

## Project Report

teamname: q80

Handed In: December 19, 2017

## 1. Milestone 1

- (a) The time elapsed is 16.373s

```

Loading fashion-mnist data... done
Loading model... done
EvalMetric: {'accuracy': 0.8673}
* The build folder has been uploaded
t.com/userdata/build-02e09e7f-715c-48
be present for only a short duration
* Server has ended your request.

real    0m16.373s
user    0m0.386s
sys     0m0.123s

```

- (b) The time elapsed is 44.267s

```

done
EvalMetric: {'accuracy': 0.8673}
* The build folder has been uploaded
t.com/userdata/build-94bcc972-3cad-48
be present for only a short duration
* Server has ended your request.

real    0m44.267s
user    0m0.378s
sys     0m0.093s

```

- (c) The result for nvprof is shown below

```

nvprof --api-calls --api-calls:python /src/ml.2.py
--306-- Profiling result:
Time(s)  Time  Calls  Avg  Min  Max  Name
36.81s  58.450ms  1  58.450ms  58.450ms  void cudnn::detail::implicit_convolve_sgemm(float, int=1824, i
nt=5, int=5, int=3, int=3, int=1, bool=0, bool=0, bool=1, bool=0, float const *, int, int)
28.77s  828.61ms  24  34.525ms  264.71us  821.38ms  cudaMemGetInfo
3.74s  129.17ms  25  5.1666ms  5.315us  81.82ms  cudaStreamSynchronize
0.36s  14.33ms  8  1.7924ms  11.75us  6.9961ms  cudaMemcpy2DAsync
0.18s  7.017ms  42  167.07us  13.71us  1.273ms  cudaMalloc
0.04s  1.460ms  4  365.46us  38.86us  1.136ms  cudaStreamCreate
0.04s  1.397ms  4  349.29us  337.77us  382.57us  cudaDeviceTotalMem
0.03s  1.0637ms  114  9.3300us  644ns  425.02us  cudaEventCreateWithFlags
0.02s  898.95us  352  2.553us  24ns  79.94us  cudaDeviceGetAttribute
0.01s  574.88us  23  24.994us  13.29us  89.23us  cudaLaunch
0.01s  452.98us  6  75.496us  27.242us  128.10us  cudaMemcpy
0.00s  119.78us  4  29.948us  26.282us  36.21us  cudaDeviceGetMem
0.00s  101.67us  32  3.1770us  1.893us  11.087us  cudaDevice
0.00s  90.232us  110  820ns  432ns  2.738us  cudaDeviceGetAttribute
0.00s  62.844us  147  427ns  256ns  1.064us  cudaSetupArgument
0.00s  47.515us  2  23.757us  22.772us  25.243us  cudaStreamCreateWithPriority
0.00s  27.137us  23  1.179us  441ns  2.650us  cudaConfigureCall
0.00s  17.540us  10  1.7540us  857ns  2.245us  cudaDevice
0.00s  9.590us  16  597ns  44ns  74ns  cudaPeekAtLastError
0.00s  8.727us  1  8.727us  8.727us  8.727us  cudaBindTexture
0.00s  5.050us  1  5.050us  5.050us  5.050us  cudaStreamGetPriority
0.00s  4.980us  6  830ns  308ns  1.643us  cudaDeviceGetCount
0.00s  4.182us  6  697ns  532ns  1.248us  cudaDeviceGet
0.00s  4.077us  2  2.038us  1.283us  2.794us  cudaStreamWaitEvent
0.00s  3.992us  2  1.996us  1.848us  2.144us  cudaDeviceGetStreamPriorityRange
0.00s  3.641us  2  1.820us  1.214us  2.427us  cudaEventRecord
0.00s  3.230us  6  538ns  281ns  872ns  cudaDeviceGetError
0.00s  2.667us  3  889ns  799ns  95ns  cuInit
0.00s  2.117us  3  703ns  665ns  771ns  cudaIversonGetVersion
0.00s  1.590us  1  1.590us  1.590us  1.590us  cudaUnbindTexture
0.00s  1.090us  1  1.090us  1.090us  1.090us  cudaGetDeviceCount

```

We can see from the result that the most time consuming kernel in profile part in `implicit_convolve_sgemm`, `sgemm_sm35_ldg_tn_128x8x256x16x32` and `activation_fw_4d_kernel`. For the API call, the most consuming kernel is `cudaStreamCreateWithFlags`, `cudaFree` and `cudaMemGetInfo`.

