Veera Adithya Dittakavi
0034728255

Assignment 4

ECE 60827, Spring 2023
May 5, 2023

Alex Net

What does it implement? A deep convolutional neural network (CNN) for image classification on the ImageNet dataset, achieving state-of-the-art performance at the time of publication. The network architecture, called AlexNet, features multiple convolutional and fully connected layers to let use techniques such as data augmentation and dropout regularization to improve generalization for classification tasks.

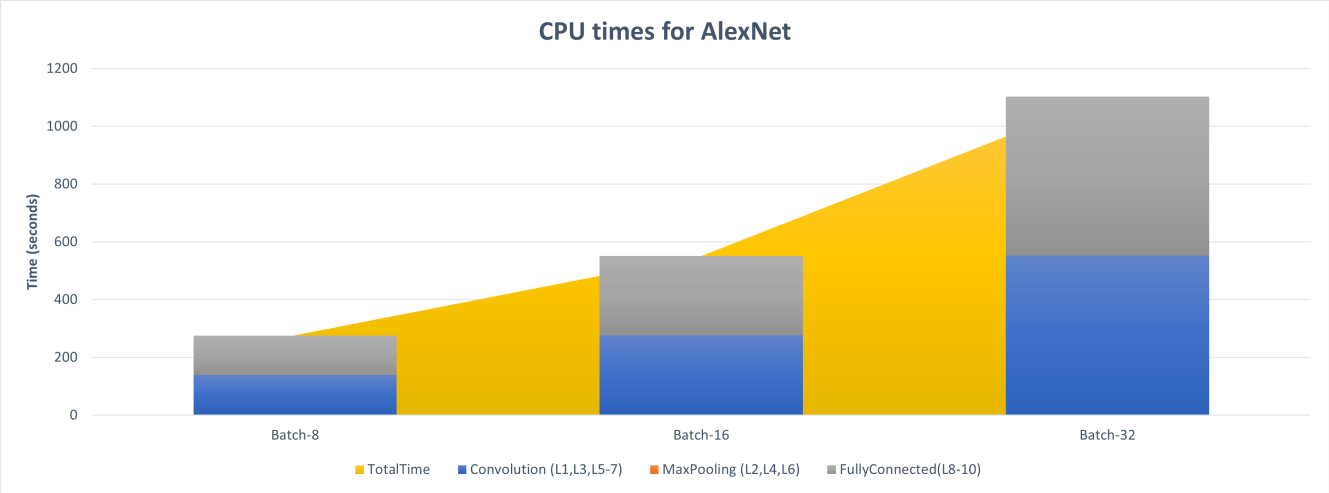

Figure 1: Plot for various batch sizes on CPU

Implementation artifacts

1. Convolutional data comprises of input, filter and bias. So each component is analysed to optimize computation. Convolutional layers are optimized by computing each pixel corresponding to output image in parallel, in a given thread block.

2. Before computation, either of the input data (representing image) or filter data (representing weights) is tiled and stored in respective dynamically allocated shared memory of thread blocks. Storing input rather than filter proved to be feasible in delivering neat code.

3. The bias is never changed by a kernel in forward propagation (inference). So it is stored in constant memory. This is theoretically efficient because each thread of warp during convolution accesses same address of constant memory.

4. Pooling is parallelized for a given tile width and the data is stored in static shared memory. This approach is used because that we know pooling does not require much dynamic changes in its arguments.

5. Fully connected components of the model are most memory consuming and have to perform GEMM (General Matrix Multiply). For this reason, the data is stored in relatively large

shared memory blocks than of Convolutional layers or Pooling. Else the thread blocks would grow exponentially which creates sequential computing.

