Improving Performance of AlexNet Framework in CUDA

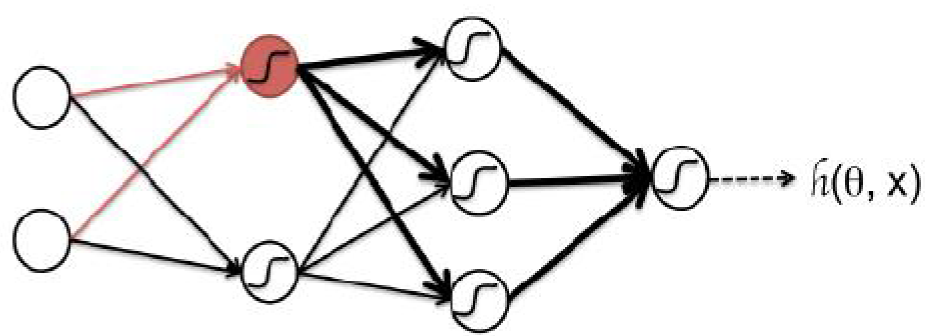
Project Group ID: 15
Himanshu Gunjal
Neel Patel

Project Topic

- We improved the performance of a CNN framework – AlexNet by using various optimization techniques learnt in class such as
- Shared Memory
- Constant Memory
- Loop Unrolling
- Streams
- Recorded the performance of all Layers using Cuda Event Timer

Deep Neural Network

- Extract complex features using more than one layer.
- Unlike Shallow Networks.
- Different layers run complex non-linear decisions to detect different features.

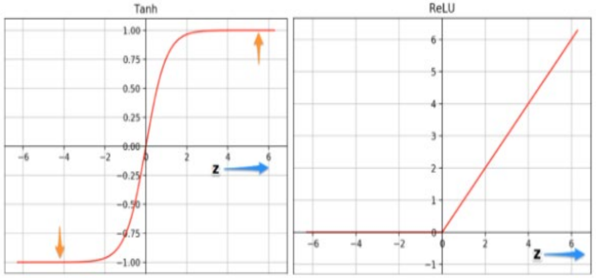


Convolutional Neural Network

- ❖ Framework of:
  - ❖ Multiple Convolution Layers
  - ❖ Multiple Pooling Layers
  - ❖ One/multiple Fully Connected Layers
  - ❖ Each neuron in one layer is connected to all neurons in the next layer.

Convolution:

- Input is a tensor with
  - Number of inputs(images)
  - Image Width
  - Height
  - Depth
- Kernel with Width and Height
  - Depth of kernel = Depth of Image
- Reduces the number of free parameters, allowing the network to be deeper with fewer parameters



- Our scenario:
  - Input = 227*227*3
  - Kernel = 11*11*3 (same depth of 3)
  - ReLu Layer (Rectified Linear Unit)
    - All negative values are brought down to 0

Pooling:

- Reduce the dimensions of the data by combining the outputs of neuron clusters at one layer into a single neuron in the next layer.
- Max pooling uses the maximum value from each of a cluster of neurons at the prior layer
- Average pooling uses the maximum value from each of a cluster of neurons at the prior layer

Fully Connected Layer:

- Connect every neuron in one layer to every neuron in another layer.
- The flattened matrix goes through a fully connected layer to classify the images.

- Receptive Fields:
  - Input area of a neuron is called its receptive field
  - (in a fully connected layer, the receptive field is the entire previous layer. In a convolutional layer, the receptive area is smaller than the entire previous layer.))))))))))))))))))))
- Weights:
  - Each neuron in a neural network computes an output value by applying some function to the input values coming from the receptive field in the previous layer
  - Biases add non-linearity to the computation

AlexNet:

