# ECE408 Report

**APLUSPLUS**

## 1. Baseline Results

## Milestone 1 : Get Started

### M1.1: Run the Baseline Forward Pass

Run CPU code in rai
**Our Results as below(add elapsed time for whole m1.1.py program):**

✱ Running time python m1.1.py
New Inference
Loading fashion-mnist data... done
Loading model... done
EvalMetric: {'accuracy': 0.8673}
11.40user 11.48system
0:21.23elapsed 107%CPU (0avgtext+0a
vgdata 1628360maxresident)k
0inputs+2624outp
uts (0major+27159minor)pagefau
lts 0swaps

### M1.2/1.3: Run the Baseline GPU implementation and generate a NVPROF profile

Modified rai_build.yml according to introduction.
Mxnet GPU layer performance results is as below:

✱ Running nvprof python m1.2.py
New Inference
Loading fashion-mnist data... done
==310== NVPROF is profiling process 310, command: python m1.2.py
Loading model...[02:36:12] src/operator/././cudnn_algoreg-inl.h:112: Running performance 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}
==310== Profiling application: python m1.2.py
==310== Profiling result:

```
Time(%)      Time     Calls       Avg       Min       Max  Name
37.01%  50.022ms         1  50.022ms  50.022ms  50.022m s  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)
 28.67%  38.751ms         1  38.751ms  38.751ms  38.751ms  sgemm_sm35_ldg_tn_128x8x256x16x32
14.34%  19.385ms         2  9.6923ms  459.06us  18.926ms  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*)
 10.69%  14.445ms         1  14.445ms  14.445ms  14.445ms  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)
  4.53%  6.1185ms        13  470.66us  1.5360us  4.1914ms  [CUDA memcpy HtoD]
  2.75%  3.7160ms         1  3.7160ms  3.7160ms  3.7160ms  sgemm_sm35_ldg_tn_64x16x128x8x32
  0.82%  1.1115ms         1  1.1115ms  1.1115ms  1.1115ms  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)
0.55%  748.43us        12  62.369us  2.0800us  378.04us  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%  433.43us         2  216.72us  16.639us  416.79us  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)
  0.29%  390.68us         1  390.68us  390.68us  390.68us  sgemm_sm35_ldg_tn_32x16x64x8x16
0.02%  22.399us         1  22.399us  22.399us  22.399us  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%  10.016us         1  10.016us  10.016us  10.016us  [CUDA memcpy DtoH]
==310== API calls:
Time(%)      Time     Calls       Avg       Min       Max  Name
 46.76%  1.85341s        18  102.97ms  17.717us  926.36ms  cudaStreamCreateWithFlags
 28.75%  1.13936s        10  113.94ms     696ns  322.52ms  cudaFree
 20.66%  818.70ms        24  34.112ms  230.81us  811.60ms  cudaMemGetInfo
  3.24%  128.32ms        25  5.1329ms  5.5140us  83.313ms  cudaStreamSynchronize
  0.31%  12.278ms         8  1.5348ms  8.4420us  6.1684ms  cudaMemcpy2DAsync
  0.17%  6.5406ms        42  155.73us  10.600us  1.1805ms  cudaMalloc
  0.03%  1.3602ms         4  340.06us  338.87us  342.68us  cuDeviceTotalMem
  0.02%  862.41us       114  7.5640us     625ns  307.12us  cudaEventCreateWithFlags
  0.02%  848.24us       352  2.4090us     248ns  63.433us  cuDeviceGetAttribute
  0.01%  527.45us        23  22.932us  11.237us  89.276us  cudaLaunch
  0.01%  419.09us         6  69.847us  59.623us  81.884us  cudaMemcpy
  0.01%  233.20us         4  58.298us  35.204us  83.353us  cudaStreamCreate
  0.00%  102.25us         4  25.561us  17.241us  32.322us  cuDeviceGetName
  0.00%  78.556us        32  2.4540us     695ns  8.6440us  cudaSetDevice
  0.00%  70.258us       110     638ns     418ns  2.3920us  cudaDeviceGetAttribute
  0.00%  57.402us       147     390ns     253ns  1.1690us  cudaSetupArgument
0.00%  40.865us         2  20.432us  18.308us  22.557us  cudaStreamCreateWithPriority
  0.00%  26.085us        23  1.1340us     492ns  3.3300us  cudaConfigureCall
  0.00%  24.680us        10  2.4680us  1.3040us  5.9290us  cudaGetDevice
```

```
 0.00%  9.0930us         1  9.0930us  9.0930us  9.0930us  cudaBindTexture
 0.00%  8.7240us        16     545ns     407ns     891ns  cudaPeekAtLastError
 0.00%  4.3310us         6     721ns     240ns  1.4230us  cuDeviceGetCount
 0.00%  4.1760us         2  2.0880us  1.5150us  2.6610us  cudaStreamWaitEvent
 0.00%  3.7180us         1  3.7180us  3.7180us  3.7180us  cudaStreamGetPriority
 0.00%  3.4330us         6     572ns     415ns     874ns  cuDeviceGet
 0.00%  3.4270us         2  1.7130us  1.3330us  2.0940us  cudaEventRecord
 0.00%  3.2360us         2  1.6180us  1.4250us  1.8110us  cudaDeviceGetStreamPriorityRange
 0.00%  3.0600us         6     510ns     325ns     737ns  cudaGetLastError
 0.00%  3.0510us         3  1.0170us     931ns  1.1630us  cuInit
 0.00%  2.0790us         3     693ns     621ns     779ns  cuDriverGetVersion
 0.00%  1.9120us         1  1.9120us  1.9120us  1.9120us  cudaUnbindTexture
 0.00%  1.1120us         1  1.1120us  1.1120us  1.1120us  cudaGetDeviceCount
```