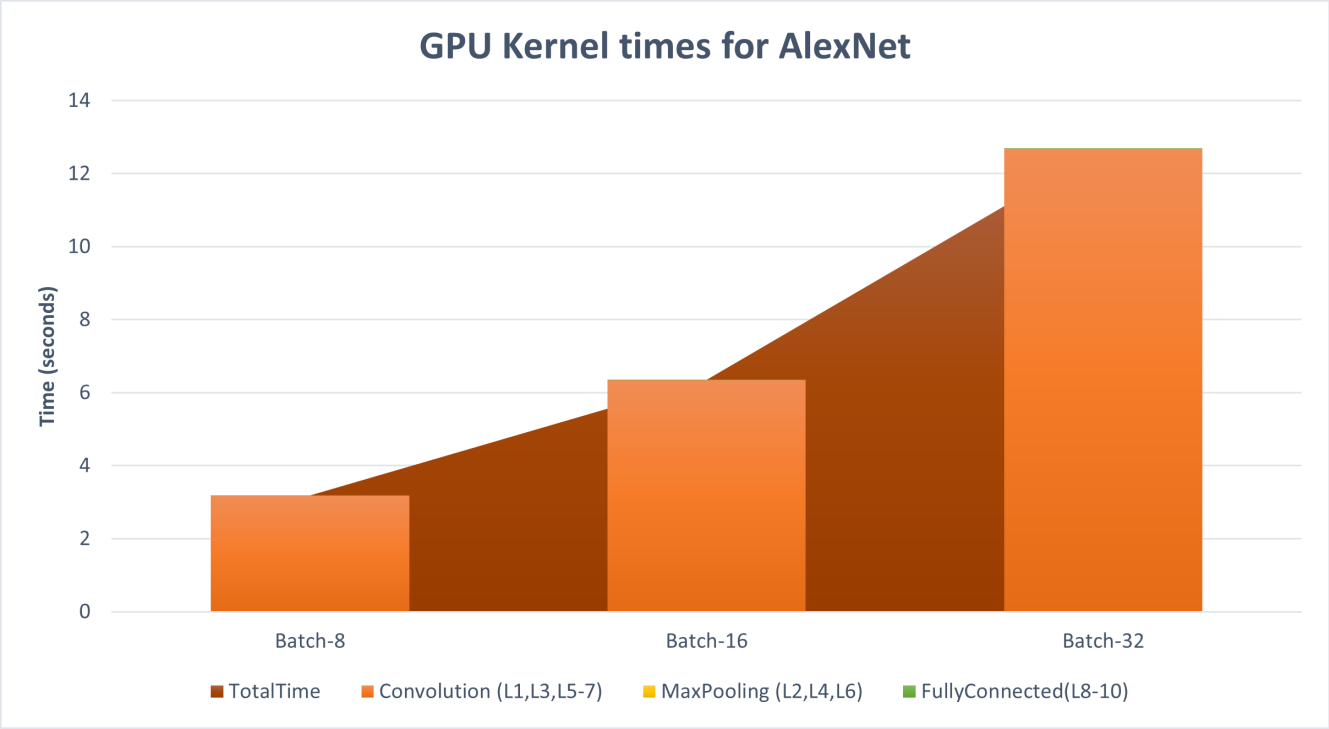

Figure 2: Plot for various batch sizes on GPU

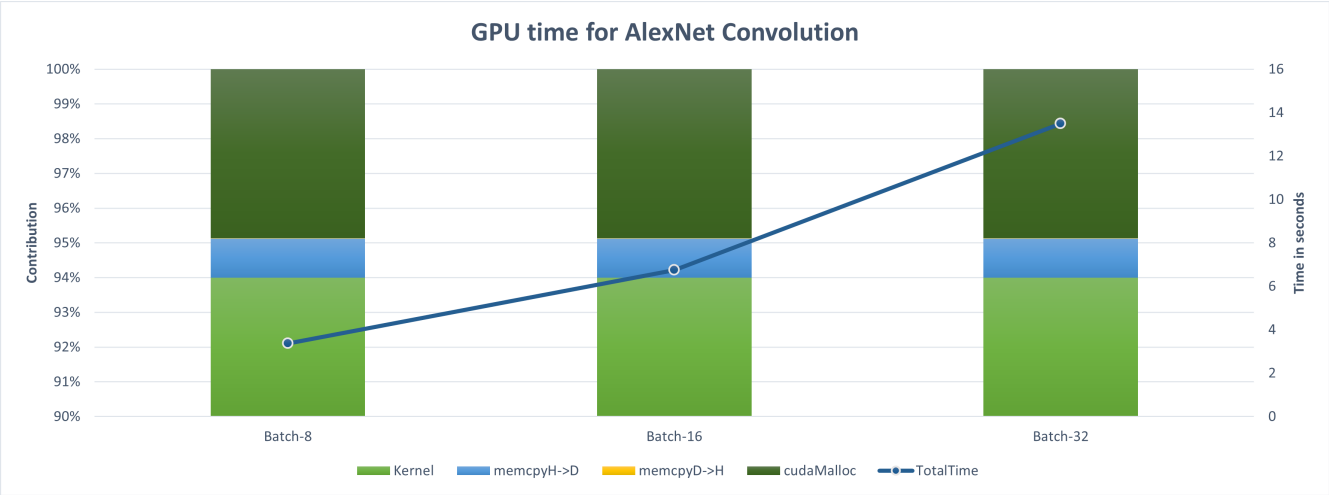

Figure 3: Plot for various batch sizes for Convolution kernel (Complex than others)