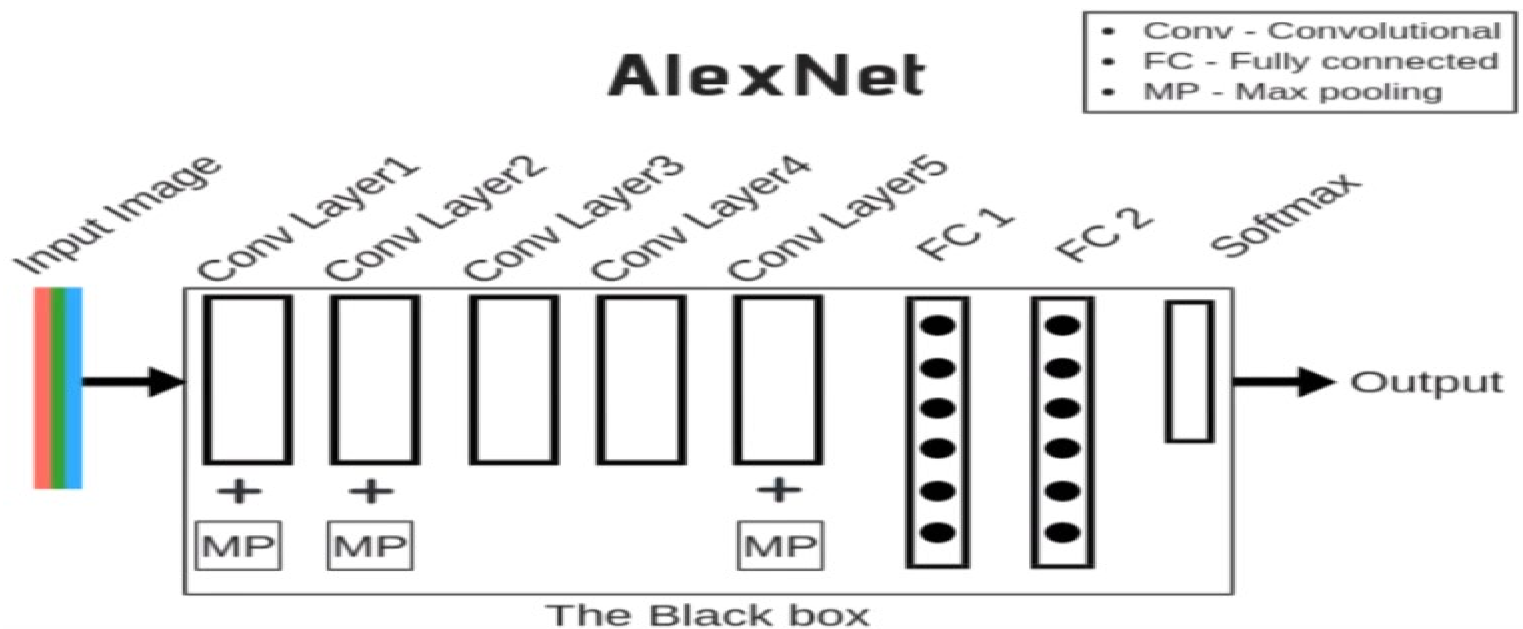
- AlexNet famously won the 2012 ImageNet LSVRC-2012 competition by a large margin.
- The network has:
  - 62.3 million parameters
  - 1.1 billion computations (in a forward pass)
  - Conv layers –
    - 6% of all parameters
    - 95% of total computation

AlexNet:

Size / Operation	Filter	Depth	Stride	Padding	Number of Parameters	Forward Computation
227 * 227 * 3						
Conv1 + Relu	11 * 11	96	4		(11*11*3 + 1) * 96=34944	(11*11*3 + 1) * 96 * 55 * 55=105705600
96 * 55 * 55						
Max Pooling	3 * 3		2			
96 * 27 * 27						
Norm						
Conv2 + Relu	5 * 5	256	1	2	(5 * 5 * 96 + 1) * 256=614656	(5 * 5 * 96 + 1) * 256 * 27 * 27=448084224
256 * 27 * 27						
Max Pooling	3 * 3		2			
256 * 13 * 13						
Norm						
Conv3 + Relu	3 * 3	384	1	1	(3 * 3 * 256 + 1) * 384=885120	(3 * 3 * 256 + 1) * 384 * 13 * 13=149585280
384 * 13 * 13						
Conv4 + Relu	3 * 3	384	1	1	(3 * 3 * 384 + 1) * 384=1327488	(3 * 3 * 384 + 1) * 384 * 13 * 13=224345472
384 * 13 * 13						
Conv5 + Relu	3 * 3	256	1	1	(3 * 3 * 384 + 1) * 256=884992	(3 * 3 * 384 + 1) * 256 * 13 * 13=149563648
256 * 13 * 13						
Max Pooling	3 * 3		2			
256 * 6 * 6						
Dropout (rate 0.5)						
FC6 + Relu					256 * 6 * 6 * 4096=37748736	256 * 6 * 6 * 4096=37748736
4096						
Dropout (rate 0.5)						
FC7 + Relu					4096 * 4096=16777216	4096 * 4096=16777216
4096						
FC8 + Relu					4096 * 1000=4096000	4096 * 1000=4096000
1000 classes						
Overall					62369152=62.3 million	1135906176=1.1 billion
Conv VS FC					Conv:3.7million (6%) , FC: 58.6	Conv: 1.08 billion (95%) , FC: 58.6 million (5%)

