ECE408/CS483 Group Project Report

Group Name: airport
Group Member: Pei Liu (NetID:peiliu3, UIN: 673753078)
Zhiyu Zhu (NetID:zzhu24, UIN: 672839294)
Ye Zhang (yezhang4, UIN: 652771627)

1. [List of all Kernel calls that collectively consume more than 90% of the program time]

Type	Time(%)	Time	Calls	Avg	Min	Max	Name
GPU activities	36.90%	37.688ms	20	0.8844ms	1.1200us	35.613ms	[CUDA memcpy HtoD]
GPU activities	22.70%	23.179ms	1	23.179ms	23.179ms	23.179ms	volta_scudnn_128x32_relu_interior_nn_v1
GPU activities	20.74%	21.177ms	1	21.177ms	21.177ms	21.177ms	void cudnn::detail::implicit_convolve_sgemm
GPU activities	7.39%	7.5441ms	1	7.5441ms	7.5441ms	7.5441ms	volta_sgemm_128x128_tn
GPU activities	7.26%	7.4096ms	2	3.7048ms	25.312us	7.3843ms	void cudnn::detail::activation_fw_4d_kernel
GPU activities	4.31%	4.3989ms	1	4.3989ms	4.3989ms	4.3989ms	void cudnn::detail::pooling_fw_4d_kernel

2. [List of all CUDA API calls that collectively consume more than 90% of the program time]

Time(%)	Time	Calls	Avg	Min	Max	Name
38.23%	2.65797s	22	120.82ms	13.101us	1.41059s	cudaStreamCreateWithFlags
35.84%	2.49162s	24	103.82ms	66.484us	2.48456s	cudaMemGetInfo

21.29%	1.48035s	19	77.913ms	284ns	391.27ms	cudaFree

3. [An explanation of the difference between kernels and API calls]
API calls, which are defined in cuda.h files, are used to allocate global memory, and transfer pertinent data from host memory to device memory, and transfer result data from device memory to host memory and free the device memory if no more needed. Kernel calls are the functions that launched by cuda programs to initiate parallel execution. The code in the kernel will be executed by all the threads during the parallel phase. The API calls are executed on the hosts, while the kernel calls would be executed on the devices. Since API calls are executed on the hosts and used to manage memory, it usually takes much longer to perform than Kernel calls based on the data in the table above.

4. [Output of rai running MXNet on the CPU]

```
* Running /usr/bin/time python m1.1.py
Loading fashion-mnist data... done
Loading model... done
New Inference
EvalMetric: {'accuracy': 0.8177}
```

5. [List program run time]

```
19.77user 4.10system 0:13.49elapsed 176%CPU (0avgtext+0avgdata 5956368maxresident)k
0inputs+2856outputs (0ma
jor+1584516minor)pagefaults 0swaps
```

6. [Output of rai running MXNet on the GPU]

```
* Running /usr/bin/time python m1.2.py
Loading fashion-mnist data... done
Loading model... done
New Inference
EvalMetric: {'accuracy': 0.8177}
```

7. [List program run time]

```
4.29user 2.87system 0:04.71elapsed 152%CPU (0avgtext+
0avgdata 2837188maxresident)k
0inputs+4568outputs (0major+703949minor)pagefaults 0swaps
```

ECE408/CS483 Group Project Report

Group Name: airport
Group Member: Pei Liu (NetID:peiliu3, UIN: 673753078)
Zhiyu Zhu (NetID:zzhu24, UIN: 672839294)
Ye Zhang (yezhang4, UIN: 652771627)

MileStone 2

10,000 Images

[python m2.1.py program output]

```
New Inference
Op Time: 20.973760
Op Time: 102.504921
Correctness: 0.8171 Model: ece408
```

[python m2.1.py runtime]

```
134.87user 10.07system 2:07.17elapsed 113%CPU (0avgtext+0avgdata 5867612maxresid
ent)k
0inputs+8outputs (0major+2249865minor)pagefaults 0swaps
```

ECE408/CS483 Group Project Report

Group Name: airport
Group Member: Pei Liu (NetID:peiliu3, UIN: 673753078)
Zhiyu Zhu (NetID:zzhu24, UIN: 672839294)
Ye Zhang (yezhang4, UIN: 652771627)

MileStone 3

10,000 Images

[python m3.1.py program output]

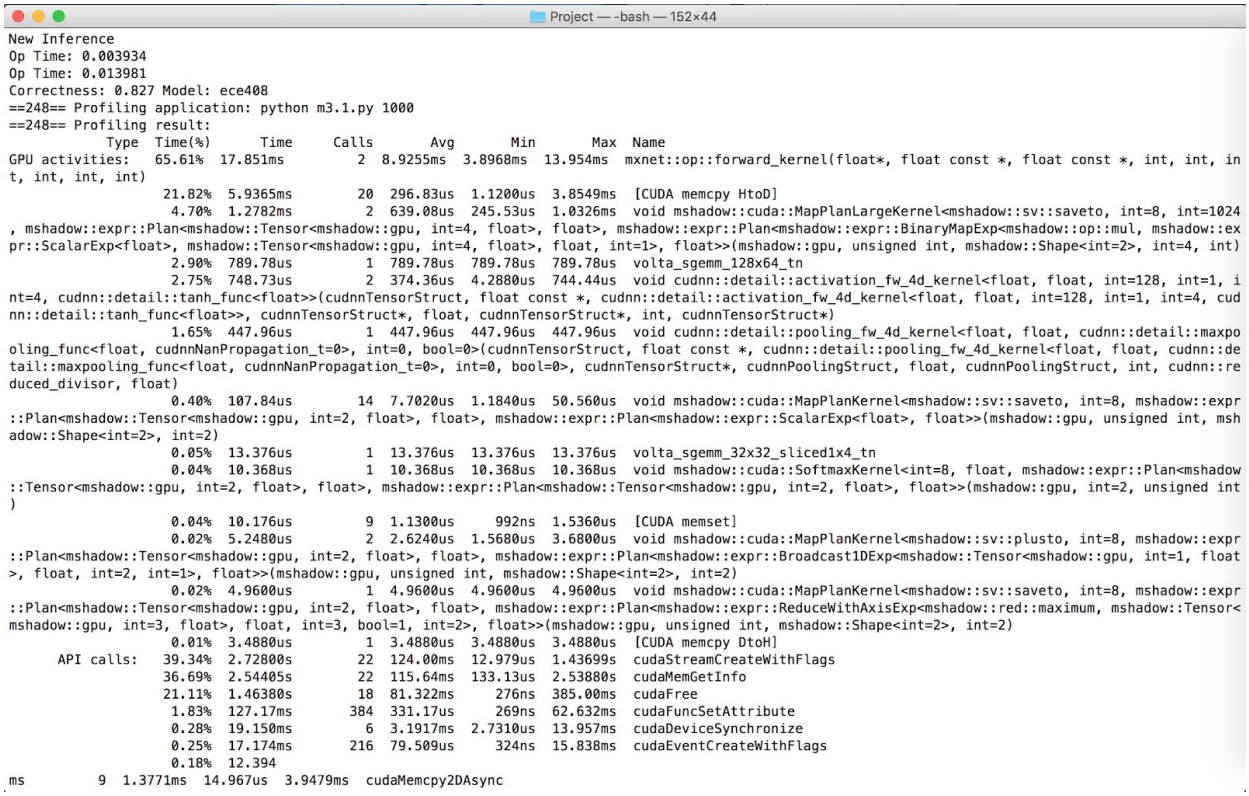
```
Op Time: 0.038911
Op Time: 0.126872
Correctness: 0.8171 Model: ece408
==248== Profiling application: python m3.1.py 10000
```

```
      API calls:   38.00%  2.76718s        22  125.78ms  13.218us  1.44515s  cudaStreamCreateWithFlags
                   35.16%  2.55974s        22  116.3
5ms  67.441us  2.55534s  cudaMemGetInfo
                   21.11%  1.53675s        18  85.375ms     308ns  409.76ms  cudaFree
                    2.42%  175.94ms         6  29.323ms  1.7510us  125.94ms  cudaDeviceSynchronize
                    1.09%  79.488ms       384  207.00us     269ns  68.994ms  cudaFuncSetAttribute
                    0.98%  71.445ms         9  7.9384ms  24.751us  33.504ms  cudaMemcpy2DAsync
                    0.64%  46.449ms        66  703.78us  3.8620us  35.192ms  cudaMalloc
                    0.25%  18.517ms        29  638.53us  1.6960us  10.964ms  cudaStreamSynchronize
                    0.11%  8.0336ms         6  1.3389ms     888ns  8.0104ms  cudaEventCreate
                    0.09%  6.5329ms         4  1.6332ms  425.05us  2.6460ms  cudaGetDeviceProperties
                    0.05%  3.7032ms       375  9.8750us     100ns  1.8112ms  cuDeviceGetAttribute
                    0.03%  2.5218ms       216  11.675us     322ns  1.6786ms  cudaEventCreateWithFlags
                    0.01%  915.19us         8  114.40us  13.353us  703.48us  cudaStreamCreateWithPriority
                    0.01%  735.87us         2  367.93us  45.778us  690.09us  cudaHostAlloc
                    0.01%  722.72us         8  90.339us  7.8080us  473.31us  cudaMemsetAsync
                    0.01%  575.36us        12  47.946us  5.0420us  203.16us  cudaMemcpy
                    0.01%  572.33us         4  143.08us  94.795us  276.00us  cuDeviceTotalMem
                    0.01%  487.16us         4  121.79us  66.064us  204.72us  cudaStreamCreate
                    0.01%  409.95us        27  15.183us  6.3100us  42.783us  cudaLaunchKernel
                    0.00%  256.84us         4  64.209us  44.754us  103.86us  cuDeviceGetName
                    0.00%  114.73us       202     567ns     198ns  2.4870us  cudaDeviceGetAttribute
                    0.00%  94.095us        29  3.2440us     477ns  11.156us  cudaSetDevice
                    0.00%  30.233us        18  1.6790us     210ns  4.8850us  cudaGetDevice
                    0.00%  9.1000us         2  4.5500us  3.8820us  5.2180us  cudaEventRecord
                    0.00%  6.1020us        20     305ns     134ns     640ns  cudaPeekAtLastError
                    0.00%  4.5050us         2  2.2520us  1.8700us  2.6350us  cudaHostGetDevicePointer
                    0.00%  2.9230us         2  1.4610us     885ns  2.0380us  cudaDeviceGetStreamPriorityRange
                    0.00%  2.7660us         1  2.7660us  2.7660us  2.7660us  cuDeviceGetPCIBusId
                    0.00%  2.6800us         6     446ns     101ns     873ns  cuDeviceGetCount
                    0.00%  2.1730us         5     434ns     147ns     688ns  cuDeviceGet
                    0.00%  2.0600us         3     686ns     351ns  1.2770us  cuDriverGetVersion
                    0.00%  2.0210us         3     673ns     451ns  1.0910us  cuInit
                    0.00%  1.9910us         4     497ns     195ns     822ns  cudaGetDeviceCount
                    0.00%  1.1880us         5     237ns     186ns     269ns  cudaGetLastError
```

