ECE 408 Final Project Report

Lihao Yu, Ningkai Wu, Yuan Cheng

1. Baseline Results

- **M1.1: mxnet CPU layer correctness and elapsed time for the whole python program. You can measure the elapsed time of the program with the time command.**

```
* Running time python /src/m1.1.py
Loading fashion-mnist data... done
Loading model... done
EvalMetric: {'accuracy': 0.8673}
10.16user 4.90system 0:05.86elapsed 256%CPU (0avgtext+0avgdata 1624224maxresident)k
0inputs+2624outputs (0major+28320minor)pagefaults 0swaps
* The build folder has been uploaded to http://s3.amazonaws.com/files.rai-project.com/userdata/build-74988929-7ebf-458c-aea4-ffb8dc45d495.tar.gz. The data will be present for only a short duration of time.
* Server has ended your request.
```

Correctness is 0. 8673

Elapsed time is 0:05.86

- **M1.2/M1.3: mxnet GPU layer performance results (nvprof profile). Include your profile, and describe in a few words how the GPU is spending its time. This is to confirm you can generate a profile and can interpret it.**

Change from CPU to GPU

```
* Running time python /src/m1.2.py
Loading fashion-mnist data... done
Loading model...[01:57:48] src/operator/./../cudnn_algoreg-inl.h:112: Running per
formance tests to find the best convolution algorithm, this can take a while...
(setting env variable MXNET_CUDNN_AUTOTUNE_DEFAULT to 0 to disable)
done
EvalMetric: {'accuracy': 0.8673}
2.31user 0.98system 0:03.09elapsed 106%CPU (0avgtext+0avgdata 911960maxresident)
k
0inputs+3136outputs (0major+157716minor)pagefaults 0swaps
* The build folder has been uploaded to http://s3.amazonaws.com/files.rai-projec
t.com/userdata/build-d3854b02-6bdf-4e30-9e5a-f070ba0d505d.tar.gz. The data will
be present for only a short duration of time.
* Server has ended your request.
```

Correctness is 0. 8673

Elapsed time is 0:03.09

Nvprof profile:

EvalMetric: {'accuracy': 0.8673}

==306== Profiling application: python /src/m1.2.py

==306== Profiling result:

Time(%) Time Calls Avg Min Max Name

36.36% 49.299ms 1 49.299ms 49.299ms 49.299ms void cudnn::detail::implicit_convolve_sgemm<float, int=1024, int=5, int=5, int=3, int=3, int=3, int=1, bool=1, bool=0, bool=1>(int, int, int, float const *, int, cudnn::detail::implicit_convolve_sgemm<float, int=1024, int=5, int=5, int=3, int=3, int=3, int=1, bool=1, bool=0, bool=1>*, float const *, kernel_conv_params, int, float, float, int, float const *, float const *, int, int)

28.13% 38.148ms 1 38.148ms 38.148ms 38.148ms sgemm_sm35_ldg_tn_128x8x256x16x32

14.31% 19.407ms 2 9.7035ms 454.46us 18.953ms void cudnn::detail::activation_fw_4d_kernel<float, float, int=128, int=1, int=4, cudnn::detail::tanh_func<float>>(cudnnTensorStruct, float const *, cudnn::detail::activation_fw_4d_kernel<float, float, int=128, int=1, int=4, cudnn::detail::tanh_func<float>>, cudnnTensorStruct*, float, cudnnTensorStruct*, int, cudnnTensorStruct*)

10.61% 14.386ms 1 14.386ms 14.386ms 14.386ms void cudnn::detail::pooling_fw_4d_kernel<float, float, cudnn::detail::maxpooling_func<float,

cudnnNanPropagation_t=0>, int=0>(cudnnTensorStruct, float const *, cudnn::detail::pooling_fw_4d_kernel<float, float, cudnn::detail::maxpooling_func<float, cudnnNanPropagation_t=0>, int=0>, cudnnTensorStruct*, cudnnPoolingStruct, float, cudnnPoolingStruct, int, cudnn::reduced_divisor, float)

5.94% 8.0555ms 13 619.65us 1.6000us 5.7608ms [CUDA memcpy HtoD]

