ECE 408 Final Project Report

Guo-Cheng Lo

Milestone 1:

1.1 : The result accuracy of running m1.1.py is 0.8673 and it took 17.66s to run the program.

1.2: The result accuracy of running m1.2.py is 0.8673 and it took 44.72s to run the program which is longer than m1.1.py since it is using GPU instead of CPU.

1.3: Top three time consuming kernels are implicit_convolve_sgemm, sgemm_sm35_ldg_tn, and activation_fw_4d_kernel which has running time of 49.975ms, 38.845ms, 19.372ms respectively. The complete profiling is described in figure 1. These three kernels are related to the computation of convolution which include using SGEMM kernels as well as forward convolution kernel. That explains why most of the time are spent on these kernels.

Milestone 2:

2.1: The accuracy are correct for both case of ece408-low and ece408-high. The running time of convolution layer in the case of ece408-low is 17.444s. It also reaches the baseline accuracy of 0.8673. Therefore, it is conclusive that the implementation for CPU version is correct.

Milestone 3:

3.1: The accuracy are correct for both cases of ece408-low and ece408-high. The nvprof profile of ece408 is shown below as figure2. The time for running ece408-low model is 1.6min .For this milestone, no optimization method has been used to increase performance but the functionality is correct since it reaches the correct accuracy for both cases.

Final Optimizations

For optimizations, an obvious one would to be improve directly on the algorithm that is implemented in milestone 3. Assume the baseline performance is the same as the milestone 3 model since there are no optimizations used in milestone 3. The running time of milestone 3 is 15.3515s shown in the nvprof in figure 2. The first optimization that can be used is to use shared memory on the convolution kernel to prevent long global memory access time and also to increase the ratio between global memory read and global memory write. In milestone 3, the ratio between global memory read and global memory write is 2 to 1 which means every 1 output requires 2 input which is not that great. A way to improve that is to use shared memory. Shared memory allow same input to be reused so that the number of global memory read is reduced and instead, the input are read from shared memory for computation. With tile size as 24*24, the running time becomes 258.34ms This shows some improvement over the baseline but there is not really any further improvement to optimize from this certain algorithm.

Another way is to change the layout of the matrix so that now it becomes a simple matrix multiplication. In order to do that, the input matrix needs to be unrolled so that all the input that need to be multiplied by the weight kernel is in the same column. However,

instead of one convolution computation, now it becomes three computations where one matrix needs to be unrolled first, two matrices need to be multiplied, and then due to the memory layout, the output matrix from matrix multiplication needs to be reordered. The time of each computation, as shown in figure 3., it is clear that matrix multiplication takes a significant of time which means it is a good place to start optimizing. It took 226.74ms to run matrix multiplication kernel. The nvprof in figure 3 for matrix multiplication is a simple matrix multiplication kernel with no optimizations used. To optimize the matrix multiplication, an easy way is to use the same optimizations that was used on the convolution algorithm which is using shared memory to reuse inputs. After using shared memory, with tile size as 32*32, the running time becomes 200.268ms as shown in nvprof in figure 3.

The final optimizations used to reduce running time are to reduce number of threads per block while keeping the same amount of blocks. In another words, there will be less threads and the rest of threads will do more work. To clarify that, essentially this optimization is trying to make one thread doing multiple instruction in parallel. This can be achieved if these instruction are not depended on each other. For example, instead of having one thread reading from one element of shared memory or global memory, now we make one thread reading 2 element out of memory by performing memory read twice. Since these instruction has no dependency issue, they can be executed in parallel. One example of it is to use pragma unroll on for loops. In optimized matrix multiplication with shared memory, where the partial sum is calculated, the partial sum is accumulated based on different location in the shared memory. If we apply the reduction of threads idea on it, essentially we are cutting the number of iteration by a certain factor since multiple iterations are executed in parallel. Also, another important optimization is to use registers instead of shared memory. Since registers have are much faster to access than shared memory, there is no reason why not to exploit this advantages. After using all the final optimizations, the final running time becomes 118.445ms and it is shown in figure 4 with nvprof. A list of optimizations and results are also summarized in table 1.