Approach on a GPU:

- Copy convolution layers into different GPUs
- Distribute the fully connected layers into different GPUs
- Feed one batch of training data into convolutional layers for every GPU (Data Parallel).
- Feed the results of convolutional layers into the distributed fully connected layers batch by batch



SJSU SAN JOSÉ STATE UNIVERSITY

Implementation/Solution

- Constant Memory: Bias
- Layer 6,7,8: Host Code:

```
/* Memcpy of weights and bias */
cudaMemcpy(Layer6_Weights_GPU,Layer6_Weights_CPU, sizeof(float)*4096*256*6*6, cudaMemcpyHostToDevice);
cudaMemcpyToSymbol(bias_L6_CM,bias_6, sizeof(float)*4096);////making this as constant memory. overflow only 10KB allowed
////cudaMemcpyToSymbol(Layer_InNeurons_GPU_CM,Layer6_Neurons_GPU,sizeof(float)*256*6*6);////making this as constant memory - didn't work >10KB

dim3 Layer6_Block(4096,1,1);
dim3 Layer6_Thread(1,1);   // combi tried 10*10*10

cudaEventRecord(start);;
executeFCLayer_L6<<<Layer6_Block,Layer6_Thread>>>(Layer6_Neurons_GPU,Layer6_Weights_GPU,Layer7_Neurons_GPU,4096,(256*6*6),true,false);
```

- Kernel Code:

```
__constant__ float bias_L6_CM[4096];
__global__ void executeFCLayer_L6(float *Layer_InNeurons_GPU, float *Layer_Weights_GPU,
            float *Layer_OutNeurons_GPU,int output, int input,bool reLU,bool dropout)
{
    float product = 0.0;
    int out = blockIdx.x;
    int weight = out * input;
    {
        for(int in = 0; in < input; in++)
        {
               product += Layer_InNeurons_GPU[in] * Layer_Weights_GPU[weight+in];
        }
        product += bias_L6_CM[out];////same for entire block so put in constant memory
        if(reLU == true)
        {
            if(product < 0) /* ReLU Layer */
                product = 0;
        }

        Layer_OutNeurons_GPU[out] = product;
        product = 0.0;
    }
```

Implementation/Solution

- Constant Memory: Bias and Neurons
- Layer 6,7,8
- Kernel Code:

```
__constant__ float bias_L7_CM[4096];
__constant__ float Layer_InNeurons_GPU_L7_CM[4096];
__global__ void executeFCLayer_L7(float *Layer_Weights_GPU,float *Layer_OutNeurons_GPU,
                                  int output, int input,bool reLU,bool dropout)
{
    float product = 0.0;
    int out = blockIdx.x;
    int weight = out * input;
    {
    //#pragma unroll ////caused increase in execution time
    for(int in = 0; in < input; in++)
        {
                product += Layer_InNeurons_GPU_L7_CM[in] * Layer_Weights_GPU[weight+in];
        }
        product += bias_L7_CM[out];////same for entire block so put in constant memory
        if(reLU == true)
        {
            if(product < 0) /* ReLU Layer */
                product = 0;
        }
```