2.66% 3.6125ms 1 3.6125ms 3.6125ms 3.6125ms sgemm_sm35_ldg_tn_64x16x128x8x32

0.81% 1.1017ms 1 1.1017ms 1.1017ms 1.1017ms void mshadow::cuda::SoftmaxKernel<int=8, float, mshadow::expr::Plan<mshadow::Tensor<mshadow::gpu, int=2, float>, float>, mshadow::expr::Plan<mshadow::Tensor<mshadow::gpu, int=2, float>, float>>(mshadow::gpu, int=2, unsigned int)

0.54% 738.26us 12 61.521us 2.0480us 372.73us void mshadow::cuda::MapPlanKernel<mshadow::sv::saveto, int=8, mshadow::expr::Plan<mshadow::Tensor<mshadow::gpu, int=2, float>, float>, mshadow::expr::Plan<mshadow::expr::ScalarExp<float>, float>>(mshadow::gpu, unsigned int, mshadow::Shape<int=2>, int=2)

0.32% 430.52us 2 215.26us 17.023us 413.50us void mshadow::cuda::MapPlanKernel<mshadow::sv::plusto, int=8, mshadow::expr::Plan<mshadow::Tensor<mshadow::gpu, int=2, float>, float>, mshadow::expr::Plan<mshadow::expr::Broadcast1DExp<mshadow::Tensor<mshadow::gpu, int=1, float>, float, int=2, int=1>, float>>(mshadow::gpu, unsigned int, mshadow::Shape<int=2>, int=2)

0.28% 384.57us 1 384.57us 384.57us 384.57us sgemm_sm35_ldg_tn_32x16x64x8x16

0.02% 22.624us 1 22.624us 22.624us 22.624us void mshadow::cuda::MapPlanKernel<mshadow::sv::saveto, int=8, mshadow::expr::Plan<mshadow::Tensor<mshadow::gpu, int=2, float>, float>, mshadow::expr::Plan<mshadow::expr::ReduceWithAxisExp<mshadow::red::maximum, mshadow::Tensor<mshadow::gpu, int=3, float>, float, int=3, bool=1, int=2>, float>>(mshadow::gpu, unsigned int, mshadow::Shape<int=2>, int=2)

0.01% 9.7600us 1 9.7600us 9.7600us 9.7600us [CUDA memcpy DtoH]

==306== API calls:

Time(%) Time Calls Avg Min Max Name

48.30% 2.67306s 18 148.50ms 24.016us 1.33610s cudaStreamCreateWithFlags

28.34% 1.56859s 10 156.86ms 915ns 422.83ms cudaFree

20.48% 1.13351s 24 47.230ms 278.70us 1.12525s cudaMemGetInfo

2.29% 126.91ms 25 5.0764ms 6.8090us 82.536ms cudaStreamSynchronize

0.29% 15.818ms 8 1.9773ms 14.646us 5.8819ms cudaMemcpy2DAsync

0.21% 11.498ms 42 273.77us 8.5690us 2.1006ms cudaMalloc

0.03% 1.5789ms 4 394.73us 368.70us 411.11us cuDeviceTotalMem

0.02% 1.0303ms 352 2.9260us 245ns 82.591us cuDeviceGetAttribute

0.01% 800.35us 114 7.0200us 818ns 383.60us cudaEventCreateWithFlags

0.01% 605.85us 23 26.341us 14.621us 133.68us cudaLaunch

0.01% 380.13us 6 63.355us 21.503us 104.12us cudaMemcpy

0.00% 218.65us 4 54.663us 28.348us 79.858us cudaStreamCreate

0.00% 119.46us 4 29.865us 20.320us 37.383us cuDeviceGetName

0.00% 117.10us 32 3.6590us 1.1590us 30.022us cudaSetDevice

0.00% 111.20us 110 1.0100us 563ns 3.2260us cudaDeviceGetAttribute

0.00% 90.232us 147 613ns 428ns 1.4500us cudaSetupArgument

0.00% 52.013us 2 26.006us 25.563us 26.450us cudaStreamCreateWithPriority

0.00% 30.684us 23 1.3340us 663ns 2.9930us cudaConfigureCall

0.00% 26.930us 10 2.6930us 1.3150us 7.9690us cudaGetDevice

0.00% 13.926us 1 13.926us 13.926us 13.926us cudaBindTexture

0.00% 10.282us 16 642ns 540ns 743ns cudaPeekAtLastError

0.00% 6.1980us 1 6.1980us 6.1980us 6.1980us cudaStreamGetPriority

0.00% 5.9520us 6 992ns 431ns 2.1370us cuDeviceGetCount

0.00% 5.6410us 2 2.8200us 1.9710us 3.6700us cudaEventRecord

0.00% 5.6280us 2 2.8140us 2.1450us 3.4830us cudaStreamWaitEvent

0.00% 5.1710us 6 861ns 528ns 1.6550us cuDeviceGet

0.00% 4.4790us 2 2.2390us 1.7620us 2.7170us cudaDeviceGetStreamPriorityRange

0.00% 4.4090us 6 734ns 478ns 1.0230us cudaGetLastError

0.00% 3.4230us 3 1.1410us 788ns 1.3550us cuInit

0.00% 2.8030us 1 2.8030us 2.8030us 2.8030us cudaUnbindTexture

0.00% 2.6280us 3 876ns 660ns 1.0250us cuDriverGetVersion

0.00% 1.5180us 1 1.5180us 1.5180us 1.5180us cudaGetDeviceCount

The most time-consuming kernels are implicit_convolve_sgemm, sgemm_sm35_ldg_tn_128x8x256x16x32, activation_fw_4d_kernel, pooling_fw_4d_kernel, CUDA memcpy HtoD and sgemm_sm35_ldg_tn_64x16x128x8x32.

implicit_convolve_sgemm uses 49.299ms and 36.36% of time;

sgemm_sm35_ldg_tn_128x8x256x16x32 uses 38.148ms and 28.13% of time;

activation_fw_4d_kernel uses 19.407ms and 14.31% of time;

pooling_fw_4d_kernel uses 14.386ms and 10.61% of time;

CUDA memcpy HtoD uses 8.0555ms and 5.94% of time;

sgemm_sm35_ldg_tn_64x16x128x8x32 uses 3.6125ms and 2.66% of time.

- **M2.1: your baseline cpu implementation correctness and performance results (time). The Op Time: printed by the program will show the time just for the convolution layer. The implementation should have the expected correctness.**

  ✱ Running time python /src/m2.1.py ece408-high 10000

  Loading fashion-mnist data... done

  Loading model... done

  Op Time: 9.297901

  Correctness: 0.8562 Model: ece408-high

  17.35user 2.97system 0:14.06elapsed 144%CPU (0avgtext+0avgdata 2755340maxresident)k0inputs+2624outputs (0major+27825minor)pagefaults 0swaps

✱ Running time python /src/m2.1.py ece408-low 10000

Loading fashion-mnist data... done

Loading model... done

Op Time: 9.236477

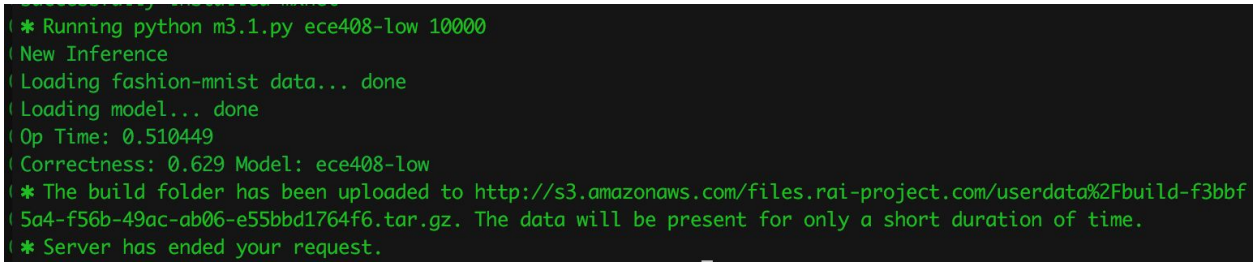
Correctness: 0.629 Model: ece408-low

18.00user 1.46system 0:12.75elapsed 152%CPU (0avgtext+0avgdata 2749056maxresident)k

0inputs+2624outputs (0major+27574minor)pagefaults 0swaps

Correctness are 0.8562 and 0.629 for ece408-high and ece408-low respectively. Op time are 9.066891 and 9.236477 respectively. Time elapse are 0:14.06 and 0:12.75 respectively.