In summary, many of the optimizations essentially are improving memory access time and creating more parallelism within instructions. If there is a long memory latency for accessing global memory, then change it to shared memory. If there is no dependency within a large iteration loop, then use thread reduction idea to increase parallelism. If one algorithm could not be improve for significant performance boost, then perhaps change the layout of input so that another algorithm could be applied.

References

Volkov, Vasily. (2015). Better Performance at Lower Occupancy. Proceedings of the GPU Technology Conference, GTC. 10. .

```
Correctness: 0.8562 Model: ece408-high
==312== Profiling application: python m3.1.py ece408-high 10000
==312== Profiling result:
Time(%)      Time     Calls       Avg       Min       Max  Name
99.32%  15.3515s         1  15.3515s  15.3515s  15.3515s  mxnet::op::forward_kernel(float*, float const *, float c
  0.25%  39.249ms         1  39.249ms  39.249ms  39.249ms  sgemm_sm35_ldg_tn_128x8x256x16x32
0.13%  19.584ms         1  19.584ms  19.584ms  19.584ms  void mshadow::cuda::MapPlanLargeKernel<mshadow::sv::savet
ow::expr::Plan<mshadow::expr::BinaryMapExp<mshadow::op::mul, mshadow::expr::ScalarExp<float>, mshadow::Tensor<msha
 int=4, int)
  0.13%  19.393ms         2  9.6966ms  461.46us  18.932ms  void cudnn::detail::activation_fw_4d_kernel<float, floa
::detail::activation_fw_4d_kernel<float, float, int=128, int=1, int=4, cudnn::detail::tanh_func<float>>, cudnnTens
0.09%  14.508ms         1  14.508ms  14.508ms  14.508ms  void cudnn::detail::pooling_fw_4d_kernel<float, float, cu
t *, cudnn::detail::pooling_fw_4d_kernel<float, float, cudnn::detail::maxpooling_func<float, cudnnNanPropagation_t=
d_divisor, float)
  0.04%  6.0571ms        13  465.93us  1.5360us  4.1334ms  [CUDA memcpy HtoD]
  0.02%  3.6285ms         1  3.6285ms  3.6285ms  3.6285ms  sgemm_sm35_ldg_tn_64x16x128x8x32
  0.01%  1.1203ms         1  1.1203ms  1.1203ms  1.1203ms  void mshadow::cuda::SoftmaxKernel<int=8, float, mshadow
sor<mshadow::gpu, int=2, float>, float>>(mshadow::gpu, int=2, unsigned int)
0.00%  754.77us        12  62.897us  2.1120us  381.02us  void mshadow::cuda::MapPlanKernel<mshadow::sv::saveto, in
mshadow::expr::ScalarExp<float>, float>>(mshadow::gpu, unsigned int, mshadow::Shape<int=2>, int=2)
0.00%  436.79us         2  218.40us  16.704us  420.09us  void mshadow::cuda::MapPlanKernel<mshadow::sv::plusto, in