1,000 Images

[python m3.1.py program output]

```
Op Time: 0.003924
Op Time: 0.013979
Correctness: 0.827 Model: ece408
==248== Profiling application: python m3.1.py 1000
```



100 Images

[python m3.1.py program output]

```
Op Time: 0.000426
Op Time: 0.001474
Correctness: 0.85 Model: ece408
==248== Profiling application: python m3.1.py 100
```

```
     API calls:   36.74%  2.61720s        22  118.96ms  12.859us  1.40162s  cudaStreamCreateWithFlags
                  35.78%  2.54861s        22  115.8
ns  66.028us  2.54409s  cudaMemGetInfo
                  20.52%  1.46147s        18  81.193ms     249ns  380.45ms  cudaFree
                   2.67%  190.08ms         6  31.680ms  1.7320us  134.73ms  cudaDeviceSynchronize
                   1.00%  71.535ms         9  7.9483ms  33.887us  33.424ms  cudaMemcpy2DAsync
                   0.97%  69.071ms       384  179.87us     257ns  25.419ms  cudaFuncSetAttribute
                   0.95%  67.578ms       216  312.86us     303ns  55.898ms  cudaEventCreateWithFlags
                   0.83%  58.834ms        66  891.42us  4.8350us  23.331ms  cudaMalloc
                   0.26%  18.839ms        29  649.62us  1.6120us  11.164ms  cudaStreamSynchronize
                   0.09%  6.5018ms         6  1.0836ms     456ns  4.5742ms  cudaEventCreate
                   0.07%  4.8565ms         4  1.2141ms  449.20us  1.7944ms  cudaGetDeviceProperties
                   0.03%  2.4437ms       375  6.5160us      99ns  328.72us  cuDeviceGetAttribute
                   0.03%  2.3204ms         2  1.1602ms  45.999us  2.2744ms  cudaHostAlloc
                   0.01%  660.23us         4  165.06us  91.615us  272.36us  cuDeviceTotalMem
                   0.01%  583.38us        27  21.606us  6.5810us  53.411us  cudaLaunchKernel
                   0.01%  580.87us        12  48.405us  5.2600us  186.79us  cudaMemcpy
                   0.01%  514.29us         4  128.57us  64.191us  231.26us  cudaStreamCreate
                   0.00%  319.79us         4  79.947us  43.068us  104.20us  cuDeviceGetName
                   0.00%  250.36us         8  31.295us  8.5430us  110.95us  cudaMemsetAsync
                   0.00%  152.92us         8  19.115us  12.815us  46.521us  cudaStreamCreateWithPriority
                   0.00%  118.93us       202     588ns     187ns  2.6780us  cudaDeviceGetAttribute
                   0.00%  97.464us        29  3.3600us     486ns  11.036us  cudaSetDevice
                   0.00%  26.178us        18  1.4540us     214ns  2.8490us  cudaGetDevice
                   0.00%  10.616us         2  5.3080us  5.1110us  5.5050us  cudaEventRecord
                   0.00%  6.9410us        20     347ns     134ns     607ns  cudaPeekAtLastError
                   0.00%  4.8640us         2  2.4320us  2.0850us  2.7790us  cudaHostGetDevicePointer
                   0.00%  3.9880us         1  3.9880us  3.9880us  3.9880us  cuDeviceGetPCIBusId
                   0.00%  3.7530us         2  1.8760us  1.2070us  2.5460us  cudaDeviceGetStreamPriorityRange
                   0.00%  3.5400us         6     590ns     201ns  1.7770us  cuDeviceGetCount
                   0.00%  3.3610us         4     840ns     312ns  1.9470us  cudaGetDeviceCount
                   0.00%  2.6110us         3     870ns     500ns  1.2110us  cuInit
                   0.00%  2.5070us         5     501ns     217ns  1.0230us  cuDeviceGet
                   0.00%  1.8500us         3     616ns     279ns  1.1660us  cuDriverGetVersion
                   0.00%  1.5700us         5     314ns     185ns     667ns  cudaGetLastError
```

ECE408/CS483 Group Project Report

Group Name: airport

Group Member: Pei Liu (NetID:peiliu3, UIN: 673753078)

Zhiyu Zhu (NetID:zzhu24, UIN: 672839294)

Ye Zhang (yezhang4, UIN: 652771627)

MileStone 4

Our approach to optimizations:

1. Sweeping various parameters to find best values

We sweeped number of coaersing instructions is to let each thread process different amount of independent calculations at the same time. Each thread process independent convolutional computations at the same time by reducing the size of the block. Each step in the parameter sweep will need to solve a partially parallelizable problem and each thread process independent convolution computations at the same time by reducing the size of the block. Under this condition, the sweeping improve the performance by processing more than one computations at the same time inside each of the threads.

```
Op Time: 0.032798
Op Time: 0.077820
Correctness: 0.8171 Model: ece408
==249== Profiling application: python m4.1.py 10000
==249== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:   61.11%  110.58ms         2  55.288ms  32.772ms  77.805ms  mxnet::op::forward_kernel(float
                   19.01%  34.397ms        20  1.7198ms  1.0880us  32.376ms  [CUDA memcpy HtoD]
                    9.16%  16.575ms         2  8.2874ms  2.6201ms  13.955ms  void mshadow::cuda::MapPlanLarg
pr::Plan<mshadow::expr::BinaryMapExp<mshadow::op::mul, mshadow::expr::ScalarExp<float>, mshadow::Tensor<msha
                    4.10%  7.4100ms         2  3.7050ms  24.543us  7.3855ms  void cudnn::detail::activation_
::activation_fw_4d_kernel<float, float, int=128, int=1, int=4, cudnn::detail::tanh_func<float>>, cudnnTensor
                    3.79%  6.8648ms         1  6.8648ms  6.8648ms  6.8648ms  volta_sgemm_128x128_tn
                    2.43%  4.4007ms         1  4.4007ms  4.4007ms  4.4007ms  void cudnn::detail::pooling_fw_
nst *, cudnn::detail::pooling_fw_4d_kernel<float, float, cudnn::detail::maxpooling_func<float, cudnnNanPropa
)
                    0.28%  511.39us         1  511.39us  511.39us  511.39us  void mshadow::cuda::MapPlanLarg
pr::Plan<mshadow::expr::ScalarExp<float>, float>>(mshadow::gpu, unsigned int, mshadow::Shape<int=2>, int=2,
                    0.04%  68.608us         1  68.608us  68.608us  68.608us  void mshadow::cuda::SoftmaxKern
dow::gpu, int=2, float>, float>>(mshadow::gpu, int=2, unsigned int)
                    0.03%  59.743us        13  4.5950us  1.1840us  21.984us  void mshadow::cuda::MapPlanKern
w::expr::ScalarExp<float>, float>>(mshadow::gpu, unsigned int, mshadow::Shape<int=2>, int=2)
                    0.02%  30.944us         1  30.944us  30.944us  30.944us  volta_sgemm_32x32_sliced1x4_tn
                    0.01%  23.936us         2  11.968us  2.3040us  21.632us  void mshadow::cuda::MapPlanKern
w::expr::Broadcast1DExp<mshadow::Tensor<mshadow::gpu, int=1, float>, float, int=2, int=1>, float>>(mshadow::
                    0.00%  8.6070us         8  1.0750us     960ns  1.3430us  [CUDA memset]
                    0.00%  5.4720us         1  5.4720us  5.4720us  5.4720us  [CUDA memcpy DtoH]
                    0.00%  4.6720us         1  4.6720us  4.6720us  4.6720us  void mshadow::cuda::MapPlanKern
w::expr::ReduceWithAxisExp<mshadow::red::maximum, mshadow::Tensor<mshadow::gpu, int=3, float>, float, int=3,
      API calls:   38.30%  2.72886s        22  124.04ms  12.458us  1.43744s  cudaStreamCreateWithFlags
                   35.45%  2.52604s        22  114.8
```