The profile displays two parts of time. First is the time consumed by each kernel. For m1.2 most time is consumed by *cudnn::detail::implicit_convolve_sgemm* this kernel(37.01% 50.022ms). The second is the time consumed by each API calls like cudaFree cudaMemcpy. For m1.2, cudaStreamCreateWithFlags API call costed 46.76%(1.85341s, called 18 times) of total time of all API calls.

# Milestone 2: A New CPU Layer in MXNet

## M2.1 Add CPU forward implementation

M2.1.1 Description of implementation

Batch size: B
Input features/channels:C inputs (H× W).
Convolution Layer: M filters (K x K).
Output Features/channels:M outputs (H – K+1) × (W – K+1).
We use ‘C’ to represent the number of input feature maps and use ‘H’ to represent the height of each input map image, and the width of each is  ‘W’.Assume that the input feature maps are stored in a 3D array X [B,C, H, W].We have M outputs feature map, and each size is (H – K+1) × (W – K+1). The following shows a sequential code of CNN for forward propagation path.

Our Code here:

```
 const int B = x.shape_[0];
const int M = y.shape_[1];
const int C = x.shape_[1];
const int H = x.shape_[2];
const int W = x.shape_[3];
const int K = k.shape_[3];
 int H_out = H - K + 1;
   int W_out = W - K + 1;
   for (int b = 0; b < B; ++b) {
       //CHECK_EQ(0, 1) << "Missing an ECE408 CPU implementation!";
```

```
/* ... a bunch of nested loops later...
        y[b][m][h][w] += x[b][c][h + p][w + q] * k[m][c][p][q];
    */
    for(int m = 0;  m < M;  m++)              // for each output feature map
        for(int h = 0; h < H_out; h++)          // for each output element
          for(int w = 0; w < W_out; w++) {
            y[b][m][h][w] = 0;
            for(int c = 0;  c < C; c++)         // sum over all input feature maps
              for(int p = 0; p < K; p++)           // KxK  filter
                for(int q = 0; q < K; q++)
                  y[b][m][h][w] += x[b][c][h + p][w + q] * k[m][c][p][q];
```

M2.1.2 Result and performance

Our baseline cpu implementation correctness and performance results  is:

```
✱ Running python m2.1.py
New Inference
Loading fashion-mnist data... done
Loading model... done
Op Time: 9.045332
Correctness: 0.8562 Model: ece408-high
```

# Milestone 3

## 3.1 Add a simple GPU forward implementation

In this milestone, we implemented the basic GPU forward convolution and then used shared memory to load filter kernel and tiled input as an optimization.

Shared memory size :

filter kernel C*K*K*sizeof(float)