mshadow::expr::Broadcast1DExp<mshadow::Tensor<mshadow::gpu, int=1, float>, float, int=2, int=1>, float>>(mshadow::
  0.00%  394.52us         1  394.52us  394.52us  394.52us  sgemm_sm35_ldg_tn_32x16x64x8x16
0.00%  24.127us         1  24.127us  24.127us  24.127us  void mshadow::cuda::MapPlanKernel<mshadow::sv::saveto, in
mshadow::expr::ReduceWithAxisExp<mshadow::red::maximum, mshadow::Tensor<mshadow::gpu, int=3, float>, float, int=3,
  0.00%  9.5360us         1  9.5360us  9.5360us  9.5360us  [CUDA memcpy DtoH]
==312== API calls:
Time(%)      Time     Calls       Avg       Min       Max  Name
79.53%  15.3710s         1  15.3710s  15.3710s  15.3710s  cudaDeviceSynchronize
  9.68%  1.87162s        18  103.98ms  17.621us  935.63ms  cudaStreamCreateWithFlags
  5.94%  1.14793s        10  114.79ms  1.0920us  325.67ms  cudaFree
  4.32%  834.24ms        23  36.271ms  235.49us  827.50ms  cudaMemGetInfo
  0.41%  78.801ms        25  3.1520ms  5.9130us  42.576ms  cudaStreamSynchronize
  0.06%  12.255ms         8  1.5319ms  12.024us  6.0906ms  cudaMemcpy2DAsync
  0.03%  6.4606ms        41  157.57us  11.808us  1.0997ms  cudaMalloc
  0.01%  1.3601ms         4  340.03us  338.32us  343.67us  cuDeviceTotalMem
  0.01%  1.1044ms       114  9.6870us     876ns  428.35us  cudaEventCreateWithFlags
  0.00%  844.27us       352  2.3980us     244ns  63.183us  cuDeviceGetAttribute
  0.00%  531.53us        24  22.146us  11.055us  76.631us  cudaLaunch
  0.00%  272.43us         6  45.405us  25.825us  120.50us  cudaMemcpy
  0.00%  246.65us         4  61.662us  35.457us  96.917us  cudaStreamCreate
  0.00%  102.19us         4  25.548us  19.805us  29.315us  cuDeviceGetName
  0.00%  84.886us       104     816ns     577ns  2.0030us  cudaDeviceGetAttribute
  0.00%  69.761us        30  2.3250us     665ns  7.9140us  cudaSetDevice
  0.00%  60.761us       145     419ns     253ns  2.4850us  cudaSetupArgument
  0.00%  45.927us         2  22.963us  22.753us  23.174us  cudaStreamCreateWithPriority
  0.00%  29.938us        10  2.9930us  1.2480us  8.2080us  cudaGetDevice
  0.00%  28.124us        24  1.1710us     439ns  2.5890us  cudaConfigureCall
  0.00%  8.3060us        17     488ns     362ns     795ns  cudaPeekAtLastError
  0.00%  5.8790us         6     979ns     387ns  2.3060us  cuDeviceGetCount
  0.00%  5.2570us         1  5.2570us  5.2570us  5.2570us  cudaStreamGetPriority
```