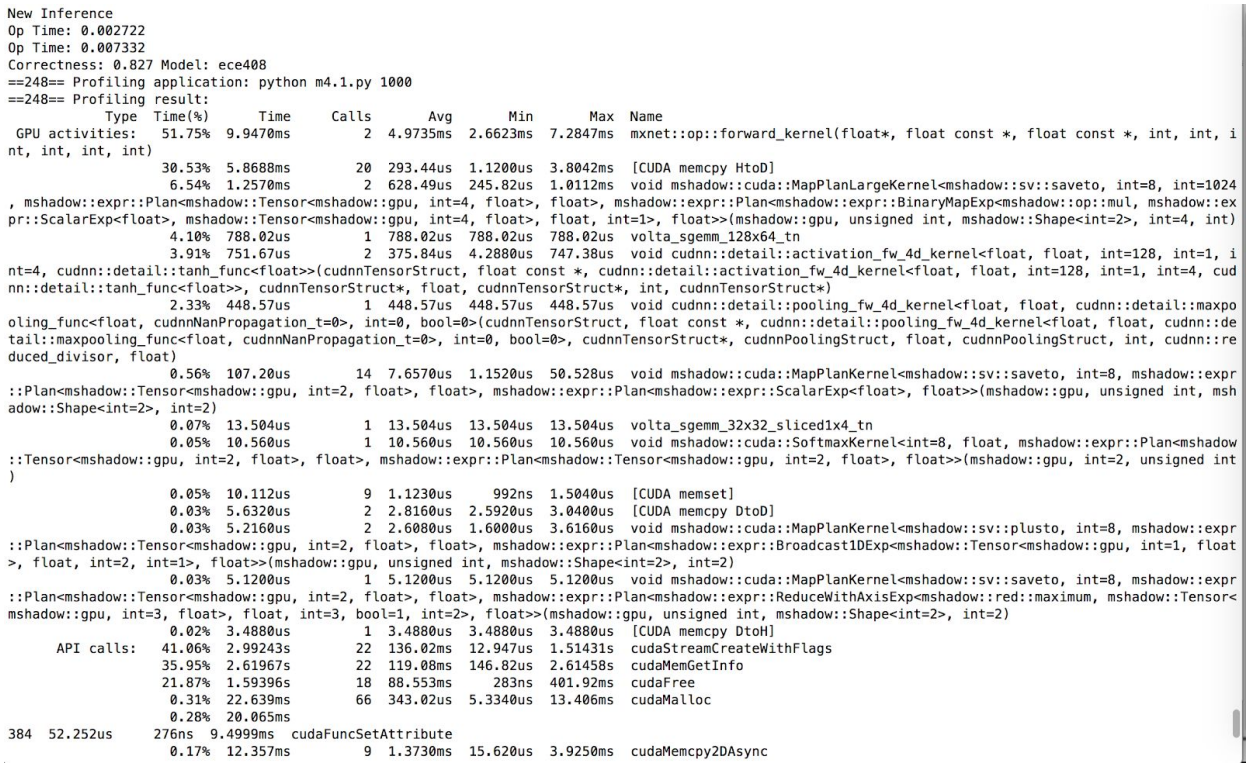
[pic 4.1: The block_size is reduced from 32 to 16: each thread processes 4 computations]

```
New Inference
Op Time: 0.018363
Op Time: 0.045315
Correctness: 0.8171 Model: ece408
==248== Profiling application: python m4.1.py 10000
==248== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:   47.59%  63.638ms         2  31.819ms  18.339ms  45.299ms  mxnet::op::forward_kernel(float*, fl
                   25.52%  34.134ms        20  1.7067ms  1.0880us  32.180ms  [CUDA memcpy HtoD]
                   12.36%  16.527ms         2  8.2637ms  2.6013ms  13.926ms  void mshadow::cuda::MapPlanLargeKern
pr::Plan<mshadow::expr::BinaryMapExp<mshadow::op::mul, mshadow::expr::ScalarExp<float>, mshadow::Tensor<mshadow::
                    5.55%  7.4182ms         2  3.7091ms  24.544us  7.3937ms  void cudnn::detail::activation_fw_4d
::activation_fw_4d_kernel<float, float, int=128, int=1, int=4, cudnn::detail::tanh_func<float>>, cudnnTensorStruc
                    5.16%  6.9015ms         1  6.9015ms  6.9015ms  6.9015ms  volta_sgemm_128x128_tn
                    3.29%  4.4022ms         1  4.4022ms  4.4022ms  4.4022ms  void cudnn::detail::pooling_fw_4d_ke
nst *, cudnn::detail::pooling_fw_4d_kernel<float, float, cudnn::detail::maxpooling_func<float, cudnnNanPropagatic
)
                    0.38%  506.94us         1  506.94us  506.94us  506.94us  void mshadow::cuda::MapPlanLargeKern
pr::Plan<mshadow::expr::ScalarExp<float>, float>>(mshadow::gpu, unsigned int, mshadow::Shape<int=2>, int=2, int)
                    0.05%  69.087us         1  69.087us  69.087us  69.087us  void mshadow::cuda::SoftmaxKernel<in
dow::gpu, int=2, float>, float>>(mshadow::gpu, int=2, unsigned int)
                    0.04%  59.136us        13  4.5480us  1.1840us  21.600us  void mshadow::cuda::MapPlanKernel<ms
w::expr::ScalarExp<float>, float>>(mshadow::gpu, unsigned int, mshadow::Shape<int=2>, int=2)
                    0.02%  29.632us         1  29.632us  29.632us  29.632us  volta_sgemm_32x32_sliced1x4_tn
                    0.02%  24.864us         2  12.432us  2.3360us  22.528us  void mshadow::cuda::MapPlanKernel<ms
w::expr::Broadcast1DExp<mshadow::Tensor<mshadow::gpu, int=1, float>, float, int=2, int=1>, float>>(mshadow::gpu,
                    0.01%  8.3200us         8  1.0400us     992ns  1.3440us  [CUDA memset]
                    0.00%  5.5360us         1  5.5360us  5.5360us  5.5360us  [CUDA memcpy DtoH]
                    0.00%  4.7680us         1  4.7680us  4.7680us  4.7680us  void mshadow::cuda::MapPlanKernel<ms
w::expr::ReduceWithAxisExp<mshadow::red::maximum, mshadow::Tensor<mshadow::gpu, int=3, float>, float, int=3, bool
      API calls:   40.09%  2.88163s        22  130.98ms  13.478us  1.44006s  cudaStreamCreateWithFlags
                   35.87%  2.57853s        22  117.2
```

[pic 4.1: The block_size is reduced from 32 to 8: each thread processes 16 computations]

2. Weight matrix (kernel values) in constant memory

We uploaded the filter bank into the constant memory, which is best choice to accelerate the accesses to the filter bank (mask array). First, the filter bank is small enough to fit in the constant memory, and the contents of the mask array are constant throughout the execution of the kernel. Moreover, all threads need to access the mask in the same order. By using the constant memory, the operation time of the program was reduced by over 20 ms.



```
New Inference
Op Time: 0.002722
Op Time: 0.007332
Correctness: 0.827 Model: ece408
==248== Profiling application: python m4.1.py 1000
==248== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:   51.75%  9.9470ms         2  4.9735ms  2.6623ms  7.2847ms  mxnet::op::forward_kernel(float*, float const *, float const *, int, int, i
nt, int, int, int)
                  30.53%  5.8688ms        20  293.44us  1.1200us  3.8042ms  [CUDA memcpy HtoD]
                   6.54%  1.2570ms         2  628.49us  245.82us  1.0112ms  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::ex
pr::ScalarExp<float>, mshadow::Tensor<mshadow::gpu, int=4, float>, float, int=1>, float>>(mshadow::gpu, unsigned int, mshadow::Shape<int=2>, int=4, int)
                   4.10%  788.02us         1  788.02us  788.02us  788.02us  volta_sgemm_128x64_tn
                   3.91%  751.67us         2  375.84us  4.2880us  747.38us  void cudnn::detail::activation_fw_4d_kernel<float, float, int=128, int=1, i
nt=4, cudnn::detail::tanh_func<float>>(cudnnTensorStruct, float const *, cudnn::detail::activation_fw_4d_kernel<float, float, int=128, int=1, int=4, cud
nn::detail::tanh_func<float>>, cudnnTensorStruct*, float, cudnnTensorStruct*, int, cudnnTensorStruct*)
                   2.33%  448.57us         1  448.57us  448.57us  448.57us  void cudnn::detail::pooling_fw_4d_kernel<float, float, cudnn::detail::maxpo
oling_func<float, cudnnNanPropagation_t=0>, int=0, bool=0>(cudnnTensorStruct, float const *, cudnn::detail::pooling_fw_4d_kernel<float, float, cudnn::de
tail::maxpooling_func<float, cudnnNanPropagation_t=0>, int=0, bool=0>, cudnnTensorStruct*, cudnnPoolingStruct, float, cudnnPoolingStruct, int, cudnn::re
duced_divisor, float)
                   0.56%  107.20us        14  7.6570us  1.1520us  50.528us  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, msh
adow::Shape<int=2>, int=2)
                   0.07%  13.504us         1  13.504us  13.504us  13.504us  volta_sgemm_32x32_sliced1x4_tn
                   0.05%  10.560us         1  10.560us  10.560us  10.560us  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
)
                   0.05%  10.112us         9  1.1230us     992ns  1.5040us  [CUDA memset]
                   0.03%  5.6320us         2  2.8160us  2.5920us  3.0400us  [CUDA memcpy DtoD]
                   0.03%  5.2160us         2  2.6080us  1.6000us  3.6160us  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)
                   0.03%  5.1200us         1  5.1200us  5.1200us  5.1200us  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)
                   0.02%  3.4880us         1  3.4880us  3.4880us  3.4880us  [CUDA memcpy DtoH]
      API calls:   41.06%  2.99243s        22  136.02ms  12.947us  1.51431s  cudaStreamCreateWithFlags
                   35.95%  2.61967s        22  119.08ms  146.82us  2.61458s  cudaMemGetInfo
                   21.87%  1.59396s        18  88.553ms     283ns  401.92ms  cudaFree
                    0.31%  22.639ms        66  343.02us  5.3340us  13.406ms  cudaMalloc
                    0.28%  20.065ms
384  52.252us     276ns  9.4999ms  cudaFuncSetAttribute
                    0.17%  12.357ms         9  1.3730ms  15.620us  3.9250ms  cudaMemcpy2DAsync
```