input: (TILE_WIDTH+K-1)^2

## 3.2 Create a GPU profile with nvprof.

```
✱ Running nvprof python m3.1.py
New Inference
Loading fashion-mnist data... done
==311== NVPROF is profiling process 311, command: python m3.1.py
```

```
Loading model... done
Op Time: 0.103691
Correctness: 0.8562 Model: ece408-high
==311== Profiling application: python m3.1.py
==311==
Profiling result:
Time(%)      Time     Calls       Avg       Min       Max  Name
34.28%  65.149ms         1  65.149ms  65.149ms  65.149ms  mxnet::op::forward_kernel(float*, float const *, float const *, int,
int, int, int, int, int, int)
 20.65%  39.252ms         1  39.252ms  39.252ms  39.252ms  sgemm_sm35_ldg_tn_128x8x256x16x32
10.21%  19.399ms         1  19.399ms  19.399ms  19.399ms  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)
10.20%  19.394ms         2  9.6968ms  461.21us  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*)
9.49%  18.029ms         3  6.0095ms  3.1360us  17.535ms  [CUDA memcpy DtoD]
7.63%  14.495ms         1  14.495ms  14.495ms  14.495ms  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)
4.19%  7.9578ms        13  612.13us  1.8560us  5.5119ms  [CUDA memcpy HtoD]
  1.92%  3.6513ms         1  3.6513ms  3.6513ms  3.6513ms  sgemm_sm35_ldg_tn_64x16x128x8x32
0.59%  1.1199ms         1  1.1199ms  1.1199ms  1.1199ms  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)
0.40%  754.10us        12  62.841us  2.0800us  380.99us  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)
0.23%  437.72us         2  218.86us  17.408us  420.31us  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)
  0.21%  394.01us         1  394.01us  394.01us  394.01us  sgemm_sm35_ldg_tn_32x16x64x8x16
0.01%  23.264us         1  23.264us  23.264us  23.264us  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%  9.9520us         1  9.9520us  9.9520us  9.9520us  [CUDA memcpy DtoH]
==311== API calls:
Time(%)      Time     Calls       Avg       Min       Max  Name
 46.50%  2.82762s        18  157.09ms  19.678us  1.42871s  cudaStreamCreateWithFlags
 29.30%  1.78217s        13  137.09ms  1.4970us  514.67ms  cudaFree
 20.66%  1.25635s        23  54.624ms  280.94us  1.24862s  cudaMemGetInfo
  1.36%  82.565ms         1  82.565ms  82.565ms  82.565ms  cudaDeviceSynchronize
  1.29%  78.531ms        25  3.1412ms  6.6570us  42.555ms  cudaStreamSynchronize
  0.56%  33.779ms        44  767.70us  11.330us  20.586ms  cudaMalloc
  0.23%  13.735ms         8  1.7169ms  14.264us  5.7998ms  cudaMemcpy2DAsync
  0.03%  1.7250ms         4  431.25us  37.675us  1.5485ms  cudaStreamCreate
```

| | | | | | | |
|---|---|---|---|---|---|---|
| 0.03% | 1.5263ms | 4 | 381.57us | 378.31us | 385.95us | cuDeviceTotalMem |
| 0.02% | 1.2081ms | 24 | 50.338us | 16.392us | 357.39us | cudaLaunch |
| 0.02% | 1.1688ms | 352 | 3.3200us | 307ns | 115.66us | cuDeviceGetAttribute |
| 0.01% | 330.49us | 9 | 36.720us | 21.596us | 85.209us | cudaMemcpy |
| 0.00% | 172.89us | 114 | 1.5160us | 925ns | 4.3730us | cudaEventCreateWithFlags |
| 0.00% | 116.52us | 4 | 29.130us | 21.645us | 33.854us | cuDeviceGetName |
| 0.00% | 113.49us | 30 | 3.7820us | 854ns | 11.621us | cudaSetDevice |
| 0.00% | 87.898us | 104 | 845ns | 592ns | 2.0750us | cudaDeviceGetAttribute |
| 0.00% | 82.803us | 146 | 567ns | 346ns | 1.5660us | cudaSetupArgument |
| 0.00% | 52.197us | 2 | 26.098us | 21.625us | 30.572us | cudaStreamCreateWithPriority |
| 0.00% | 42.172us | 24 | 1.7570us | 682ns | 5.1630us | cudaConfigureCall |
| 0.00% | 26.656us | 10 | 2.6650us | 1.3070us | 8.4980us | cudaGetDevice |
| 0.00% | 13.254us | 17 | 779ns | 613ns | 1.0220us | cudaPeekAtLastError |
| 0.00% | 9.2620us | 2 | 4.6310us | 2.1300us | 7.1320us | cudaEventRecord |
| 0.00% | 7.1010us | 1 | 7.1010us | 7.1010us | 7.1010us | cudaStreamGetPriority |
| 0.00% | 5.8340us | 6 | 972ns | 485ns | 2.1170us | cuDeviceGetCount |
| 0.00% | 5.7600us | 2 | 2.8800us | 2.2420us | 3.5180us | cudaStreamWaitEvent |
| 0.00% | 4.6220us | 6 | 770ns | 471ns | 1.2630us | cuDeviceGet |
| 0.00% | 4.3260us | 5 | 865ns | 667ns | 1.0680us | cudaGetLastError |
| 0.00% | 4.2820us | 2 | 2.1410us | 2.0100us | 2.2720us | cudaDeviceGetStreamPriorityRange |
| 0.00% | 3.7080us | 3 | 1.2360us | 1.0900us | 1.4200us | cuDriverGetVersion |
| 0.00% | 3.4300us | 3 | 1.1430us | 1.0180us | 1.3220us | cuInit |
| 0.00% | 1.5140us | 1 | 1.5140us | 1.5140us | 1.5140us | cudaGetDeviceCount |