Figure 1. Profiling of m1.2.py with GPU running convolution

```
==310== Profiling result:
Time(%)      Time     Calls       Avg       Min       Max  Name
36.80%  49.975ms         1  49.975ms  49.975ms  49.975ms  void cudnn::detail::implicit_convolve_sgemm<flo
nt=5, int=3, int=3, int=3, int=1, bool=1, bool=0, bool=1>*, float const *, kernel_conv_params, int, float
 28.60%  38.845ms         1  38.845ms  38.845ms  38.845ms  sgemm_sm35_ldg_tn_128x8x256x16x32
14.26%  19.372ms         2  9.6861ms  457.95us  18.914ms  void cudnn::detail::activation_fw_4d_kernel<flo
dnn::detail::tanh_func<float>>, cudnnTensorStruct*, float, cudnnTensorStruct*, int, cudnnTensorStruct*)
 10.64%  14.447ms         1  14.447ms  14.447ms  14.447ms  void cudnn::detail::pooling_fw_4d_kernel<float
maxpooling_func<float, cudnnNanPropagation_t=0>, int=0>, cudnnTensorStruct*, cudnnPoolingStruct, float, c
  4.97%  6.7499ms        13  519.23us  1.5360us  4.8203ms  [CUDA memcpy HtoD]
  2.72%  3.6931ms         1  3.6931ms  3.6931ms  3.6931ms  sgemm_sm35_ldg_tn_64x16x128x8x32
0.82%  1.1113ms         1  1.1113ms  1.1113ms  1.1113ms  void mshadow::cuda::SoftmaxKernel<int=8, float,
)
0.55%  748.60us        12  62.383us  2.0800us  377.82us  void mshadow::cuda::MapPlanKernel<mshadow::sv::s
adow::Shape<int=2>, int=2)
0.32%  434.33us         2  217.17us  17.152us  417.18us  void mshadow::cuda::MapPlanKernel<mshadow::sv::p
>, float, int=2, int=1>, float>>(mshadow::gpu, unsigned int, mshadow::Shape<int=2>, int=2)
  0.29%  391.93us         1  391.93us  391.93us  391.93us  sgemm_sm35_ldg_tn_32x16x64x8x16
0.02%  23.392us         1  23.392us  23.392us  23.392us  void mshadow::cuda::MapPlanKernel<mshadow::sv::s
mshadow::gpu, int=3, float>, float, int=3, bool=1, int=2>, float>>(mshadow::gpu, unsigned int, mshadow::S
  0.01%  9.7280us         1  9.7280us  9.7280us  9.7280us  [CUDA memcpy DtoH]
==310== API calls:
Time(%)      Time     Calls       Avg       Min       Max  Name
 46.94%  1.88272s        18  104.60ms  20.468us  941.00ms  cudaStreamCreateWithFlags
 28.44%  1.14063s        10  114.06ms     934ns  323.20ms  cudaFree
 20.70%  830.50ms        24  34.604ms  215.62us  823.31ms  cudaMemGetInfo
3.19%  128.10ms        25  5.1240ms  5.6380us  83.234ms  cudaStreamSynchronize
  0.40%  15.922ms         8  1.9902ms  18.694us  5.4170ms  cudaMemcpy2DAsync
  0.21%  8.2295ms        42  195.94us  11.761us  1.6084ms  cudaMalloc
  0.03%  1.3842ms         4  346.06us  337.63us  362.05us  cuDeviceTotalMem
  0.02%  842.46us       352  2.3930us     245ns  63.567us  cuDeviceGetAttribute
  0.02%  787.66us       114  6.9090us  1.0370us  158.00us  cudaEventCreateWithFlags
  0.02%  655.11us        23  28.482us  17.745us  111.41us  cudaLaunch
  0.01%  439.35us         6  73.225us  33.590us  125.25us  cudaMemcpy
  0.01%  382.03us         2  191.01us  24.115us  357.91us  cudaStreamCreateWithPriority
  0.01%  254.67us         4  63.667us  46.661us  91.407us  cudaStreamCreate
  0.00%  112.91us         4  28.226us  22.044us  37.280us  cuDeviceGetName
  0.00%  87.311us       110     793ns     495ns  2.2990us  cudaDeviceGetAttribute
  0.00%  82.884us        32  2.5900us     657ns  7.3020us  cudaSetDevice
  0.00%  78.627us       147     534ns     308ns  1.4770us  cudaSetupArgument
  0.00%  28.155us        23  1.2240us     702ns  3.2990us  cudaConfigureCall
  0.00%  23.458us        10  2.3450us  1.1460us  7.8010us  cudaGetDevice
  0.00%  12.896us         1  12.896us  12.896us  12.896us  cudaBindTexture
  0.00%  10.845us        16     677ns     490ns     984ns  cudaPeekAtLastError
  0.00%  5.4750us         2  2.7370us  2.6570us  2.8180us  cudaEventRecord
  0.00%  5.3770us         1  5.3770us  5.3770us  5.3770us  cudaStreamGetPriority
  0.00%  5.2450us         2  2.6220us  2.5280us  2.7170us  cudaStreamWaitEvent
  0.00%  5.1090us         6     851ns     287ns  1.5910us  cuDeviceGetCount
0.00%  4.4690us         2  2.2340us  2.0980us  2.3710us  cudaDeviceGetStreamPriorityRange
```