## 2. Milestone 2

The result is shown below.

```
Running setup.py develop for mxnet
Successfully installed mxnet
* Running python /src/m2.1.py
Loading fashion-mnist data... done
Loading model... done
Op Time: 12.220183
Correctness: 0.8562 Model: ece408-high

Successfully installed mxnet
* Running python /src/m2.1.py ece408-low 100
Loading fashion-mnist data... done
Loading model... done
Op Time: 0.121339
Correctness: 0.63 Model: ece408-low
* The build folder has been uploaded to https://cs2
```

### 3. Milestone 3

The result is shown below.

```
Successfully installed mxnet
* Running nvprof python m3.1.py ece408-high 10000
New Inference
Loading fashion-mnist data... done
==309== NVPROF is profiling process 309, command: python m3.1.py ece408-high 10000
Loading model... done
Op Time: 0.428778
Correctness: 0.8562 Model: ece408-high
==309== Profiling application: python m3.1.py ece408-high 10000
==309== Profiling result:
Time(%) Time Calls Avg Min Max Name
83.54% 428.70ms 1 428.70ms 428.70ms 428.70ms void mxnet::op::forward_kernel<mshadow::gpu, float>(float*, mxnet::op::forward_kernel<mshadow::gpu, float> const *, mxnet::op::forward_kernel<mshadow::gpu, float> const, int, int, int, int)
7.68% 39.403ms 1 39.403ms 39.403ms 39.403ms sgemm_sm35_ldg_tn_128x8x256x16x32
```

In this example, the forward layer took 0.4288 seconds, and the forward\_kernel took 0.4287 seconds.

### 4. Final step

#### (a) Methods implemented for optimization

##### Step 1: Use shared memory to improve tile convolution

In milestone 3, we implemented a naive version of tile convolution but without using any shared memory. To improve the performance, we defined a dynamic shared memory to store input tile and kernel. The size of input tile is  $(H_{out} + K - 1) \times (W_{out} + K - 1) + K.SIZE \times K.SIZE$ . This will increase the performance a lot since we reduce the number of global memory access and the access to shared memory will be way faster. The Op time is about 140ns.

##### Step 2: Add unroll and mapping to previous step

We then consider all the optimization methods introduced in the course including using the shared memory to reduce the number of global read and write, use the constant memory to reduce the number of global read and write, avoid the control

divergence, unroll the matrix to do the convolution operation so that the memory coalescing can be achieved, try to use as many threads in a thread block as possible.

Our second approach simply consists of two steps. First unroll the input matrix  $X$ , then perform tiled matrix multiplication and generate the result to the output matrix. In this approach we first unroll the input matrix of size  $[batch, num\_filter, y, x]$  into  $[k*k, (y-k+1)*(x-k+1)*batch]$ . Then we do tiled matrix multiplication using the tile width of  $[25*32]$ . the reason of choosing this shared memory size is because the output of the unrolled matrix multiplication is  $[num\_kernel, (y-k+1)*(x-k+1)*batch]$ . The number of rows of the matrix and the number of columns of the matrix are the integer multiples of the shared memory dimensions. 32 is also the number of threads in a warp so the control divergence is minimized. Lastly we map the result matrix  $[num\_kernel, (y-k+1)*(x-k+1)*batch]$  back to the  $[batch, num\_filter, (y-k+1), (x-k+1)]$  and write the result to output pointer.

### (b) Result for optimization

```
Successfully installed mxnet
* Running /usr/bin/time -f "%User %Ssystem %elapsed" python /eval-scripts/final.py ece408-high
New Inference
Loading fashion-mnist data... done
Loading model... done
Op Time: 0.067217
Correctness: 0.8562 Model: ece408-high
1.84user 0.92system 2.33elapsed
* Running /usr/bin/time -f "%User %Ssystem %elapsed" python /eval-scripts/final.py ece408-low
New Inference
Loading fashion-mnist data... done
Loading model... done
Op Time: 0.060283
Correctness: 0.629 Model: ece408-low
1.49user 0.80system 1.88elapsed
```

### (c) Work distribution

Our team meet every week to optimize the system. We use the Codepad to share our code and did the coding part together.