- **M3.1: your baseline gpu implementation performance results (time, nvprof profile). The implementation should have the expected correctness. Include how you divided work amongst your team (even though there is not much work).**



```
* Running python m3.1.py ece408-high 10000
New Inference
Loading fashion-mnist data... done
Loading model... done
Op Time: 0.501864
Correctness: 0.8562 Model: ece408-high
* The build folder has been uploaded to http://s3.amazonaws.com/files.rai-project.com/userdata%2Fbuild-2367fa39-9514-42d3-a223-46c5e963d08b.tar.gz. The data will be present for only a short duration of time.
* Server has ended your request.
```

* Running nvprof python m3.1.py

New Inference

Loading fashion-mnist data... done

==312== NVPROF is profiling process 312, command: python m3.1.py

Loading model... done

Op Time: 0.577761

Correctness: 0.8562 Model: ece408-high

==312== Profiling application: python m3.1.py

==312== Profiling result:

Time(%) Time Calls Avg Min Max Name

84.11% 558.14ms 1 558.14ms 558.14ms 558.14ms 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, int, int)

5.93% 39.359ms 1 39.359ms 39.359ms 39.359ms sgemm_sm35_ldg_tn_128x8x256x16x32

2.95% 19.602ms 1 19.602ms 19.602ms 19.602ms void mshadow::cuda::MapPlanLargeKernel<mshadow::sv::saveto, int=8, int=1024, mshadow::expr::Plan<mshadow::Tensor<mshadow::gpu, int=4, float>, float>, mshadow::expr::Plan<mshadow::expr::BinaryMapExp<mshadow::op::mul, mshadow::expr::ScalarExp<float>, mshadow::Tensor<mshadow::gpu,

int=4, float>, float, int=1>, float>>(mshadow::gpu, unsigned int, mshadow::Shape<int=2>, int=4, int)

2.92% 19.392ms 2 9.6962ms 460.09us 18.932ms void cudnn::detail::activation_fw_4d_kernel<float, float, int=128, int=1, int=4, cudnn::detail::tanh_func<float>>(cudnnTensorStruct, float const *, cudnn::detail::activation_fw_4d_kernel<float, float, int=128, int=1, int=4, cudnn::detail::tanh_func<float>>, cudnnTensorStruct*, float, cudnnTensorStruct*, int, cudnnTensorStruct*)

2.19% 14.500ms 1 14.500ms 14.500ms 14.500ms void cudnn::detail::pooling_fw_4d_kernel<float, float, cudnn::detail::maxpooling_func<float, cudnnNanPropagation_t=0>, int=0>(cudnnTensorStruct, float const *, cudnn::detail::pooling_fw_4d_kernel<float, float, cudnn::detail::maxpooling_func<float, cudnnNanPropagation_t=0>, int=0>, cudnnTensorStruct*, cudnnPoolingStruct, float, cudnnPoolingStruct, int, cudnn::reduced_divisor, float)

0.93% 6.1829ms 13 475.61us 1.5360us 4.2532ms [CUDA memcpy HtoD]

0.55% 3.6573ms 1 3.6573ms 3.6573ms 3.6573ms sgemm_sm35_ldg_tn_64x16x128x8x32

0.17% 1.1207ms 1 1.1207ms 1.1207ms 1.1207ms void mshadow::cuda::SoftmaxKernel<int=8, float, mshadow::expr::Plan<mshadow::Tensor<mshadow::gpu, int=2, float>, float>, mshadow::expr::Plan<mshadow::Tensor<mshadow::gpu, int=2, float>, float>>(mshadow::gpu, int=2, unsigned int)

0.11% 754.84us 12 62.902us 2.0800us 381.15us void mshadow::cuda::MapPlanKernel<mshadow::sv::saveto, int=8, mshadow::expr::Plan<mshadow::Tensor<mshadow::gpu, int=2, float>, float>, mshadow::expr::Plan<mshadow::expr::ScalarExp<float>, float>>(mshadow::gpu, unsigned int, mshadow::Shape<int=2>, int=2)