Figure 2. Profiling of m3.1.py with GPU running convolution on ECE408-High model with dataset of 10000

	Baseline	Shared Memory on Convolutional Kernel	Matrix Multiplication baseline	Shared Memory On Matrix Multiplication	Final Optimization with parallelism
Time	15.3515 s	258.34ms	274.137ms	200.268ms	118.445ms
Speedup over baseline	1x	60x	56x	76x	130x

Table 1. Summary of each optimization and its running time and speedup over baseline

```
Op Time: 0.274137
Correctness: 0.8562 Model: ece408-high
==311== Profiling application: python final.py ece408-high 10000
==311== Profiling result:
Time(%)      Time     Calls       Avg       Min       Max  Name
 63.13%  226.74ms         1  226.74ms  226.74ms  226.74ms  mxnet::op::matrixMultiply(float*, float*, floa
 10.77%  38.684ms         1  38.684ms  38.684ms  38.684ms  sgemm_sm35_ldg_tn_128x8x256x16x32
  6.27%  22.531ms         1  22.531ms  22.531ms  22.531ms  mxnet::op::matrixShift(float*, float*, int, in
5.41%  19.444ms         1  19.444ms  19.444ms  19.444ms  void mshadow::cuda::MapPlanLargeKernel<mshadow::
:Tensor<mshadow::gpu, int=4, float>, float>, mshadow::expr::Plan<mshadow::expr::BinaryMapExp<mshadow::op:
hadow::gpu, int=4, float>, float, int=1>, float>>(mshadow::gpu, unsigned int, mshadow::Shape<int=2>, int=
  5.40%  19.378ms         2  9.6891ms  458.97us  18.919ms  void cudnn::detail::activation_fw_4d_kernel<fl
nc<float>>(cudnnTensorStruct, float const *, cudnn::detail::activation_fw_4d_kernel<float, float, int=128
nsorStruct*, float, cudnnTensorStruct*, int, cudnnTensorStruct*)
4.02%  14.435ms         1  14.435ms  14.435ms  14.435ms  void cudnn::detail::pooling_fw_4d_kernel<float,
agation_t=0>, int=0>(cudnnTensorStruct, float const *, cudnn::detail::pooling_fw_4d_kernel<float, float,
_t=0>, int=0>, cudnnTensorStruct*, cudnnPoolingStruct, float, cudnnPoolingStruct, int, cudnn::reduced_div
  1.73%  6.2103ms        13  477.71us  1.5360us  4.2881ms  [CUDA memcpy HtoD]
  1.50%  5.3908ms         1  5.3908ms  5.3908ms  5.3908ms  mxnet::op::unroll_kernel(int, int, int, int, i
  1.01%  3.6453ms         1  3.6453ms  3.6453ms  3.6453ms  sgemm_sm35_ldg_tn_64x16x128x8x32
0.31%  1.1130ms         1  1.1130ms  1.1130ms  1.1130ms  void mshadow::cuda::SoftmaxKernel<int=8, float,
float>, float>, mshadow::expr::Plan<mshadow::Tensor<mshadow::gpu, int=2, float>, float>>(mshadow::gpu, in
  0.21%  746.84us        12  62.236us  2.1120us  376.60us  void mshadow::cuda::MapPlanKernel<mshadow::sv:
ow::gpu, int=2, float>, float>, mshadow::expr::Plan<mshadow::expr::ScalarExp<float>, float>>(mshadow::gpu
0.12%  434.30us         2  217.15us  17.056us  417.24us  void mshadow::cuda::MapPlanKernel<mshadow::sv::p
::gpu, int=2, float>, float>, mshadow::expr::Plan<mshadow::expr::Broadcast1DExp<mshadow::Tensor<mshadow::
::gpu, unsigned int, mshadow::Shape<int=2>, int=2)
  0.11%  390.94us         1  390.94us  390.94us  390.94us  sgemm_sm35_ldg_tn_32x16x64x8x16
0.01%  22.975us         1  22.975us  22.975us  22.975us  void mshadow::cuda::MapPlanKernel<mshadow::sv::s
::gpu, int=2, float>, float>, mshadow::expr::Plan<mshadow::expr::ReduceWithAxisExp<mshadow::red::maximum,
3, bool=1, int=2>, float>>(mshadow::gpu, unsigned int, mshadow::Shape<int=2>, int=2)
  0.00%  10.111us         1  10.111us  10.111us  10.111us  [CUDA memcpy DtoH]
```