3. Unrolling

The goal of loop unrolling is to increase a program's speed by reducing or eliminating instructions that control the loop which means "end of loop" tests of incremental boundary checking at the end each iteration and therefore reduce branch penalties as well as hiding latencies including the delay in reading data from memory. To eliminate this computational overhead, loops can be re-written as a repeated sequence of similar independent statements. However, this optimization method did not improve the performance since there are dependencies on increment of accuracy variables.

```
New Inference
Op Time: 0.155526
Op Time: 0.566826
Correctness: 0.8171 Model: ece408
==248== Profiling application: python m4.1.py 10000
==248== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:   91.25%  722.23ms         2  361.11ms  155.47ms  566.76ms  mxnet::op::forward_kernel(float*, float const *, float const *, int, int, int, int, int, int)
                    4.50%  35.648ms        20  1.7824ms  1.0880us  33.665ms  [CUDA memcpy HtoD]
                    1.83%  14.517ms         2  7.2587ms  2.5239ms  11.993ms  void mshadow::cuda::MapPlanLargeKernel<mshadow::sv::saveto, int=8, int=1024, mshadow::expr::Plan<msha
dow::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)
                    0.92%  7.2958ms         2  3.6479ms  24.320us  7.2715ms  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*, f
loat, cudnnTensorStruct*, int, cudnnTensorStruct*)
                    0.86%  6.7885ms         1  6.7885ms  6.7885ms  6.7885ms  volta_sgemm_128x128_tn
                    0.55%  4.3810ms         1  4.3810ms  4.3810ms  4.3810ms  void cudnn::detail::pooling_fw_4d_kernel<float, float, cudnn::detail::maxpooling_func<float, cudnnNan
Propagation_t=0>, int=0, bool=0>(cudnnTensorStruct, float const *, cudnn::detail::pooling_fw_4d_kernel<float, float, cudnn::detail::maxpooling_func<float, cudnnNanPropagation_t=0
>, int=0, bool=0>, cudnnTensorStruct*, cudnnPoolingStruct, float, cudnnPoolingStruct, int, cudnn::reduced_divisor, float)
                    0.06%  459.77us         1  459.77us  459.77us  459.77us  void mshadow::cuda::MapPlanLargeKernel<mshadow::sv::saveto, int=8, int=1024, mshadow::expr::Plan<msha
dow::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, int)
                    0.01%  68.288us         1  68.288us  68.288us  68.288us  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)
                    0.01%  57.632us        13  4.4330us  1.1840us  21.280us  void mshadow::cuda::MapPlanKernel<mshadow::sv::saveto, int=8, mshadow::expr::Plan<mshadow::Tensor<msh
adow::gpu, int=2, float>, float>, mshadow::expr::Plan<mshadow::expr::ScalarExp<float>, float>>(mshadow::gpu, unsigned int, mshadow::Shape<int=2>, int=2)
                    0.00%  29.856us         1  29.856us  29.856us  29.856us  volta_sgemm_32x32_sliced1x4_tn
                    0.00%  24.032us         2  12.016us  2.3360us  21.696us  void mshadow::cuda::MapPlanKernel<mshadow::sv::plusto, int=8, mshadow::expr::Plan<mshadow::Tensor<msh
adow::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, unsig
ned int, mshadow::Shape<int=2>, int=2)
                    0.00%  8.2880us         8  1.0360us     960ns  1.3440us  [CUDA memset]
                    0.00%  5.5040us         1  5.5040us  5.5040us  5.5040us  [CUDA memcpy DtoH]
                    0.00%  5.4400us         2  2.7200us  2.4640us  2.9760us  [CUDA memcpy DtoD]
                    0.00%  4.6080us         1  4.6080us  4.6080us  4.6080us  void mshadow::cuda::MapPlanKernel<mshadow::sv::saveto, int=8, mshadow::expr::Plan<mshadow::Tensor<msh
adow::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, i
nt=2>, float>>(mshadow::gpu, unsigned int, mshadow::Shape<int=2>, int=2)
      API calls:   58.41%  7.48674s        22  340.31ms  66
.486us  7.47337s  cudaMemGetInfo
                   22.07%  2.82946s        22  128.61ms  13.044us  1.45382s  cudaStreamCreateWithFlags
                   12.25%  1.56982s        18  87.212ms     295ns  421.80ms  cudaFree
                    5.75%  736.78ms         6  122.80ms  3.0930us  566.77ms  cudaDeviceSynchronize
                    0.56%  72.137ms         9  8.0152ms  23.998us  33.833ms  cudaMemcpy2DAsync
                    0.45%  57.459ms       384  149.63us     272ns  24.664ms  cudaFuncSetAttribute
                    0.14%  18.468ms        29  636.82us  3.4290us  11.033ms  cudaStreamSynchronize
                    0.13%  17.244ms       216  79.831us     321ns  11.201ms  cudaEventCreateWithFlags
                    0.13%  16.440ms        66  249.09us  4.5130us  4.5434ms  cudaMalloc
                    0.04%  4.5849ms         4  1.1462ms  413.42us  1.6973ms  cudaGetDeviceProperties
                    0.02%  2.6793ms         6  446.56us     716ns  2.6614ms  cudaEventCreate
                    0.02%  2.2537ms       375  6.0090us     100ns  322.41us  cuDeviceGetAttribute
                    0.01%  731.94us         2  365.97us  46.185us  685.76us  cudaHostAlloc
                    0.00%  638.76us         4  159.69us  96.237us  274.01us  cuDeviceTotalMem
```

ECE408/CS483 Group Project Report

Group Name: airport
Group Member:
Pei Liu (NetID:peiliu3, UIN: 673753078)
Zhiyu Zhu (NetID:zzhu24, UIN: 672839294)
Ye Zhang (yezhang4, UIN: 652771627)

Final Submission

Opportunity identification, justification, and effect

In milestone 2, we constructed a CPU implementation of Convolution Neural Network, which has 6 loops for each input dimension. Then we identified several opportunities for speeding up the process using parallel programming.

Optimization 1: Thread Coarsening

Thread coarsening was identified as an optimization opportunity. According to ECE 508 lecture 3 slides**,** thread coarsening reduces the redundant work done by parallel execution by merging multiple threads into one.