0.07% 437.40us 2 218.70us 17.152us 420.25us void mshadow::cuda::MapPlanKernel<mshadow::sv::plusto, int=8, mshadow::expr::Plan<mshadow::Tensor<mshadow::gpu, int=2, float>, float>, mshadow::expr::Plan<mshadow::expr::Broadcast1DExp<mshadow::Tensor<mshadow::gpu, int=1, float>, float, int=2, int=1>, float>>(mshadow::gpu, unsigned int, mshadow::Shape<int=2>, int=2)

0.06% 392.54us 1 392.54us 392.54us 392.54us sgemm_sm35_ldg_tn_32x16x64x8x16

0.00% 23.295us 1 23.295us 23.295us 23.295us void mshadow::cuda::MapPlanKernel<mshadow::sv::saveto, int=8, mshadow::expr::Plan<mshadow::Tensor<mshadow::gpu, int=2, float>, float>, mshadow::expr::Plan<mshadow::expr::ReduceWithAxisExp<mshadow::red::maximum, mshadow::Tensor<mshadow::gpu, int=3, float>, float, int=3, bool=1, int=2>, float>>(mshadow::gpu, unsigned int, mshadow::Shape<int=2>, int=2)

0.00% 9.4720us 1 9.4720us 9.4720us 9.4720us [CUDA memcpy DtoH]

==312== API calls:

Time(%) Time Calls Avg Min Max Name

41.22% 1.84119s 18 102.29ms 15.789us 920.24ms cudaStreamCreateWithFlags

25.27% 1.12899s 10 112.90ms 777ns 321.08ms cudaFree

18.26% 815.77ms 23 35.468ms 237.17us 809.00ms cudaMemGetInfo

12.93% 577.70ms 1 577.70ms 577.70ms 577.70ms cudaDeviceSynchronize

1.76% 78.801ms 25 3.1520ms 5.5750us 42.576ms cudaStreamSynchronize

0.28% 12.473ms 8 1.5591ms 11.921us 4.3106ms cudaMemcpy2DAsync

0.15% 6.5828ms 41 160.56us 11.723us 1.1363ms cudaMalloc

0.03% 1.4212ms 4 355.30us 42.035us 1.2537ms cudaStreamCreate

0.03% 1.3641ms 4 341.03us 340.04us 343.50us cuDeviceTotalMem

0.02% 837.32us 352 2.3780us 247ns 62.330us cuDeviceGetAttribute

0.02% 762.82us   114 6.6910us   624ns 338.82us cudaEventCreateWithFlags

 0.01% 527.65us   24 21.985us 10.170us 55.793us cudaLaunch

 0.01% 341.30us   6 56.883us 22.825us 115.09us cudaMemcpy

 0.00% 97.930us   4 24.482us 17.527us 29.758us cuDeviceGetName

 0.00% 70.303us   30 2.3430us   612ns 6.7390us cudaSetDevice

 0.00% 63.950us   104   614ns   418ns 1.5200us cudaDeviceGetAttribute

 0.00% 60.204us   145   415ns   254ns 1.5510us cudaSetupArgument

 0.00% 38.034us   2 19.017us 18.543us 19.491us cudaStreamCreateWithPriority

 0.00% 27.596us   24 1.1490us   345ns 3.4540us cudaConfigureCall

 0.00% 15.417us   10 1.5410us 1.0820us 2.0790us cudaGetDevice

 0.00% 9.0600us   17   532ns   358ns   835ns cudaPeekAtLastError

 0.00% 3.8910us   6   648ns   256ns 1.3710us cuDeviceGetCount

 0.00% 3.8530us   1 3.8530us 3.8530us 3.8530us cudaStreamGetPriority

 0.00% 3.8000us   2 1.9000us 1.4980us 2.3020us cudaStreamWaitEvent

 0.00% 3.4310us   2 1.7150us 1.2800us 2.1510us cudaEventRecord

 0.00% 3.2100us   6   535ns   339ns   787ns cuDeviceGet

 0.00% 2.7540us   2 1.3770us 1.2790us 1.4750us cudaDeviceGetStreamPriorityRange

 0.00% 2.6030us   5   520ns   397ns   699ns cudaGetLastError

0.00% 2.5610us 3 853ns 834ns 867ns cuInit

0.00% 1.8950us 3 631ns 600ns 677ns cuDriverGetVersion

0.00% 1.0880us 1 1.0880us 1.0880us 1.0880us cudaGetDeviceCount

✱ The build folder has been uploaded to http://s3.amazonaws.com/files.rai-project.com/userdata%2Fbuild-be08d18f-a3cd-442a-9d01-da14560821af.tar.gz. The data will be present for only a short duration of time.