Implementation/Solution

- Constant Memory: Bias and Neurons
- Layer 6,7,8
- Kernel Code:

```
__constant__ float bias_L8_CM[1000];
//__constant__ float Layer_InNeurons_GPU_L8_CM[4096]; // Constant memory overflow
__global__ void executeFCLayer_L8(float* Layer_InNeurons_GPU_L8,float *Layer_Weights_GPU,
                    float *Layer_OutNeurons_GPU,int output, int input,bool reLU,bool dropout)
{

    float product = 0.0;
    int out = blockIdx.x;
    int weight = out * input;
    {
        for(int in = 0; in < input; in++)
        {
            product += Layer_InNeurons_GPU_L8[in] * Layer_Weights_GPU[weight+in];
        }
        product += bias_L8_CM[out];////same for entire block so put in constant memory
        if(reLU == true)
        {
```

Implementation/Solution

- Layer 1-5: Kernel Code:
- Number of iterations varied upon block dimension
- Optimization using #pragma unroll:

```
#pragma unroll
    for(int feature =0; feature < in_output ; feature++) // calculate the feature maps
    {
        for(int i =0; i < loopr ; i++) // kernel convolution
        {
            for(int j =0; j < loopc ; j++) // kernel convolution
            {
                product += ( Layer2_Neurons_GPU[feature*fr*fc + i*fc + j + stride + colstride]
                    * Layer2_Weights_GPU[output*kernel*kernel*in_output + feature*kernel*kernel + i*kernel + j + kernel*x_pad + y_pad]);
            }
        }
    }
    product += bias[output];
    if(product < 0) /* ReLU Layer */
        product = 0;
    Layer3_Neurons_GPU[output*fr*fc + row*fc + col] = product;
    product = 0.0;
    if(col >= pad)
        stride+=stride_width;
```

Implementation/Solution

- Layer 1: Host Code:

```
executeFirstLayer<<<Layer1_Block,Layer1_Thread>>>(Layer1_bias_GPU,Layer1_Neurons_GPU,Layer1_Weights_GPU,Layer1_Norm_GPU,0,0);
dim3 Layer11_Block(96,1,1);
dim3 Layer11_Thread(32,23);
executeFirstLayer<<<Layer11_Block,Layer11_Thread>>>(Layer1_bias_GPU,Layer1_Neurons_GPU,Layer1_Weights_GPU,Layer1_Norm_GPU,0,32);
dim3 Layer12_Block(96,1,1);
dim3 Layer12_Thread(23,32);
executeFirstLayer<<<Layer12_Block,Layer12_Thread>>>(Layer1_bias_GPU,Layer1_Neurons_GPU,Layer1_Weights_GPU,Layer1_Norm_GPU,32,0);
dim3 Layer13_Block(96,1,1);
dim3 Layer13_Thread(23,23);
executeFirstLayer<<<Layer13_Block,Layer13_Thread>>>(Layer1_bias_GPU,Layer1_Neurons_GPU,Layer1_Weights_GPU,Layer1_Norm_GPU,32,32);
```