Thread Coarsening can merge multiple threads and perform the redundant work once and save the result into registers. According to the runtime, we found that if we reduce thread by 4*4 it gives the most effective result and the performance was largely improved. The performance is shown in figure 1.

```
New Inference
Op Time: 0.054829
Op Time: 0.080684
Correctness: 0.8171 Model: ece408
==254== Profiling application: python final.py 10000
==254== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:   65.65%  135.34ms         2  67.672ms  54.708ms  80.636ms  mxnet::op::forward_kernel(float*, float const *,
 float const *, int, int, int, int, int, int)
                  16.63%  34.293ms        20  1.7147ms  1.1200us  32.597ms  [CUDA memcpy HtoD]
                   7.98%  16.443ms         2  8.2217ms  2.6204ms  13.823ms  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<msh
adow::expr::BinaryMapExp<mshadow::op::mul, mshadow::expr::ScalarExp<float>, mshadow::Tensor<mshadow::gpu, int=4, float>, floa
t, int=1>, float>>(mshadow::gpu, unsigned int, mshadow::Shape<int=2>, int=4, int)
                   3.66%  7.5454ms         1  7.5454ms  7.5454ms  7.5454ms  volta_sgemm_128x128_tn
                   3.60%  7.4137ms         2  3.7068ms  24.864us  7.3888ms  void cudnn::detail::activation_fw_4d_kernel<floa
t, 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, cudnnTensorSt
ruct*, int, cudnnTensorStruct*)
                   2.14%  4.4037ms         1  4.4037ms  4.4037ms  4.4037ms  void cudnn::detail::pooling_fw_4d_kernel<float,
float, cudnn::detail::maxpooling_func<float, cudnnNanPropagation_t=0>, int=0, bool=0>(cudnnTensorStruct, float const *, cudnn
::detail::pooling_fw_4d_kernel<float, float, cudnn::detail::maxpooling_func<float, cudnnNanPropagation_t=0>, int=0, bool=0>,
cudnnTensorStruct*, cudnnPoolingStruct, float, cudnnPoolingStruct, int, cudnn::reduced_divisor, float)
                   0.25%  510.36us         1  510.36us  510.36us  510.36us  void mshadow::cuda::MapPlanLargeKernel<mshadow::
sv::saveto, int=8, int=1024, mshadow::expr::Plan<mshadow::Tensor<mshadow::gpu, int=2, float>, float>, mshadow::expr::Plan<msh
adow::expr::ScalarExp<float>, float>>(mshadow::gpu, unsigned int, mshadow::Shape<int=2>, int=2, int)
                   0.04%  75.263us         1  75.263us  75.263us  75.263us  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, in
t=2, float>, float>>(mshadow::gpu, int=2, unsigned int)
                   0.03%  58.624us        13  4.5090us  1.1840us  21.760us  void mshadow::cuda::MapPlanKernel<mshadow::sv::s
aveto, int=8, mshadow::expr::Plan<mshadow::Tensor<mshadow::gpu, int=2, float>, float>, mshadow::expr::Plan<mshadow::expr::Sca
larExp<float>, float>>(mshadow::gpu, unsigned int, mshadow::Shape<int=2>, int=2)
                   0.02%  32.607us         1  32.607us  32.607us  32.607us  volta_sgemm_32x32_sliced1x4_tn
                   0.01%  24.255us         2  12.127us  2.5600us  21.695us  void mshadow::cuda::MapPlanKernel<mshadow::sv::p
lusto, int=8, mshadow::expr::Plan<mshadow::Tensor<mshadow::gpu, int=2, float>, float>, mshadow::expr::Plan<mshadow::expr::Bro
adcast1DExp<mshadow::Tensor<mshadow::gpu, int=1, float>, float, int=2, int=1>, float>>(mshadow::gpu, unsigned int, mshadow::S
hape<int=2>, int=2)
                   0.00%  8.3840us         8  1.0480us     992ns  1.3760us  [CUDA memset]
                   0.00%  6.1760us         2  3.0880us  2.9440us  3.2320us  [CUDA memcpy DtoD]
                   0.00%  5.7600us         1  5.7600us  5.7600us  5.7600us  [CUDA memcpy DtoH]
                   0.00%  5.1840us         1  5.1840us  5.1840us  5.1840us  void mshadow::cuda::MapPlanKernel<mshadow::sv::s
aveto, int=8, mshadow::expr::Plan<mshadow::Tensor<mshadow::gpu, int=2, float>, float>, mshadow::expr::Plan<mshadow::expr::Red
uceWithAxisExp<mshadow::red::maximum, mshadow::Tensor<mshadow::gpu, int=3, float>, float, int=3, bool=1, int=2>, float>>(msha
dow::gpu, unsigned int, mshadow::Shape<int=2>, int=2)
      API calls:   47.68%  4.45063s        22  202.30ms  86
.501us  4.44725s  cudaMemGetInfo
```

[Figure 1: Runtime report for thread coarsening]

Optimization 2 Memory Coalescing

Since memory is retrieved in large blocks from the ram, memory coalescing was applied to make sure threads try to access memory that is nearby, in an attempt to reduce overhead for accessing global memory. This was made possible by making the threaded problem reading in an orderly manner to reduce memory retrieval, rather than jumping all over, making the processing unit wait for memory requests to be fulfilled

We implemented the most inner loop to access all continuous memory horizontally. Since each time multiple consecutive memory is loaded into the cache and the cache has much faster access speed than directly reading from global memory.

The result was remarkable. Figure 2 below shows a performance analysis of the new runtime after memory coalescing is added. The original implementation without

consideration for memory coalescing had a run time of over 130ms, and after memory coalescing was introduced, the runtime was reduced by more than 50%, and the new runtime was about 60ms. In particular, the time spent in forward_kernel was reduced from 67.672ms to 27.639, proving the effectiveness of memory coalescing.

```
New Inference
Op Time: 0.016340
Op Time: 0.039045
Correctness: 0.8171 Model: ece408
==252== Profiling application: python final.py 10000
==252== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:   43.08%  55.278ms         2  27.639ms  16.280ms  38.997ms  mxnet::op::forward_kernel(float*, float const *,
 float const *, int, int, int, int, int, int)
                   28.93%  37.121ms        20  1.8561ms  1.1200us  35.069ms  [CUDA memcpy HtoD]
                   12.85%  16.482ms         2  8.2412ms  2.6424ms  13.840ms  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<msh
adow::expr::BinaryMapExp<mshadow::op::mul, mshadow::expr::ScalarExp<float>, mshadow::Tensor<mshadow::gpu, int=4, float>, floa
t, int=1>, float>>(mshadow::gpu, unsigned int, mshadow::Shape<int=2>, int=4, int)
                    5.76%  7.3926ms         2  3.6963ms  25.407us  7.3672ms  void cudnn::detail::activation_fw_4d_kernel<floa
t, 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, cudnnTensorSt
ruct*, int, cudnnTensorStruct*)
                    5.38%  6.9028ms         1  6.9028ms  6.9028ms  6.9028ms  volta_sgemm_128x128_tn
                    3.43%  4.4072ms         1  4.4072ms  4.4072ms  4.4072ms  void cudnn::detail::pooling_fw_4d_kernel<float,
float, cudnn::detail::maxpooling_func<float, cudnnNanPropagation_t=0>, int=0, bool=0>(cudnnTensorStruct, float const *, cudnn
::detail::pooling_fw_4d_kernel<float, float, cudnn::detail::maxpooling_func<float, cudnnNanPropagation_t=0>, int=0, bool=0>,
cudnnTensorStruct*, cudnnPoolingStruct, float, cudnnPoolingStruct, int, cudnn::reduced_divisor, float)
                    0.41%  523.13us         1  523.13us  523.13us  523.13us  void mshadow::cuda::MapPlanLargeKernel<mshadow::
sv::saveto, int=8, int=1024, mshadow::expr::Plan<mshadow::Tensor<mshadow::gpu, int=2, float>, float>, mshadow::expr::Plan<msh
adow::expr::ScalarExp<float>, float>>(mshadow::gpu, unsigned int, mshadow::Shape<int=2>, int=2, int)
                    0.05%  69.087us         1  69.087us  69.087us  69.087us  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, in
t=2, float>, float>>(mshadow::gpu, int=2, unsigned int)
                    0.05%  57.887us        13  4.4520us  1.1840us  21.407us  void mshadow::cuda::MapPlanKernel<mshadow::sv::s
aveto, int=8, mshadow::expr::Plan<mshadow::Tensor<mshadow::gpu, int=2, float>, float>, mshadow::expr::Plan<mshadow::expr::Sca
larExp<float>, float>>(mshadow::gpu, unsigned int, mshadow::Shape<int=2>, int=2)
                    0.02%  30.400us         1  30.400us  30.400us  30.400us  volta_sgemm_32x32_sliced1x4_tn
                    0.02%  23.840us         2  11.920us  2.3040us  21.536us  void mshadow::cuda::MapPlanKernel<mshadow::sv::p
lusto, int=8, mshadow::expr::Plan<mshadow::Tensor<mshadow::gpu, int=2, float>, float>, mshadow::expr::Plan<mshadow::expr::Bro
adcast1DExp<mshadow::Tensor<mshadow::gpu, int=1, float>, float, int=2, int=1>, float>>(mshadow::gpu, unsigned int, mshadow::S
hape<int=2>, int=2)
                    0.01%  8.8320us         8  1.1040us     992ns  1.3760us  [CUDA memset]
                    0.01%  7.5520us         1  7.5520us  7.5520us  7.5520us  [CUDA memcpy DtoH]
                    0.00%  5.5680us         2  2.7840us  2.4320us  3.1360us  [CUDA memcpy DtoD]
                    0.00%  4.4800us         1  4.4800us  4.4800us  4.4800us  void mshadow::cuda::MapPlanKernel<mshadow::sv::s
aveto, int=8, mshadow::expr::Plan<mshadow::Tensor<mshadow::gpu, int=2, float>, float>, mshadow::expr::Plan<mshadow::expr::Red
uceWithAxisExp<mshadow::red::maximum, mshadow::Tensor<mshadow::gpu, int=3, float>, float, int=3, bool=1, int=2>, float>>(msha
dow::gpu, unsigned int, mshadow::Shape<int=2>, int=2)
      API calls:   51.46%  5.02629s        22  228.47ms  66
```

[Figure 2: Runtime report for memory coalescing]

Optimization 3: Constant Memory

Since for all threads of a half-warp, reading from a register as long as all threads read the same address has almost the same speed as reading from cache, if the data that will never be changed is loaded into the cache, the speed will be linearly improved according to the number of access to the memory. Motivated by this, constant memory was implemented to speed up the convolution neural network.