Figure 3. Nvprof of matrix multiplication baseline on ECE408-High Model

```
Op Time: 0.118445
Correctness: 0.8562 Model: ece408-high
==311== Profiling application: python final.py ece408-high 10000
==311== Profiling result:
Time(%)      Time     Calls       Avg       Min       Max  Name
 35.38%  74.424ms         1  74.424ms  74.424ms  74.424ms  mxnet::op::matrixMultiplyShared2(float*, float*, float*,
 18.30%  38.491ms         1  38.491ms  38.491ms  38.491ms  sgemm_sm35_ldg_tn_128x8x256x16x32
9.24%  19.429ms         1  19.429ms  19.429ms  19.429ms  void mshadow::cuda::MapPlanLargeKernel<mshadow::sv::saveto
:Tensor<mshadow::gpu, int=4, float>, float>, mshadow::expr::Plan<mshadow::expr::BinaryMapExp<mshadow::op::mul, msha
hadow::gpu, int=4, float>, float, int=1>, float>>(mshadow::gpu, unsigned int, mshadow::Shape<int=2>, int=4, int)
  9.20%  19.360ms         2  9.6802ms  455.26us  18.905ms  void cudnn::detail::activation_fw_4d_kernel<float, float
nc<float>>(cudnnTensorStruct, float const *, cudnn::detail::activation_fw_4d_kernel<float, float, int=128, int=1, i
nsorStruct*, float, cudnnTensorStruct*, int, cudnnTensorStruct*)
  9.09%  19.119ms         1  19.119ms  19.119ms  19.119ms  mxnet::op::matrixShift(float*, float*, int, int, int, in
  6.96%  14.635ms        13  1.1258ms  1.5680us  9.9523ms  [CUDA memcpy HtoD]
6.84%  14.390ms         1  14.390ms  14.390ms  14.390ms  void cudnn::detail::pooling_fw_4d_kernel<float, float, cud
agation_t=0>, int=0>(cudnnTensorStruct, float const *, cudnn::detail::pooling_fw_4d_kernel<float, float, cudnn::det
_t=0>, int=0>, cudnnTensorStruct*, cudnnPoolingStruct, float, cudnnPoolingStruct, int, cudnn::reduced_divisor, floa
  2.55%  5.3649ms         1  5.3649ms  5.3649ms  5.3649ms  mxnet::op::unroll_kernel(int, int, int, int, int, float*
  1.16%  2.4421ms         1  2.4421ms  2.4421ms  2.4421ms  sgemm_sm35_ldg_tn_64x16x128x8x32
  0.53%  1.1048ms         1  1.1048ms  1.1048ms  1.1048ms  void mshadow::cuda::SoftmaxKernel<int=8, float, mshadow:
, float>, float>, mshadow::expr::Plan<mshadow::Tensor<mshadow::gpu, int=2, float>, float>>(mshadow::gpu, int=2, uns
0.35%  739.26us        12  61.604us  2.0800us  373.05us  void mshadow::cuda::MapPlanKernel<mshadow::sv::saveto, int
::gpu, int=2, float>, float>, mshadow::expr::Plan<mshadow::expr::ScalarExp<float>, float>>(mshadow::gpu, unsigned i
0.20%  430.59us         2  215.29us  16.640us  413.95us  void mshadow::cuda::MapPlanKernel<mshadow::sv::plusto, int
::gpu, int=2, float>, float>, mshadow::expr::Plan<mshadow::expr::Broadcast1DExp<mshadow::Tensor<mshadow::gpu, int=1
::gpu, unsigned int, mshadow::Shape<int=2>, int=2)
  0.18%  381.21us         1  381.21us  381.21us  381.21us  sgemm_sm35_ldg_tn_32x16x64x8x16
  0.01%  23.392us         1  23.392us  23.392us  23.392us  void mshadow::cuda::MapPlanKernel<mshadow::sv::saveto, i
ow::gpu, int=2, float>, float>, mshadow::expr::Plan<mshadow::expr::ReduceWithAxisExp<mshadow::red::maximum, mshadow
t=3, bool=1, int=2>, float>>(mshadow::gpu, unsigned int, mshadow::Shape<int=2>, int=2)
  0.01%  17.024us         1  17.024us  17.024us  17.024us  [CUDA memcpy DtoH]
  0.00%  4.0960us         1  4.0960us  4.0960us  4.0960us  [CUDA memcpy DtoD]
```

Figure 4. Nvprof of matrix multiplication with final optimizations on ECE408-High Model