- Optimization using Streams:

```
cudaMemcpyAsync(Layer1_Weights_GPU_0,Layer1_Weights_CPU+i, (sizeof(float)*L1_KERNEL_SIZE * L1_OUT)/4, cudaMemcpyHostToDevice,stream0);
cudaMemcpyAsync(Layer1_Neurons_GPU_0,Layer1_Neurons_CPU+i, (sizeof(float)*INPUT_SIZE)/4, cudaMemcpyHostToDevice,stream0);
cudaMemcpyAsync(Layer1_bias_GPU_0,bias_1+i, (sizeof(float)* L1_OUT)/4, cudaMemcpyHostToDevice,stream0);
executeFirstLayer<<<Layer1_Block,Layer1_Thread,0,stream0>>>(Layer1_bias_GPU_0,Layer1_Neurons_GPU_0,Layer1_Weights_GPU_0,Layer1_Norm_GPU,0,0);


cudaMemcpyAsync(Layer1_Weights_GPU_1,Layer1_Weights_CPU+i+segment_size_L1, (sizeof(float)*L1_KERNEL_SIZE * L1_OUT)/4, cudaMemcpyHostToDevice,stream1);
cudaMemcpyAsync(Layer1_Neurons_GPU_1,Layer1_Neurons_CPU+i+segment_size_L1, (sizeof(float)*INPUT_SIZE)/4, cudaMemcpyHostToDevice,stream1);
cudaMemcpyAsync(Layer1_bias_GPU_1,bias_1+i+segment_size_L1, (sizeof(float)* L1_OUT)/4, cudaMemcpyHostToDevice,stream1);
executeFirstLayer<<<Layer11_Block,Layer11_Thread,0,stream1>>>(Layer1_bias_GPU_1,Layer1_Neurons_GPU_1,Layer1_Weights_GPU_1,Layer1_Norm_GPU,0,32);

cudaMemcpyAsync(Layer1_Weights_GPU_2,Layer1_Weights_CPU+i+(segment_size_L1*2), (sizeof(float)*L1_KERNEL_SIZE * L1_OUT)/4, cudaMemcpyHostToDevice,stream2);
cudaMemcpyAsync(Layer1_Neurons_GPU_2,Layer1_Neurons_CPU+i+(segment_size_L1*2), (sizeof(float)*INPUT_SIZE)/4, cudaMemcpyHostToDevice,stream2);
cudaMemcpyAsync(Layer1_bias_GPU_2,bias_1+i+(segment_size_L1*2), (sizeof(float)* L1_OUT)/4, cudaMemcpyHostToDevice,stream2);
executeFirstLayer<<<Layer12_Block,Layer12_Thread,0,stream2>>>(Layer1_bias_GPU_2,Layer1_Neurons_GPU_2,Layer1_Weights_GPU_2,Layer1_Norm_GPU,32,0);

cudaMemcpyAsync(Layer1_Weights_GPU_3,Layer1_Weights_CPU+i+(segment_size_L1*3), (sizeof(float)*L1_KERNEL_SIZE * L1_OUT)/4, cudaMemcpyHostToDevice,stream3);
cudaMemcpyAsync(Layer1_Neurons_GPU_3,Layer1_Neurons_CPU+i+(segment_size_L1*3), (sizeof(float)*INPUT_SIZE)/4, cudaMemcpyHostToDevice,stream3);
cudaMemcpyAsync(Layer1_bias_GPU_3,bias_1+i+(segment_size_L1*3), (sizeof(float)* L1_OUT)/4, cudaMemcpyHostToDevice,stream3);
executeFirstLayer<<<Layer13_Block,Layer13_Thread,0,stream3>>>(Layer1_bias_GPU_3,Layer1_Neurons_GPU_3,Layer1_Weights_GPU_3,Layer1_Norm_GPU,32,32);
```

Evaluation: Timing Comparison

```
nvidia@tegra-ubuntu:~/AlexNet$ sh build.sh
nvidia@tegra-ubuntu:~/AlexNet$ ./AN 1
 File FOUND
READ INPUT Final Count :: 154587
 File FOUND data/bias1.txt
Final Count : 96
 File FOUND data/bias2.txt
Final Count : 256
 File FOUND data/bias3.txt
Final Count : 384
 File FOUND data/bias4.txt
Final Count : 384
 File FOUND data/bias5.txt
Final Count : 256
 File FOUND data/bias6.txt
Final Count : 4096
 File FOUND data/bias7.txt
Final Count : 4096
 File FOUND data/bias8.txt
Final Count : 1000
 File FOUND data/conv1.txt
Final Count : 34848
 File FOUND data/conv2.txt
Final Count : 307200
 File FOUND data/conv3.txt
Final Count : 884736
 File FOUND data/conv4.txt
Final Count : 663552
 File FOUND data/conv5.txt
Final Count : 442368
 File FOUND data/fc6.txt
Final Count : 37748736
 File FOUND data/fc7.txt
Final Count : 16777216
 File FOUND data/fc8.txt
Final Count : 4096000
Extracted Weights and Bias successfully

 Elapsed time for layer 1 is 162.32 ms
 Elapsed time for layer 2 is 115.44 ms
 Elapsed time for layer 3 is 74.32 ms
 Elapsed time for layer 4 is 60.29 ms
 Elapsed time for layer 5 is 34.67 ms
 Elapsed time for layer 6 is 96.28 ms
 Elapsed time for layer 7 is 41.13 ms
 Elapsed time for layer 8 is 10.28 ms
 INDEX = 285
nvidia@tegra-ubuntu:~/AlexNet$
```