We implemented the constant memory after we finished thread coarsening and memory coalescing with 4*4 and the constant memory improved the runtime obviously. Figure 3

shows the improvement of runtime, which can be compared with figure 2. The first layer was improved by 0.4 ms and the second layer was improved by 2 ms. The improvement for the second layer kernel function is more significant than the first layer because for the first layer, each thread access C(1)*K(7)*K(7) and second layer access C(2)*K(7)*K(7) times to the constant memory. Since there is a payoff to read from host memory and write to the constant memory, the improvement will be more obvious on the kernel that has a larger number of accesses to the constant memory.

```
New Inference
Op Time: 0.016770
Op Time: 0.041047
Correctness: 0.8171 Model: ece408
==254== Profiling application: python final.py 10000
==254== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:   45.43%  57.753ms         2  28.877ms  16.731ms  41.022ms  mxnet::op::forward_kernel(float*, float const *,
 float const *, int, int, int, int, int, int)
                   28.32%  35.996ms        20  1.7998ms  1.0560us  33.901ms  [CUDA memcpy HtoD]
                   11.23%  14.279ms         2  7.1395ms  2.5105ms  11.769ms  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<msh
adow::expr::BinaryMapExp<mshadow::op::mul, mshadow::expr::ScalarExp<float>, mshadow::Tensor<mshadow::gpu, int=4, float>, floa
t, int=1>, float>>(mshadow::gpu, unsigned int, mshadow::Shape<int=2>, int=4, int)
                    5.74%  7.2977ms         2  3.6488ms  24.352us  7.2733ms  void cudnn::detail::activation_fw_4d_kernel<floa
t, 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, cudnnTensorSt
ruct*, int, cudnnTensorStruct*)
                    5.34%  6.7848ms         1  6.7848ms  6.7848ms  6.7848ms  volta_sgemm_128x128_tn
                    3.45%  4.3818ms         1  4.3818ms  4.3818ms  4.3818ms  void cudnn::detail::pooling_fw_4d_kernel<float,
float, cudnn::detail::maxpooling_func<float, cudnnNanPropagation_t=0>, int=0, bool=0>(cudnnTensorStruct, float const *, cudnn
::detail::pooling_fw_4d_kernel<float, float, cudnn::detail::maxpooling_func<float, cudnnNanPropagation_t=0>, int=0, bool=0>,
cudnnTensorStruct*, cudnnPoolingStruct, float, cudnnPoolingStruct, int, cudnn::reduced_divisor, float)
                    0.34%  437.50us         1  437.50us  437.50us  437.50us  void mshadow::cuda::MapPlanLargeKernel<mshadow::
sv::saveto, int=8, int=1024, mshadow::expr::Plan<mshadow::Tensor<mshadow::gpu, int=2, float>, float>, mshadow::expr::Plan<msh
adow::expr::ScalarExp<float>, float>>(mshadow::gpu, unsigned int, mshadow::Shape<int=2>, int=2, int)
                    0.05%  67.839us         1  67.839us  67.839us  67.839us  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, in
t=2, float>, float>>(mshadow::gpu, int=2, unsigned int)
                    0.04%  53.280us        13  4.0980us  1.0560us  19.392us  void mshadow::cuda::MapPlanKernel<mshadow::sv::s
aveto, int=8, mshadow::expr::Plan<mshadow::Tensor<mshadow::gpu, int=2, float>, float>, mshadow::expr::Plan<mshadow::expr::Sca
larExp<float>, float>>(mshadow::gpu, unsigned int, mshadow::Shape<int=2>, int=2)
                    0.02%  30.207us         1  30.207us  30.207us  30.207us  volta_sgemm_32x32_sliced1x4_tn
                    0.02%  26.624us         2  13.312us  2.3680us  24.256us  void mshadow::cuda::MapPlanKernel<mshadow::sv::p
lusto, int=8, mshadow::expr::Plan<mshadow::Tensor<mshadow::gpu, int=2, float>, float>, mshadow::expr::Plan<mshadow::expr::Bro
adcast1DExp<mshadow::Tensor<mshadow::gpu, int=1, float>, float, int=2, int=1>, float>>(mshadow::gpu, unsigned int, mshadow::S
hape<int=2>, int=2)
                    0.01%  8.7360us         8  1.0920us     928ns  1.4400us  [CUDA memset]
                    0.00%  5.5030us         1  5.5030us  5.5030us  5.5030us  [CUDA memcpy DtoH]
                    0.00%  4.4160us         1  4.4160us  4.4160us  4.4160us  void mshadow::cuda::MapPlanKernel<mshadow::sv::s
aveto, int=8, mshadow::expr::Plan<mshadow::Tensor<mshadow::gpu, int=2, float>, float>, mshadow::expr::Plan<mshadow::expr::Red
uceWithAxisExp<mshadow::red::maximum, mshadow::Tensor<mshadow::gpu, int=3, float>, float, int=3, bool=1, int=2>, float>>(msha
dow::gpu, unsigned int, mshadow::Shape<int=2>, int=2)
API calls:   39.64%  2.96588s        22  134.81ms  11.767us  1.53231s  cudaStreamCreateWithFlags
                   35.82%  2.67952s        22  121.80ms  69.512us  2.67743s  cudaMemGetInfo
```

[Figure 3: Runtime report for constant memory]

Optimization 4: Shared Memory

According to the access speed, shared memory is much more efficient than the access to the global memory. Since the shared memory doesn't need to be reallocated and is shared inside the block, if implemented reasonably, the shared memory will improve the performance of the kernel.

Figure 4 shows the runtime report for the shared memory optimization. Compared with the original implementation shown in figure 3, runtime is not significantly improved. In our implementation, the shared memory slows down the performance of the first layer because the reading from the global memory and writing to shared memory will not have pros when the shared memory is not reused enough. The payoff for implementing and transferring data generated great cost and are not effective for the first layer.

For the second layer, the efficiency stays the same. Although the shared memory is reused, the synchronization of the thread barriers takes many time since all thread has to wait to do the forward CNN after loading to share memory and wait second time before they enter the next C loops. Therefore, the shared memory is not efficient enough generally.

```
New Inference
Op Time: 0.016340
Op Time: 0.039045
Correctness: 0.8171 Model: ece408
==252== Profiling application: python final.py 10000
==252== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:   43.08%  55.278ms         2  27.639ms  16.280ms  38.997ms  mxnet::op::forward_kernel(float*, float const *,
 float const *, int, int, int, int, int, int)
                   28.93%  37.121ms        20  1.8561ms  1.1200us  35.069ms  [CUDA memcpy HtoD]
                   12.85%  16.482ms         2  8.2412ms  2.6424ms  13.840ms  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<msh
adow::expr::BinaryMapExp<mshadow::op::mul, mshadow::expr::ScalarExp<float>, mshadow::Tensor<mshadow::gpu, int=4, float>, floa
t, int=1>, float>>(mshadow::gpu, unsigned int, mshadow::Shape<int=2>, int=4, int)
                    5.76%  7.3926ms         2  3.6963ms  25.407us  7.3672ms  void cudnn::detail::activation_fw_4d_kernel<floa
t, 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, cudnnTensorSt
ruct*, int, cudnnTensorStruct*)
                    5.38%  6.9028ms         1  6.9028ms  6.9028ms  6.9028ms  volta_sgemm_128x128_tn
                    3.43%  4.4072ms         1  4.4072ms  4.4072ms  4.4072ms  void cudnn::detail::pooling_fw_4d_kernel<float,
float, cudnn::detail::maxpooling_func<float, cudnnNanPropagation_t=0>, int=0, bool=0>(cudnnTensorStruct, float const *, cudnn
::detail::pooling_fw_4d_kernel<float, float, cudnn::detail::maxpooling_func<float, cudnnNanPropagation_t=0>, int=0, bool=0>,
cudnnTensorStruct*, cudnnPoolingStruct, float, cudnnPoolingStruct, int, cudnn::reduced_divisor, float)
                    0.41%  523.13us         1  523.13us  523.13us  523.13us  void mshadow::cuda::MapPlanLargeKernel<mshadow::
sv::saveto, int=8, int=1024, mshadow::expr::Plan<mshadow::Tensor<mshadow::gpu, int=2, float>, float>, mshadow::expr::Plan<msh
adow::expr::ScalarExp<float>, float>>(mshadow::gpu, unsigned int, mshadow::Shape<int=2>, int=2, int)
                    0.05%  69.087us         1  69.087us  69.087us  69.087us  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, in
t=2, float>, float>>(mshadow::gpu, int=2, unsigned int)
                    0.05%  57.887us        13  4.4520us  1.1840us  21.407us  void mshadow::cuda::MapPlanKernel<mshadow::sv::s
aveto, int=8, mshadow::expr::Plan<mshadow::Tensor<mshadow::gpu, int=2, float>, float>, mshadow::expr::Plan<mshadow::expr::Sca
larExp<float>, float>>(mshadow::gpu, unsigned int, mshadow::Shape<int=2>, int=2)
                    0.02%  30.400us         1  30.400us  30.400us  30.400us  volta_sgemm_32x32_sliced1x4_tn
                    0.02%  23.840us         2  11.920us  2.3040us  21.536us  void mshadow::cuda::MapPlanKernel<mshadow::sv::p
lusto, int=8, mshadow::expr::Plan<mshadow::Tensor<mshadow::gpu, int=2, float>, float>, mshadow::expr::Plan<mshadow::expr::Bro
adcast1DExp<mshadow::Tensor<mshadow::gpu, int=1, float>, float, int=2, int=1>, float>>(mshadow::gpu, unsigned int, mshadow::S
hape<int=2>, int=2)
                    0.01%  8.8320us         8  1.1040us     992ns  1.3760us  [CUDA memset]
                    0.01%  7.5520us         1  7.5520us  7.5520us  7.5520us  [CUDA memcpy DtoH]
                    0.00%  5.5680us         2  2.7840us  2.4320us  3.1360us  [CUDA memcpy DtoD]
                    0.00%  4.4800us         1  4.4800us  4.4800us  4.4800us  void mshadow::cuda::MapPlanKernel<mshadow::sv::s
aveto, int=8, mshadow::expr::Plan<mshadow::Tensor<mshadow::gpu, int=2, float>, float>, mshadow::expr::Plan<mshadow::expr::Red
uceWithAxisExp<mshadow::red::maximum, mshadow::Tensor<mshadow::gpu, int=3, float>, float, int=3, bool=1, int=2>, float>>(msha
dow::gpu, unsigned int, mshadow::Shape<int=2>, int=2)
      API calls:   51.46%  5.02629s        22  228.47ms  66
```