The forward layer took 0.577761s, and the forward_kernel took 558.14ms.

2. Optimization Approach and Results

- **How we identified the optimization opportunity**
- According to the CPU implementation we have done for M2.1, the serial code has 6 nested for-loops. Therefore, we decide to exploit the multiple dimensions of CUDA threads to parallelize the code.
- There is also a lot of overhead involved in terms of reading and writing data from the disk memory for completing the matrix multiplication step. In CUDA, the concept of shared memory can be easily applied in order to minimize the overhead of accessing disk memory.
- According to Chapter 16 of the textbook, in order to increase the speed, we reduce the convolutional layer to a matrix multiplication, because we can easily unfold the inputs and we learned highly efficient matrix multiplication on CUDA in MP2 and MP3.
- According to Chapter 16 of the textbook, the current sequential code does not make full usage of the CPU when we read the input feature maps X. In order to optimize performance, we think first unroll the X will make more efficient bandwidth usage of CPU.
- The essential part of the sequential code is doing a matrix multiplication of weight w and input feature map x. So we think we can make use of GPU to parallelize matrix multiplication according to our MP on tiled matrix multiplication. By using the shared memory in the matrix multiplication kernel, we can reduce the global read, which we can efficiently decrease the running time and increase the performance.
- In the original GPU forward function implementation, in a batch, it only have one stream to sequentially process each sample, which means only one stream is working at a time. In order to improve this serial part to

parallel process, we use multiple streams to do the work at the same time, which improves the performance a lot.

- **Why you thought the approach would be fruitful?**

- First of all, in the CPU implementation, there are six for loops. This serial operation will increase the running time by a large factor. By using parallel programming, each thread can work simultaneously to do different calculation, which can improve the speed a lot.
- Matrix multiplication is fast on GPU because it has a high ratio of floating point operations per byte of global memory data access (Chapter 16).
- Accessing data with the disk memory take much more time (500 cycles) than accessing data with the shared memory in GPU device (5 cycles). Thus, using shared memory in matrix multiplication to store part of data from global memory will have significant improvement on the performance.
- In our implementation for loading shared memory, all the neighboring threads will access neighboring memory locations so that the write pattern is coalesced. As we learnt from class, coalesced memory access of threads will optimize the speed of memory access.
- To allow concurrent kernel execution, we utilized 32 streams. Each stream is responsible for one or more sample input unroll and its multiplication depending on batch size.

- **The effect of the optimization. was it fruitful, and why or why not. Use nvprof as needed to justify your explanation.**

- Our final implementation uses three ideas: multiple streams to handle the batch input, unroll for each input sample x to do matrix multiplication and shared memory in matrix multiplication to minimize the overhead of accessing global memory. It turned out our forward function is much faster after these optimizations and reached a Op time below 130ms. The reasons why these worked were mentioned above. However, we did have some unsuccessful attempts. We tried to implement the 4d tiled convolution without changing the format of batch input. This turned out to be much slower than our current implementation. There might be two reasons: first, we could only use one stream for 4d tiled convolution and thus the floating point operations could not be paralleled by multiple streams. Moreover, the 4d input X requires prohibitively large temporary allocation, which might cause the slow down.

Here is our final version Nvprof:

✱ Running nvprof python m3.1.py

New Inference

Loading fashion-mnist data... done

==312== NVPROF is profiling process 312, command: python m3.1.py

Loading model... done

Op Time: 0.268047

Correctness: 0.8562 Model: ece408-high

==312== Profiling application: python m3.1.py

==312== Profiling result:

Time(%) Time Calls Avg Min Max Name

42.70% 188.88ms 10000 18.888us 17.216us 30.208us void mxnet::op::unroll_Kernel<mshadow::gpu, float>(int, int, int, int, float*, float)