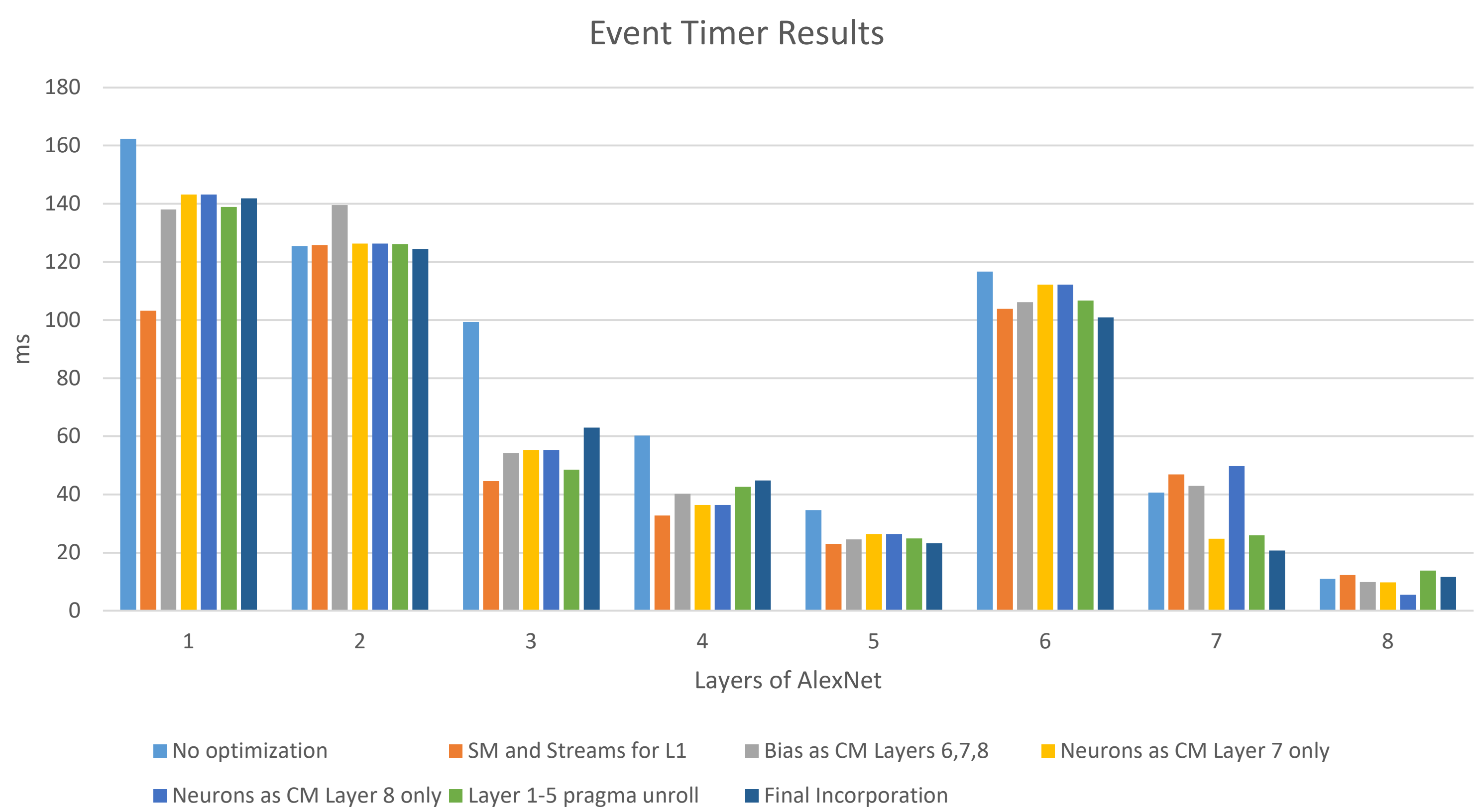
```
nvidia@tegra-ubuntu:~/AlexNet$ sh build.sh
nvidia@tegra-ubuntu:~/AlexNet$ ./AN 1
 File FOUND
READ INPUT Final Count :: 154587
 File FOUND data/bias1.txt
Final Count : 96
 File FOUND data/bias2.txt
Final Count : 256
 File FOUND data/bias3.txt
Final Count : 384
 File FOUND data/bias4.txt
Final Count : 384
 File FOUND data/bias5.txt
Final Count : 256
 File FOUND data/bias6.txt
Final Count : 4096
 File FOUND data/bias7.txt
Final Count : 4096
 File FOUND data/bias8.txt
Final Count : 1000
 File FOUND data/conv1.txt
Final Count : 34848
 File FOUND data/conv2.txt
Final Count : 307200
 File FOUND data/conv3.txt
Final Count : 884736
 File FOUND data/conv4.txt
Final Count : 663552
 File FOUND data/conv5.txt
Final Count : 442368
 File FOUND data/fc6.txt
Final Count : 37748736
 File FOUND data/fc7.txt
Final Count : 16777216
 File FOUND data/fc8.txt
Final Count : 4096000
Extracted Weights and Bias successfully

 Elapsed time for layer 1 is 147.77 ms
 Elapsed time for layer 2 is 123.52 ms
 Elapsed time for layer 3 is 44.63 ms
 Elapsed time for layer 4 is 32.74 ms
 Elapsed time for layer 5 is 23.02 ms
 Elapsed time for layer 6 is 101.84 ms
 Elapsed time for layer 7 is 24.78 ms
 Elapsed time for layer 8 is 13.16 ms
INDEX = 285
nvidia@tegra-ubuntu:~/AlexNet$
```