[Figure 4: Runtime Report for Shared Memory]

Optimization 5: Loop Unrolling

The goal of loop unrolling is to increase a program's speed by reducing or eliminating instructions that control the loop which means "end of loop" tests of incremental boundary checking at the end each iteration and therefore reduce branch penalties as well as hiding latencies including the delay in reading data from memory. To eliminate this computational overhead, loops can be re-written as a repeated sequence of similar independent statements. Therefore, loop unrolling was believed to be fruitful to reduce runtime. To test the effect of loop unrolling, we implemented a separate kernel. Figure 5 below shows the runtime in the testing kernel without loop unrolling. Unfortunately, the performance was not improved as expected. There are several potential reasons that could cause such slowdown. First, loop unrolling increases register usages in a single iteration to store local variables that were originally used only once in regular iterative logic, which may reduce performance. Second, the increased program code size is undesirable and could potentially cause an increase in instruction cache misses. Figure 5 and figure 6 compares the performance of loop unrolling implemented in the separate kernel.

```
Op Time: 0.032798
Op Time: 0.077820
Correctness: 0.8171 Model: ece408
==249== Profiling application: python m4.1.py 10000
==249== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:   61.11%  110.58ms         2  55.288ms  32.772ms  77.805ms  mxnet::op::forward_kernel(float
                   19.01%  34.397ms        20  1.7198ms  1.0880us  32.376ms  [CUDA memcpy HtoD]
                    9.16%  16.575ms         2  8.2874ms  2.6201ms  13.955ms  void mshadow::cuda::MapPlanLarg
pr::Plan<mshadow::expr::BinaryMapExp<mshadow::op::mul, mshadow::expr::ScalarExp<float>, mshadow::Tensor<msha
                    4.10%  7.4100ms         2  3.7050ms  24.543us  7.3855ms  void cudnn::detail::activation_
::activation_fw_4d_kernel<float, float, int=128, int=1, int=4, cudnn::detail::tanh_func<float>>, cudnnTensor
                    3.79%  6.8648ms         1  6.8648ms  6.8648ms  6.8648ms  volta_sgemm_128x128_tn
                    2.43%  4.4007ms         1  4.4007ms  4.4007ms  4.4007ms  void cudnn::detail::pooling_fw_
nst *, cudnn::detail::pooling_fw_4d_kernel<float, float, cudnn::detail::maxpooling_func<float, cudnnNanPropa
)
                    0.28%  511.39us         1  511.39us  511.39us  511.39us  void mshadow::cuda::MapPlanLarg
pr::Plan<mshadow::expr::ScalarExp<float>, float>>(mshadow::gpu, unsigned int, mshadow::Shape<int=2>, int=2,
                    0.04%  68.608us         1  68.608us  68.608us  68.608us  void mshadow::cuda::SoftmaxKern
dow::gpu, int=2, float>, float>>(mshadow::gpu, int=2, unsigned int)
                    0.03%  59.743us        13  4.5950us  1.1840us  21.984us  void mshadow::cuda::MapPlanKern
w::expr::ScalarExp<float>, float>>(mshadow::gpu, unsigned int, mshadow::Shape<int=2>, int=2)
                    0.02%  30.944us         1  30.944us  30.944us  30.944us  volta_sgemm_32x32_sliced1x4_tn
                    0.01%  23.936us         2  11.968us  2.3040us  21.632us  void mshadow::cuda::MapPlanKern
w::expr::Broadcast1DExp<mshadow::Tensor<mshadow::gpu, int=1, float>, float, int=2, int=1>, float>>(mshadow::
                    0.00%  8.6070us         8  1.0750us     960ns  1.3430us  [CUDA memset]
                    0.00%  5.4720us         1  5.4720us  5.4720us  5.4720us  [CUDA memcpy DtoH]
                    0.00%  4.6720us         1  4.6720us  4.6720us  4.6720us  void mshadow::cuda::MapPlanKern
w::expr::ReduceWithAxisExp<mshadow::red::maximum, mshadow::Tensor<mshadow::gpu, int=3, float>, float, int=3,
      API calls:   38.30%  2.72886s        22  124.04ms  12.458us  1.43744s  cudaStreamCreateWithFlags
                   35.45%  2.52604s        22  114.8
```

[Figure 5: Runtime report for the separate kernel without loop unrolling]

```
New Inference
Op Time: 0.153072
Op Time: 0.567877
Correctness: 0.8171 Model: ece408
==258== Profiling application: python final.py 10000
==258== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:   90.74%  720.81ms         2  360.40ms  153.00ms  567.81ms  mxnet::op::forward_kernel(float*, floa
t const *, float const *, int, int, int, int, int, int)
                   4.72%  37.526ms        20  1.8763ms  1.0880us  35.484ms  [CUDA memcpy HtoD]
                   2.09%  16.577ms         2  8.2884ms  2.6342ms  13.943ms  void mshadow::cuda::MapPlanLargeKernel
<mshadow::sv::saveto, int=8, int=1024, mshadow::expr::Plan<mshadow::Tensor<mshadow::gpu, int=4, float>, float>, msh
adow::expr::Plan<mshadow::expr::BinaryMapExp<mshadow::op::mul, mshadow::expr::ScalarExp<float>, mshadow::Tensor<msh
adow::gpu, int=4, float>, float, int=1>, float>>(mshadow::gpu, unsigned int, mshadow::Shape<int=2>, int=4, int)
                   0.94%  7.4533ms         2  3.7266ms  25.888us  7.4274ms  void cudnn::detail::activation_fw_4d_k
ernel<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>>, cudnnTenso
rStruct*, float, cudnnTensorStruct*, int, cudnnTensorStruct*)
                   0.86%  6.8303ms         1  6.8303ms  6.8303ms  6.8303ms  volta_sgemm_128x128_tn
                   0.56%  4.4327ms         1  4.4327ms  4.4327ms  4.4327ms  void cudnn::detail::pooling_fw_4d_kern
el<float, float, cudnn::detail::maxpooling_func<float, cudnnNanPropagation_t=0>, int=0, bool=0>(cudnnTensorStruct,
float const *, cudnn::detail::pooling_fw_4d_kernel<float, float, cudnn::detail::maxpooling_func<float, cudnnNanProp
agation_t=0>, int=0, bool=0>, cudnnTensorStruct*, cudnnPoolingStruct, float, cudnnPoolingStruct, int, cudnn::reduce
d_divisor, float)
                   0.06%  515.39us         1  515.39us  515.39us  515.39us  void mshadow::cuda::MapPlanLargeKernel
<mshadow::sv::saveto, int=8, int=1024, mshadow::expr::Plan<mshadow::Tensor<mshadow::gpu, int=2, float>, float>, msh
adow::expr::Plan<mshadow::expr::ScalarExp<float>, float>>(mshadow::gpu, unsigned int, mshadow::Shape<int=2>, int=2,
 int)
                   0.01%  68.479us         1  68.479us  68.479us  68.479us  void mshadow::cuda::SoftmaxKernel<int=
8, float, mshadow::expr::Plan<mshadow::Tensor<mshadow::gpu, int=2, float>, float>, mshadow::expr::Plan<mshadow::Ten
sor<mshadow::gpu, int=2, float>, float>>(mshadow::gpu, int=2, unsigned int)
                   0.01%  59.264us        13  4.5580us  1.1840us  21.632us  void mshadow::cuda::MapPlanKernel<msha
```

[Figure 6: Runtime report for the separate kernel with loop unrolling]

