# ECE408 Report

**APLUSPLUS**

## 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  50.022ms  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
```

# Contribution

Chaohua Shang: Write and run the code, write most of the milestone 1 part in report
Haojia: Write and run the code, 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 part in report