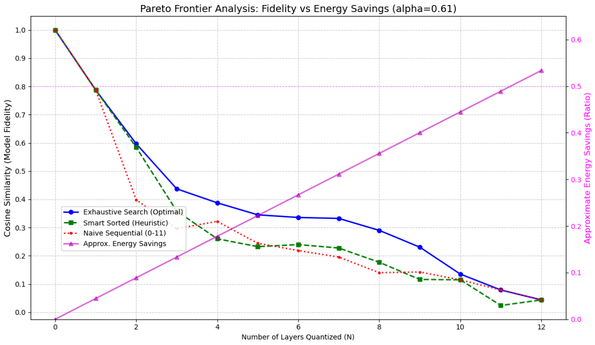

Fig 4: Exp (2) pareto frontier for energy savings

Fig 3, shows us that the model is especially sensitive to quantizing the last 2 blocks. Although conventional wisdom would indicate that quantizing the earlier layers would be worse, BERT is quite special since it has critical residual connections grouping towards the final two layers. From Fig 4, we can see that the fidelity of the model rapidly decays initially with up to N = 4, and then levels out. The blue line indicating the pareto frontier shows the optimal fidelity. The Naive approach is surprisingly close to the frontier attributable to the fact that experiment (1) indicates that the model is only vulnerable to the final layers. The “smart approach” where we quantize the layers based on sensitivity performs better than naive initially which is where we would want to sit to have a good balance between fidelity and energy savings. The purple line shows the approximate energy savings. With a given datacenter TDP per node and a minimum required fidelity, we could set our quantization to the required quantization level based on this chart.

Alpha 3 (Hardware Alpha 1)

Multi-core Wrapper

In the 2-bit-activation-4-bit-weight (2A4W) mode, a single 8×8 systolic array can compute at most 8 output channels per execution. As a result, a convolution layer with 16 output channels must be tiled into two output-channel tiles, requiring the array to be executed twice. This approach not only doubles the execution latency for computing all partial sums, but also incurs additional overhead from repeatedly loading two independent sets of weights into the processing elements, leading to inefficient utilization of compute and memory bandwidth. To mitigate these inefficiencies, we propose a multi-core wrapper architecture that instantiates two independent 8×8 MAC arrays operating in parallel. Each MAC array is preloaded with a distinct set of weights corresponding to a different output-channel tile, while both arrays share the same streamed activation data during execution. Partial sums generated by the two arrays are forwarded concurrently to the output FIFO (OFIFO) and subsequent accumulation stages. Compared to the single-core tiled execution, the multi-core design reduces the total number of execution passes from two to one, achieving approximately 2× throughput improvement and ~50% reduction in execution latency, while avoiding redundant activation streaming and repeated weight loading. By computing both output tiles simultaneously, the multi-core design eliminates the need for output-channel tiling, significantly reducing total execution time and avoiding redundant weight and activation loading, thereby improving overall throughput and efficiency.

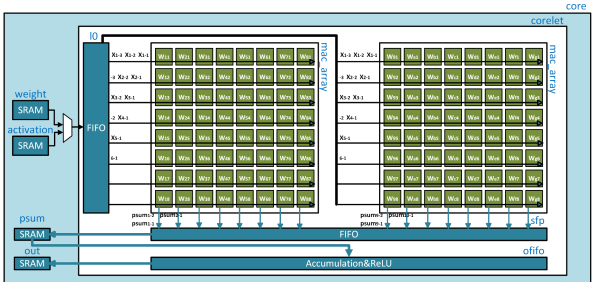

Fig.5 Diagram for Multi-core Wrapper

Alpha 4 & FPGA Synth

FPGA Synthesis of Alpha 3

The results of running FPGA synthesis on both vanilla and multi-core versions are shown below.

	Vanilla	Multi-core
Total Ops per cycle	128	256
Frequency	109.96 MHz	103.9 MHz
Resource Utilization	4-input LUTs: 16865 Reg: 12224	4-input LUTs: 18831 Reg: 13416
Power (mW)	282.14	291.9
TOPs (Trillion Ops per second)	4A4W: 0.0141	4A4W: 0.0266 2A4W: 0.0532
TOPS/W	4A4W: 0.05	4A4W: 0.0911 2A4W: 0.1823

FPGA synthesis results demonstrate that the proposed multi-core wrapper provides a clear performance and efficiency advantage over the vanilla single-core design. The multi-core implementation leads to a modest increase in hardware resources, with LUT usage rising from 16,865 to 18,831 and register count from 12,224 to 13,416. The operating frequency drops slightly from 109.96 MHz to 103.9 MHz. By enabling parallel execution across two 8×8 MAC arrays, the multi-core design doubles the number of operations per cycle from 128 to 256, resulting in a 1.9× throughput improvement in 4A4W mode (0.0266 TOPS vs. 0.0141 TOPS) and achieving up to 0.0532 TOPS in 2A4W mode. Power consumption increases marginally from 282.14 mW to 291.9 mW, which is small relative to the performance gains, leading to a substantial improvement in energy efficiency, with TOPS/W increasing from approximately 0.05 to 0.0911 in 4A4W mode and reaching 0.1823 in 2A4W mode. These results confirm that the multi-core wrapper effectively trades a small increase in area and power for significantly higher throughput and energy efficiency, making it a promising architecture.

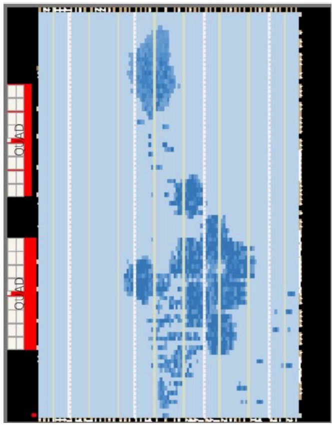

Fig. 6 FPGA Mapping of Vanilla Version

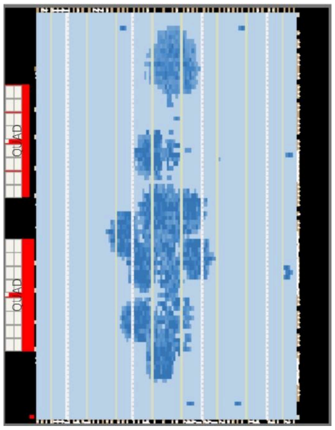

Fig. 7 FPGA Mapping of Multi-core Version