Evaluation: Timing Comparison

Layer	No optimization	SM and Streams for L1	Bias as CM Layers 6,7,8	Neurons as CM Layer 7 only	Neurons as CM Layer 8 only	Layer 1-5 pragma unroll	Final Incorporation
1	162.32	103.21	138.01	143.17	143.17	138.93	141.83
2	125.43	125.79	139.57	126.32	126.32	126.1	124.4
3	99.34	44.64	54.19	55.3	55.3	48.52	62.96
4	60.29	32.72	40.23	36.42	36.42	42.66	44.79
5	34.67	23.03	24.58	26.37	26.37	24.9	23.26
6	116.71	103.87	106.18	112.18	112.18	106.75	100.85
7	40.63	46.93	43	24.77	49.8	25.99	20.72
8	10.93	12.27	9.89	9.79	5.5	13.87	11.62
Total	650.32	492.46	555.65	534.32	555.06	527.72	530.43
Index	285	188	285	285	285	285	285

Evaluation: Statistical Comparison



Demonstration

Your project file structure

- Explain how your project file is structured by indicating the important files to check (You don’t need to present this page; we will use this for verification)
  - 1. Base Code
    - Make file, alexnet_host.cu, an_kernel.cu, Data
  - 2. Optimized Code
    - Make file, alexnet_host.cu, an_kernel.cu, Data
  - 3. Results
    - Project_PPT, Project_Excel

Execution guidance

- Provide the list of commands that need to be executed to get the output that you showed in the demonstration (Again, you don't need to present this page)
  1. Decompress the folder
  2. Go to the base code folder.
     This contains the unoptimized code which we started with. We just added timing for each layer in this file. Rest is the same.
  3. Open terminal here and type the following commands.
  4. Sh build.sh
  5. ./AN 1
  6. Check the timings on the command line.

Execution guidance

7. Go back and open Optimized folder.
8. Open terminal here and type the following commands.
9. Sh build.sh
10. ./AN 1
11. Check the improved timings on the command line
12. Go back and open Results folder
13. This folder contains the rest of the files of our project like the PPT, Excel data of results etc.

References

- http://developer.download.nvidia.com/GTC/PDF/1083_Wang.pdf
- http://vision.stanford.edu/teaching/cs231b_spring1415/slides/alexnet_tugce_kyunghee.pdf
- https://en.wikipedia.org/wiki/AlexNet

- https://www.mathworks.com/help/images/image-processing-on-a-gpu.html
- https://medium.com/@smallfishbigsea/a-walk-through-of-alexnet-6cbd137a5637
- https://www.youtube.com/watch?v=uLddd86qVFs
- https://www.learnopencv.com/understanding-alexnet/

<u>Thank You</u>