# 2.Optimization Approach and Results

1,2,3.how you identified the optimization opportunity
We use following optimised functions:
**Unrolled** the input features and convert the convolution to **matrix multiplication**. While optimizing the matrix multiplication we considered the memory access coalesce, control divergence and global memory access. Used tiled matrix multiplication and applied some modification according to our input data size.

First of all, we implemented the MP3. if we use 32 as TILE_WIDTH, every warp will have control divergence while loading W and X into shared memory then, and warp near the boundary will have control divergency writing the product into Y according as the input size of w is 50*25, and X size is 25*576. So, we chose the 25 as the TILE_WIDTH. In this case, threads have no control divergence while loading input W into shared memory, but it still have control divergency while loading X and writing Y. After changed the size of TILE_WIDTH, the time decreased from 0.150975s to 0.140642s.

We also tried constant memory. We thought it could be effective by using constant memory to improve the performance a little bit because filter W is used by every batch and w. However, the results turned out to be opposite. The time it spent was 0.19625s, which was longer than the time using shared memory 0.140642s.

We then analyzed the tiled matrix multiplication. If we using 25 as Tile width, the kernel will launch 2*24 thread blocks for each input batch . During the first iteration, block(0,0), block(0,1)... block(0,23) loaded $W_{0,0}$...$W_{24,24}$(25X25) into shared memory respectively. So, the kernel access the same W elements multiple times in order to calculate Y in one batch. We figured that we could use one thread to calculate two output by using one block to load two block size of X and calculate two block size of Y. In this case, we only launched 2*12 thread blocks for one batch. And block(0,0) loaded $W_{0,0}$...$W_{24,24}$(25X25) and $X_{0,0}$...$X_{24,24}$(25X25) $X_{25,0}$...$X_{49,49}$(25X25), and calculated $X_{0,0}$...$X_{24,49}$(25X50). This reduced 1X global memory access. After implemented above, the time decreased to 0.099665s.

Similarly, we can use one thread to calculate four output. Block(0,0) will load $W_{0,0}$...$W_{49,24}$(50X25) and $X_{0,0}$...$X_{49,49}$(25X50) and calculate $X_{0,0}$...$X_{49,49}$(50X50). This will reduce the number of parallel computation but will reduce the time in the global memory access. The time performance of this method is 0.071211s. The time of Forward_kernel which is matrix multiplication kernel is 43.79ms

```
New Inference
Loading fashion-mnist data... done
==315== NVPROF is profiling process 315, command: python final.py
Loading model... done
Op Time: 0.071211
Correctness: 0.8562 Model: ece408-high
==315== Profiling application: python final.py
==315== Profiling result:
Time(%)      Time     Calls       Avg       Min       Max  Name
25.21%  43.790ms         1  43.790ms  43.790ms  43.790ms  mxnet::op::forward_kernel(float*, float*, float*, int, int, int, int, int, int)
```

4, Any external references used during identification or development of the optimization

[1]Guangming Tan, Linchuan Li, Sean Treichler, Everett Phillips, Yungang Bao, and Ninghui Sun. Fast implementation of DGEMM on Fermi GPU. In Supercomputing 2011, SC ’11, pages 35:1–35:11, New York, NY, USA, 2011. ACM.
[2] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In NIPS, pages 1097–1105, 2012.

5, How your team organized and divided up this work.

In Final stage:
We worked together to come up with the optimized function in order to accelerate the model.

6, References (as needed)
[1]NVIDIA cuDNN - GPU accelerated deep learning.
[2]Chapter 16 - 3rd-Edition-Chapter16-case-study-DNN-FINAL-corrected.pdf

7,(Optional) Suggestions for Improving Next Year
We hope to make our project in  flexible topics. For example,Game Tree Search, CoMD GPU implementation. And use criteria to grade so that student could Play creativity.

## Contribution

Chaohua Shang: Write and run the code, wrote most of the milestone 1, milestone 3 and final part in report
Haojia: Write and run the code, wrote the final part and assist Chaohua and Jiayue with the report by providing information and writing part of it.
Jiayue Wang: Write and run the code, write most of the milestone 2 and final part in report