Optimization 6: **Multilayer kernel**

Since there are two different layers with different size, the multilayer kernels will help reduce redundant end loop conditions and reduce redundant loading. In the first layer with C=1 and shared memory that reduces the performance, the small kernel get rid of the for loop on C or shared memory. For the large layer, the shared memory is implemented and the C loop is necessary. As the result, the total op time with the Multilayer-Kernel optimization is 5 ms faster than that without the Multilayer-Kernel optimization.

```
New Inference
Op Time: 0.016398
Op Time: 0.037425
Correctness: 0.8171 Model: ece408
==253== Profiling application: python final.py 10000
==253== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:   43.46%  53.718ms         2  26.859ms  16.324ms  37.394ms  mxnet::op::forward_kernel_original(float*, float
 const *, int, int, int, int, int, int)
                   29.36%  36.290ms        20  1.8145ms  1.0880us  34.361ms  [CUDA memcpy HtoD]
                   11.68%  14.437ms         2  7.2183ms  2.5101ms  11.927ms  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<msh
adow::expr::BinaryMapExp<mshadow::op::mul, mshadow::expr::ScalarExp<float>, mshadow::Tensor<mshadow::gpu, int=4, float>, floa
t, int=1>, float>>(mshadow::gpu, unsigned int, mshadow::Shape<int=2>, int=4, int)
                    5.87%  7.2514ms         2  3.6257ms  24.576us  7.2268ms  void cudnn::detail::activation_fw_4d_kernel<floa
t, 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, cudnnTensorSt
ruct*, int, cudnnTensorStruct*)
                    5.58%  6.9015ms         1  6.9015ms  6.9015ms  6.9015ms  volta_sgemm_128x128_tn
                    3.52%  4.3547ms         1  4.3547ms  4.3547ms  4.3547ms  void cudnn::detail::pooling_fw_4d_kernel<float,
float, cudnn::detail::maxpooling_func<float, cudnnNanPropagation_t=0>, int=0, bool=0>(cudnnTensorStruct, float const *, cudnn
::detail::pooling_fw_4d_kernel<float, float, cudnn::detail::maxpooling_func<float, cudnnNanPropagation_t=0>, int=0, bool=0>,
cudnnTensorStruct*, cudnnPoolingStruct, float, cudnnPoolingStruct, int, cudnn::reduced_divisor, float)
                    0.36%  448.70us         1  448.70us  448.70us  448.70us  void mshadow::cuda::MapPlanLargeKernel<mshadow::
sv::saveto, int=8, int=1024, mshadow::expr::Plan<mshadow::Tensor<mshadow::gpu, int=2, float>, float>, mshadow::expr::Plan<msh
adow::expr::ScalarExp<float>, float>>(mshadow::gpu, unsigned int, mshadow::Shape<int=2>, int=2, int)
                    0.06%  68.831us         1  68.831us  68.831us  68.831us  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, in
t=2, float>, float>>(mshadow::gpu, int=2, unsigned int)
                    0.05%  57.823us        13  4.4470us  1.2160us  21.375us  void mshadow::cuda::MapPlanKernel<mshadow::sv::s
aveto, int=8, mshadow::expr::Plan<mshadow::Tensor<mshadow::gpu, int=2, float>, float>, mshadow::expr::Plan<mshadow::expr::Sca
larExp<float>, float>>(mshadow::gpu, unsigned int, mshadow::Shape<int=2>, int=2)
                    0.02%  29.663us         1  29.663us  29.663us  29.663us  volta_sgemm_32x32_sliced1x4_tn
                    0.02%  24.160us         2  12.080us  2.2720us  21.888us  void mshadow::cuda::MapPlanKernel<mshadow::sv::p
lusto, int=8, mshadow::expr::Plan<mshadow::Tensor<mshadow::gpu, int=2, float>, float>, mshadow::expr::Plan<mshadow::expr::Bro
adcast1DExp<mshadow::Tensor<mshadow::gpu, int=1, float>, float, int=2, int=1>, float>>(mshadow::gpu, unsigned int, mshadow::S
hape<int=2>, int=2)
                    0.01%  8.7360us         8  1.0920us     992ns  1.3440us  [CUDA memset]
                    0.01%  7.5520us         1  7.5520us  7.5520us  7.5520us  [CUDA memcpy DtoH]
                    0.00%  6.0480us         2  3.0240us  2.7840us  3.2640us  [CUDA memcpy DtoD]
                    0.00%  4.4800us         1  4.4800us  4.4800us  4.4800us  void mshadow::cuda::MapPlanKernel<mshadow::sv::s
aveto, int=8, mshadow::expr::Plan<mshadow::Tensor<mshadow::gpu, int=2, float>, float>, mshadow::expr::Plan<mshadow::expr::Red
uceWithAxisExp<mshadow::red::maximum, mshadow::Tensor<mshadow::gpu, int=3, float>, float, int=3, bool=1, int=2>, float>>(msha
dow::gpu, unsigned int, mshadow::Shape<int=2>, int=2)
```

[Figure 7: Runtime Report with Multilayer Kernel]

Optimization 7: **Unrolling Matrix**

The unrolling Matrix is to unroll all the input feature maps and filter bank to do the matrix multiplication the unrolled input matrix and unrolled weight matrix. Since we can use tile and share memory to improve the performance of matrix multiplication, it can be very efficient. However, the implementation of unrolling matrix does not improve the performance because the size of unrolled matrix depends on the products of all parameters and generates a large payoff on the unrolling kernels.

```
New Inference
Op Time: 0.013539
Op Time: 0.036629
Correctness: 0.8171 Model: ece408
==256== Profiling application: python final.py 10000
==256== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:   30.91%  36.503ms         1  36.503ms  36.503ms  36.503ms  mxnet::op::forward_kernel_original(float*, float
 const *, int, int, int, int, int, int)
                  29.32%  34.633ms        20  1.7316ms  1.0560us  32.632ms  [CUDA memcpy HtoD]
                  12.14%  14.339ms         2  7.1695ms  2.5268ms  11.812ms  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<msh
adow::expr::BinaryMapExp<mshadow::op::mul, mshadow::expr::ScalarExp<float>, mshadow::Tensor<mshadow::gpu, int=4, float>, floa
t, int=1>, float>>(mshadow::gpu, unsigned int, mshadow::Shape<int=2>, int=4, int)
                  11.36%  13.417ms         1  13.417ms  13.417ms  13.417ms  mxnet::op::forward_kernel_first(float*, float co
nst *, int, int, int, int, int, int)
                   6.18%  7.3013ms         2  3.6507ms  24.256us  7.2771ms  void cudnn::detail::activation_fw_4d_kernel<floa
t, 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, cudnnTensorSt
ruct*, int, cudnnTensorStruct*)
                   5.75%  6.7865ms         1  6.7865ms  6.7865ms  6.7865ms  volta_sgemm_128x128_tn
                   3.71%  4.3825ms         1  4.3825ms  4.3825ms  4.3825ms  void cudnn::detail::pooling_fw_4d_kernel<float,
float, cudnn::detail::maxpooling_func<float, cudnnNanPropagation_t=0>, int=0, bool=0>(cudnnTensorStruct, float const *, cudnn
::detail::pooling_fw_4d_kernel<float, float, cudnn::detail::maxpooling_func<float, cudnnNanPropagation_t=0>, int=0, bool=0>,
cudnnTensorStruct*, cudnnPoolingStruct, float, cudnnPoolingStruct, int, cudnn::reduced_divisor, float)
                   0.37%  438.65us         1  438.65us  438.65us  438.65us  void mshadow::cuda::MapPlanLargeKernel<mshadow::
sv::saveto, int=8, int=1024, mshadow::expr::Plan<mshadow::Tensor<mshadow::gpu, int=2, float>, float>, mshadow::expr::Plan<msh
adow::expr::ScalarExp<float>, float>>(mshadow::gpu, unsigned int, mshadow::Shape<int=2>, int=2, int)
                   0.09%  107.14us         2  53.567us  35.136us  71.999us  mxnet::op::unroll_Kernel(int, int, int, int, flo
at*, float*)
                   0.06%  67.936us         1  67.936us  67.936us  67.936us  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, in
t=2, float>, float>>(mshadow::gpu, int=2, unsigned int)
                   0.04%  52.768us        13  4.0590us  1.0560us  19.520us  void mshadow::cuda::MapPlanKernel<mshadow::sv::s
aveto, int=8, mshadow::expr::Plan<mshadow::Tensor<mshadow::gpu, int=2, float>, float>, mshadow::expr::Plan<mshadow::expr::Sca
larExp<float>, float>>(mshadow::gpu, unsigned int, mshadow::Shape<int=2>, int=2)
                   0.03%  30.752us         1  30.752us  30.752us  30.752us  volta_sgemm_32x32_sliced1x4_tn
                   0.02%  25.120us         2  12.560us  2.3040us  22.816us  void mshadow::cuda::MapPlanKernel<mshadow::sv::p
lusto, int=8, mshadow::expr::Plan<mshadow::Tensor<mshadow::gpu, int=2, float>, float>, mshadow::expr::Plan<mshadow::expr::Bro
adcast1DExp<mshadow::Tensor<mshadow::gpu, int=1, float>, float, int=2, int=1>, float>>(mshadow::gpu, unsigned int, mshadow::S
hape<int=2>, int=2)
                   0.01%  8.1600us         8  1.0200us     928ns  1.3120us  [CUDA memset]
                   0.00%  5.7920us         1  5.7920us  5.7920us  5.7920us  [CUDA memcpy DtoH]
                   0.00%  5.6640us         2  2.8320us  2.7840us  2.8800us  [CUDA memcpy DtoD]
```

[Figure 8: Runtime Report with Unrolling]

Any external references used during identification or development of the optimization

Chapter 16 Application Case Study - Machine Learning

Chapter 9 Parallel Patterns - Parallel Histogram Computation

ECE 508-Lecture 3-slides-Spring 2017

How your team organized and divided up this work

The team worked together to implement the baseline of the final project together, including the host function, kernel function, thread coarsening, memory coalescing, and sweeping parameters. In addition, Zhiyu helped with milestone 1-3, implemented the constant memory, shared memory, unrolling, multiple layer kernel and contributed to the final report, Ye Zhang implemented milestone 3 and wrote the final report. Pei implemented milestone 1 & 2, loop unrolling and wrote the final report.

Suggestion for next year

Thanks a lot to the professor and the TA team. One suggestion would be providing more optional practice questions like the ones in the exams after each lecture.

References

Hwu, W.(2011) "*Advanced Algorithmic Techniques for GPUs." PowerPoint Presentation*. University of California, Berkeley. 24 January 2011.

Hwu, W. and Kirk, D. (2016). "*Ch.9 Parallel Patterns - Parallel Histogram Computation.*" 3rd ed. pp.2-10.

Hwu, W. and Kirk, D. (2016). "*Ch.16 Application Case Study - Machine Learning.*" 3rd ed. pp.2-10.