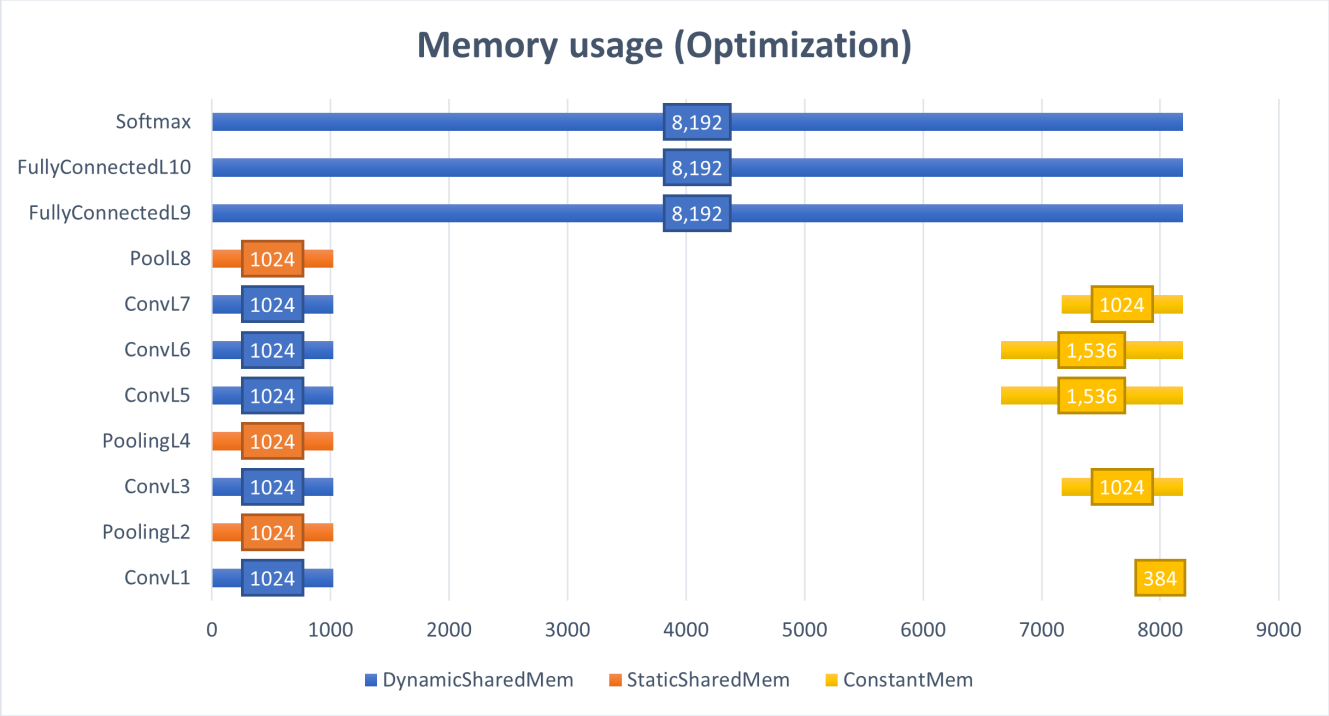

Figure 4: Plot for memory usage in Bytes (B) for each component of the DNN

Discussion

1. Fig. 1 depicts the inference on a CPU (where there is no optimization). We can see for 32 images, each of size 227x227 takes more than 15 min, which is not at all acceptable. Imagine training data over here, sound ridiculous right!.

2. So we use GPU for doing the same task of inference and we can see an improvement of 90x. Refer to Fig. 2 that provides analysis on the same. We can see how computation times increase for increasing batch size.

3. In the same figure, we can also see how convolution takes much more time than Pooling and GEMM which can be lightly seen. This is because the convolution is based on Floating point MUL, ADD on data from different locations. This rather creates a bottleneck at cache, thus requiring more optimization.

4. Fig. 3 considers analysis further only on the Convolutional layers of the AlexNet. The layers times are aggregated and divided into contributions of each Kernel, Memory copying from Device to Host, vice versa and Memory allocation. Memory allocation takes the longest time for each input, filter and bias because of large data. Then the kernel contributes to running time added its complexity in performing convolution operation. The higher time might be because of the sequential loop over the batch of images. Additionally, when we see Memory copy from Host to Device, it is much larger than Device to Host because we take an image, weight matrix and bias to output a batch of images which are considerably of low dimension.

5. Total time on the GPU for Convolution increases linearly with increasing batch size.

6. Finally coming to the memory usage by each specific kernel, referring to Fig. 4, we can see the division of different memory components to each kernel. Convolution and Pooling use 1KB of shared memory for a thread block. This might be a small number given total available shared memory. But large shared memory pools greatly affect the concurrency over a kernel by limiting allocation. Constant memory depends on the number of filters required for each of the convolution layers and we can see the allocation difference.

7. Fully connected layers on the other hand are data hungry, but the operation over memory is quite simple. So it's been optimized by giving a huge chunk of shared memory to perform quick accesses rather than waiting for increased time.

Continued optimization possibility

1. One drawback over the application here is that each and every kernel runs on a single stream: 7. So, every kernel call and memory copying is synchronized which increases computation time. One idea to eliminate this problem is to create multiple streams for sectors of a batch. Let us say, we have 32 images. We can divided the batch into sub-batches of 8 images each and perform inference on parallel streams. This approach greatly reduces linear dependency of convolutional running time over increasing batch size.

2. Another approach is to create patches from the given image. Patching is to divide the image into sequential image with a key. With the key, we can order the data we are performing computation on. With this division, we can perform convolution for each patch over different streams to reduce running time. The idea of creating patch comes from the concept of Vision Transformer (ViT).