33.53% 148.33ms 10000 14.833us 12.928us 165.66us void mxnet::op::matrixMultiplyShared<mshadow::gpu, float>(float*, float, float, int, int, int, int, int, int)

8.85% 39.153ms 1 39.153ms 39.153ms 39.153ms sgemm_sm35_ldg_tn_128x8x256x16x32

4.42% 19.557ms 1 19.557ms 19.557ms 19.557ms void mshadow::cuda::MapPlanLargeKernel<mshadow::sv::saveto, int=8, int=1024, mshadow::expr::Plan<mshadow::Tensor<mshadow::gpu, int=4, float>, float>, mshadow::expr::Plan<mshadow::expr::BinaryMapExp<mshadow::op::mul, mshadow::expr::ScalarExp<float>, mshadow::Tensor<mshadow::gpu, int=4, float>, float, int=1>, float>>(mshadow::gpu, unsigned int, mshadow::Shape<int=2>, int=4, int)

4.38% 19.388ms 2 9.6939ms 458.40us 18.929ms void cudnn::detail::activation_fw_4d_kernel<float, float, int=128, int=1, int=4, cudnn::detail::tanh_func<float>>(cudnnTensorStruct, float const *, cudnn::detail::activation_fw_4d_kernel<float, float, int=128, int=1, int=4, cudnn::detail::tanh_func<float>>, cudnnTensorStruct*, float, cudnnTensorStruct*, int, cudnnTensorStruct*)

3.27% 14.463ms 1 14.463ms 14.463ms 14.463ms void cudnn::detail::pooling_fw_4d_kernel<float, float, cudnn::detail::maxpooling_func<float,

cudnnNanPropagation_t=0>, int=0>(cudnnTensorStruct, float const *, cudnn::detail::pooling_fw_4d_kernel<float, float, cudnn::detail::maxpooling_func<float, cudnnNanPropagation_t=0>, int=0>, cudnnTensorStruct*, cudnnPoolingStruct, float, cudnnPoolingStruct, int, cudnn::reduced_divisor, float)

1.39% 6.1563ms 13 473.56us 1.5360us 4.2268ms [CUDA memcpy HtoD]

0.84% 3.7127ms 1 3.7127ms 3.7127ms 3.7127ms sgemm_sm35_ldg_tn_64x16x128x8x32

0.25% 1.1160ms 1 1.1160ms 1.1160ms 1.1160ms void mshadow::cuda::SoftmaxKernel<int=8, float, mshadow::expr::Plan<mshadow::Tensor<mshadow::gpu, int=2, float>, float>, mshadow::expr::Plan<mshadow::Tensor<mshadow::gpu, int=2, float>, float>>(mshadow::gpu, int=2, unsigned int)

0.17% 751.04us 12 62.586us 2.0800us 378.88us void mshadow::cuda::MapPlanKernel<mshadow::sv::saveto, int=8, mshadow::expr::Plan<mshadow::Tensor<mshadow::gpu, int=2, float>, float>, mshadow::expr::Plan<mshadow::expr::ScalarExp<float>, float>>(mshadow::gpu, unsigned int, mshadow::Shape<int=2>, int=2)

0.10% 434.66us 2 217.33us 16.768us 417.89us void mshadow::cuda::MapPlanKernel<mshadow::sv::plusto, int=8, mshadow::expr::Plan<mshadow::Tensor<mshadow::gpu, int=2, float>, float>, mshadow::expr::Plan<mshadow::expr::Broadcast1DExp<mshadow::Tensor<mshadow::gpu, int=1, float>, float, int=2, int=1>, float>>(mshadow::gpu, unsigned int, mshadow::Shape<int=2>, int=2)

0.09% 394.05us 1 394.05us 394.05us 394.05us sgemm_sm35_ldg_tn_32x16x64x8x16

0.01% 22.880us 1 22.880us 22.880us 22.880us void mshadow::cuda::MapPlanKernel<mshadow::sv::saveto, int=8, mshadow::expr::Plan<mshadow::Tensor<mshadow::gpu, int=2, float>, float>, mshadow::expr::Plan<mshadow::expr::ReduceWithAxisExp<mshadow::red::maximum, mshadow::Tensor<mshadow::gpu, int=3, float>, float, int=3, bool=1, int=2>, float>>(mshadow::gpu, unsigned int, mshadow::Shape<int=2>, int=2)

