**Assignment 3**
**GPU Architecture**
**Spring 2018**

**Ankit Sinha**
**GTID: 903116700**

# *Project Report*

For the purpose of the project, I am using three different kernels. They have been optimized in different ways to achieve the best output. The outputs shown are were obtained on the pace cluster using the teslap100 (Pascal) node.

1. **Convolution**:
   The convolution kernel is the first kernel which gets called. All the 8 convolutions are performed in this kernel using 8 different filters.
   I call this kernel with 8 blocks containing 28*28 threads each. All of these blocks load the 28*28 image into shared memory before beginning the computation. This helps because each element of the image is used multiple times during convolution.
   Above this, all the filter images were stored in constant memory.
   After convolution, this kernel also performs the biasing and RELU for all the 6272 output elements.
   The reshaping is also done in this kernel as writing back to the global memory is done keeping reshaping in mind. This decreases the overhead of another kernel call for reshaping.

   If we use separate kernels for each convolution, each kernel call takes around 12µs without using shared memory and one has to use streams to optimize this. But if we use 8 blocks, we get the same effect.

   With a simple implementation without any shared and constant memory, we get a timing of around 31µs.
   Without shared memory and using only constant memory where one thread does 8 calculations, the timing was around 15.968µs.

```
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:   85.06%  1.4620ms         8  182.75us     960ns  1.4515ms  [CUDA memcpy HtoD]
                   13.31%  228.77us         1  228.77us  228.77us  228.77us  Multiply0(float*, float*, int, float*, float*)
                    0.93%  15.968us         1  15.968us  15.968us  15.968us  convolution(float*, float*, float*)
                    0.41%  7.0080us         1  7.0080us  7.0080us  7.0080us  Multiply1(float*, float*, float*, float*)
                    0.22%  3.8080us         3  1.2690us     928ns  1.8560us  [CUDA memset]
                    0.08%  1.3120us         1  1.3120us  1.3120us  1.3120us  [CUDA memcpy DtoH]
```

   With shared memory and constant memory where each thread does one operation, we see the timing as 10.656µs.

```
==12857== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:   85.26%  1.4496ms         8  181.21us     959ns  1.4395ms  [CUDA memcpy HtoD]
                   13.43%  228.29us         1  228.29us  228.29us  228.29us  Multiply0(float*, float*, int, float*, float*)
                    0.63%  10.656us         1  10.656us  10.656us  10.656us  convolution(float*, float*)
                    0.39%  6.6560us         1  6.6560us  6.6560us  6.6560us  Multiply1(float*, float*, float*, float*)
                    0.23%  3.8400us         3  1.2800us     960ns  1.8560us  [CUDA memset]
                    0.08%  1.2800us         1  1.2800us  1.2800us  1.2800us  [CUDA memcpy DtoH]
```

   So, we see an improvement of around 48.49% by using the shared memory and constant technique.

2. **Multiplication0**:
   This kernel does the multiplication for the [1*6272] elements of the output of convolution kernel with the matrix fc1[6272*512].
   The multiplication is done using the tiling approach employing the shared memory. The tile size that is chose is 128. Elements from the reshaped array of 6272 elements are loaded into shared memory.

This is important since the 6272 elements of the reshaped array are used 512 times each for multiplication.
I perform tiling with a tile width of 128 elements.
Without using this optimization, the linear multiplication using 6272 threads took around 1.55ms. With shared memory approach, the timing has reduced to around 228µs.
That's an impressive reduction of around 85.29%.
It works well because the overall global memory accesses are reduced by 512 times.

Apart from the multiplication, this kernel also performs the biasing and RELU operations for the 512 elements that go on to the next stage.

```
==12857== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:   85.26%  1.4496ms         8  181.21us     959ns  1.4395ms  [CUDA memcpy HtoD]
                   13.43%  228.29us         1  228.29us  228.29us  228.29us  Multiply0(float*, float*, int, float*, float*)
                    0.63%  10.656us         1  10.656us  10.656us  10.656us  convolution(float*, float*)
                    0.39%  6.6560us         1  6.6560us  6.6560us  6.6560us  Multiply1(float*, float*, float*, float*)
                    0.23%  3.8400us         3  1.2800us     960ns  1.8560us  [CUDA memset]
                    0.08%  1.2800us         1  1.2800us  1.2800us  1.2800us  [CUDA memcpy DtoH]
```

3. **Mutiplication1**:
   The second multiplication is performed in this kernel.
   For this purpose, I use 10 blocks of 512 threads each. Each block performs multiplication on a column of multiplicand matrix. Then 10 threads proceed to compute the addition of the 10 columns in the sequential manner.
   Hence this way each thread performs one multiplication and simultaneously loads the result into the shared memory. After this, 10 threads proceed and do the sum and biasing.
   If we just use shared memory for this step employing tiling, we get a result of around 46µs.

```
==16070== Profiling application: ./mnist ./images/xaa ./labels/xaa
==16070== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:   83.45%  1.4595ms         8  182.44us     896ns  1.4498ms  [CUDA memcpy HtoD]
                   13.05%  228.22us         1  228.22us  228.22us  228.22us  Multiply0(float*, float*, int, float*, float*)
                    2.63%  46.048us         1  46.048us  46.048us  46.048us  Multiply1(float*, float*, int, int, float*, float*)
                    0.60%  10.432us         1  10.432us  10.432us  10.432us  convolution(float*, float*)
                    0.20%  3.5840us         3  1.1940us     864ns  1.7920us  [CUDA memset]
                    0.06%  1.0880us         1  1.0880us  1.0880us  1.0880us  [CUDA memcpy DtoH]
```

With the above approach, we get the result as 6.53µs.

```
==32098== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:   85.17%  1.4408ms         8  180.10us     960ns  1.4304ms  [CUDA memcpy HtoD]
                   13.50%  228.41us         1  228.41us  228.41us  228.41us  Multiply0(float*, float*, int, float*, float*)
                    0.63%  10.655us         1  10.655us  10.655us  10.655us  convolution(float*, float*)
                    0.39%  6.5280us         1  6.5280us  6.5280us  6.5280us  Multiply1(float*, float*, float*, float*)
                    0.23%  3.8720us         3  1.2900us     992ns  1.8560us  [CUDA memset]
```

Hence with this approach, we see a benefit of around 85.8%.

Summary:
Final timings for the three kernels after optimizations are:

| Kernel Name | Execution Time |
|---|---|
| Convolution | 10.655µs |
| Multiplication0 | 228.41µs |
| Multiplication1 | 6.528µs |