0.00% 9.9840us 1 9.9840us 9.9840us 9.9840us [CUDA memcpy DtoH]

==312== API calls:

Time(%) Time Calls Avg Min Max Name

44.33% 1.95141s 18 108.41ms 18.089us 975.33ms cudaStreamCreateWithFlags

27.01% 1.18903s 10 118.90ms 697ns 339.65ms cudaFree

21.23% 934.73ms 23 40.640ms 235.80us 927.95ms cudaMemGetInfo

3.97% 174.55ms 20023 8.7170us 6.5840us 2.3893ms cudaLaunch

1.79% 78.579ms 25 3.1432ms 5.7640us 42.478ms cudaStreamSynchronize

0.96% 42.157ms 150136 280ns 253ns 159.16us cudaSetupArgument

0.28% 12.142ms 8 1.5177ms 16.781us 4.2957ms cudaMemcpy2DAsync

0.16% 7.1068ms 42 169.21us 11.640us 1.1112ms cudaMalloc

0.16% 6.8811ms 20023 343ns 294ns 14.727us cudaConfigureCall

0.04% 1.5431ms 36 42.862us 12.346us 331.59us cudaStreamCreate

0.03% 1.3606ms 4 340.16us 338.80us 343.56us cuDeviceTotalMem

0.02% 942.53us 352 2.6770us 245ns 155.23us cuDeviceGetAttribute

0.02% 713.60us 114 6.2590us 630ns 336.01us cudaEventCreateWithFlags

0.01% 365.53us 6 60.921us 25.128us 127.97us cudaMemcpy

0.00% 111.31us 4 27.827us 22.145us 34.336us cuDeviceGetName

0.00% 77.266us 30 2.5750us 629ns 9.6980us cudaSetDevice

0.00% 72.910us 1 72.910us 72.910us 72.910us cudaDeviceSynchronize

0.00% 68.661us 104 660ns 417ns 1.9650us cudaDeviceGetAttribute

0.00% 43.718us 2 21.859us 19.225us 24.493us cudaStreamCreateWithPriority

0.00% 21.187us 10 2.1180us 1.2630us 6.5990us cudaGetDevice

0.00% 9.8220us 17 577ns 328ns 1.0290us cudaPeekAtLastError

0.00% 5.2710us 6 878ns 293ns 2.4000us cuDeviceGetCount

0.00% 5.0530us 1 5.0530us 5.0530us 5.0530us cudaStreamGetPriority

```
 0.00%  4.2430us         2  2.1210us  1.7330us  2.5100us  cudaStreamWaitEvent
 0.00%  4.2230us         2  2.1110us  1.4370us  2.7860us  cudaEventRecord
 0.00%  3.3090us         5     661ns     560ns     810ns  cudaGetLastError
 0.00%  3.0220us         6     503ns     377ns     692ns  cuDeviceGet
 0.00%  2.9900us         2  1.4950us  1.3700us  1.6200us
cudaDeviceGetStreamPriorityRange
 0.00%  2.7670us         3     922ns     845ns  1.0130us  cuInit
 0.00%  2.1430us         3     714ns     645ns     753ns  cuDriverGetVersion
 0.00%     986ns         1     986ns     986ns     986ns  cudaGetDeviceCount
```

- **Any external references used during identification or development of the optimization**

  - Kirk, D. & Hwu, W. (2016). Chapter 16: Application Case Study- Machine Learning.

- **How your team organized and divided up this work**

- Lihao Yu is responsible for writing the report part 2, milestone 1&2 and writing the host functions for final optimization.
- Ningkai Wu is responsible for writing the report part1, milestone 3 and writing the unrollX kernel and host forward function, matrix multiplication kernel and host for final optimization.
- Yuan Cheng is responsible for writing the report part1, milestone 3, writing the matrix multiplication kernel, unroll X and specific ideas on how to optimize the project.

3. References

- Kirk, D. & Hwu, W. (2016). Chapter 16: Application Case Study- Machine Learning.

4. Suggestions for Improving Next Year

Provide more information about how to debug the cuda code.

Fix the typos in textbook.

Thanks for the help from Professor Hwu and all qualified TAs.