ECE408 Final Project Report

Group name:sync_chicken
NetId:kesun5/yuanm2/yanyeli2

Milestone 1

Register your team in the google sheet.
Report: Include a list of all kernels that collectively consume more than 90% of the program time.
Report: Include a list of all CUDA API calls that collectively consume more than 90% of the program time.
Report: Include an explanation of the difference between kernels and API calls
Report: Show output of rai running MXNet on the CPU
Report: List program run time
Report: Show output of rai running MXNet on the GPU
Report: List program run time

A list of all kernels that collectively consume more than 90% of the program time:

1. **void fermiPlusCgemmLDS128_batched<bool=0, bool=1, bool=0, bool=0, int=4, int=4, int=4, int=3, int=3, bool=1, bool=1>(float2**, float2**, float2**, float2*, float2 const *, float2 const *, int, int, int, int, int, int, __int64, __int64, __int64, float2 const *, float2 const *, float2, float2, int)**

2. **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)**
3. **void fft2d_c2r_32x32<float, bool=0, unsigned int=0, bool=0, bool=0>(float*, float2 const *, int, int, int, int, int, int, int, int, int, float, float, cudnn::reduced_divisor, bool, float*, float*)**
4. **Sgemm_sm35_ldg_tn_128x8x256x16x32**
5. **[CUDA memcpy HtoD]**
6. **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*)**
7. **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)**

Include a list of all CUDA API calls that collectively consume more than 90% of the program time:

1. **cudaStreamCreateWithFlags**
2. **cudaFree**
3. **cudaMemGetInfo**
4. **cudaMemcpy2DAsync**

Difference between kernels and API calls:

- A **kernel** is a low level program interfacing with the hardware on top of which applications are running. It is the lowest level program running on computers although with virtualization you can have multiple kernels running on top of virtual machines which themselves run on top of another operating system.

- An **API** is a generic term defining the interface developers have to use when writing code using libraries and a programming language. **Kernels have no APIs** as they are not libraries.

Show output of rai running MXNet on the CPU

Loading fashion-mnist data...

done

Loading model...

done

New Inference

EvalMetric: {'accuracy': 0.8444}

List program run time on CPU

13.12user 8.31system 0:10.59elapsed 202%CPU

Show output of rai running MXNet on the GPU

Loading fashion-mnist data...

done

Loading model...

done

New Inference

EvalMetric: {'accuracy': 0.8444}

✱ Running /usr/bin/time python m1.2.py

Loading fashion-mnist data...

done

Loading model...

[04:33:40] 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

New Inference

EvalMetric: {'accuracy': 0.8444}

(0avgtext+0avgdata 1136388maxresident)k

0inputs+3136outputs (0major+158216minor)pagefaults 0swaps

List program run time on GPU

2.27user 1.11system 0:02.84elapsed 119%CPU

Milestone 2

Everything from Milestone 1
Create a CPU implementation
Report: List whole program execution time
Report: List Op Times

```
* Running /usr/bin/time python m2.1.py
Loading fashion-mnist data...
done
Loading model...
done
New Inference
Op Time: 7.463287
Op Time: 25.678084
Correctness: 0.8451 Model: ece408
37.80user 1.35system 0:37.09elapsed 105%CPU (0avgtext+0avgdata 2814784maxresiden
t)k
```

The whole program execution time is 37.09 seconds

The first layer's op time is 7.463287 seconds

The second layer's op time is 25.678084 seconds

Milestone 3

Everything from Milestone 2
Implement a GPU Convolution
Report: demonstrate `nvprof` profiling the execution
Use `rai -p <project folder> --submit=m3` to mark your job for grading

GPU running performance in datasize 100:

Name	Invocations	Avg. Duration	Regs	Static SMem	Avg. Dynamic SMem	Inter-Thread Instructions	Issue Stall Reasons (Other)
memset (0)	0	0 ns	0	0	0	n/a	n/a
void scal_kernel<float, float, int=1, bool=1, i...	1	3.584 µs	17	0	0	0	0.8%
void mshadow::cuda::MapPlanKernel<mshad...	2	3.76 µs	8	0	0	0	0.4%
void mshadow::cuda::MapPlanKernel<mshad...	1	12.608 µs	22	0	0	0	0.1%
void mshadow::cuda::MapPlanKernel<mshad...	14	14.23 µs	8	0	0	0	0.6%
void mshadow::cuda::SoftmaxKernel<int=8, f...	1	18.496 µs	18	1024	0	0	4.4%
sgemm_sm35_ldg_tn_32x16x64x8x16	1	41.279 µs	77	6676	0	0	4.9%
void cudnn::detail::activation_fw_4d_kernel<f...	2	80.319 µs	30	0	0	0	1.4%
void mshadow::cuda::MapPlanKernel<mshad...	2	127.71 µs	8	0	0	0	0.9%
void cudnn::detail::pooling_fw_4d_kernel<flo...	1	142.814 µs	35	0	3616	0	1.9%
mxnet::op::unroll(float const *, float*, int, int, ...	2	528.809 µs	22	0	0	0	2%
void sgemm_largek_lds64<bool=1, bool=0, i...	1	686.487 µs	33	4356	0	0	2.3%
mxnet::op::forward_kernel(float*, float const ...	2	1.02995 ms	21	0	0	0	1.2%

```
Compute
44.3% mxnet::op::forward_kernel(float*, float const *, float const *, int, int, int, int, int, int)
22.8% mxnet::op::unroll(float const *, float*, int, int, int, int, int, int)
14.8% void sgemm_largek_lds64<bool=1, bool=0, int=5, int=5, int=4, int=4, int=4, int=34>(float*, float const *, float const *, int, int, int, int, int, int, float const *, float const *, float, float, int, int, int*, int*)
5.5% void mshadow::cuda::MapPlanKernel<mshadow::sv::saveto, int=8, mshadow::expr::Plan<mshadow::Tensor<mshadow::gpu, int=4, float>, float>, mshadow::expr::Plan<mshadow::expr::BinaryMapExp<mshadow::op::mul, mshad...
4.3% 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,...
3.5% 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, cud...
3.1% 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::deta...
0.9% sgemm_sm35_ldg_tn_32x16x64x8x16
0.4% 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, i...
0.3% 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::maxi...
0.2% 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<msha...
0.1% void scal_kernel<float, float, int=1, bool=1, int=6, int=5, int=5, int=3>(cublasTransposeParams<float>, float const *, float*, float const *)
0.0% memset (0)
```

We have two kernels, an unrolled kernel and a matrix multiplication kernel to complete the convolution job. According to the performance report, matrix multiplication kernel takes the most computation time in the forward propagation layer. We believe this kernel can be further optimized by using tiling matrix multiplication. Details will be discussed later in milestone 4.

And the actual Optime of gpu implementation with data size 100.

```
Running nvprof -o timeline.nvprof python m3.1.py 100
Loading fashion-mnist data...
done
Loading model...
==341== NVPROF is profiling process 341, command: python m3.1.py 100
done
New Inference
Op Time: 0.003760
Op Time: 0.007205
Correctness: 0.88 Model: ece408
```

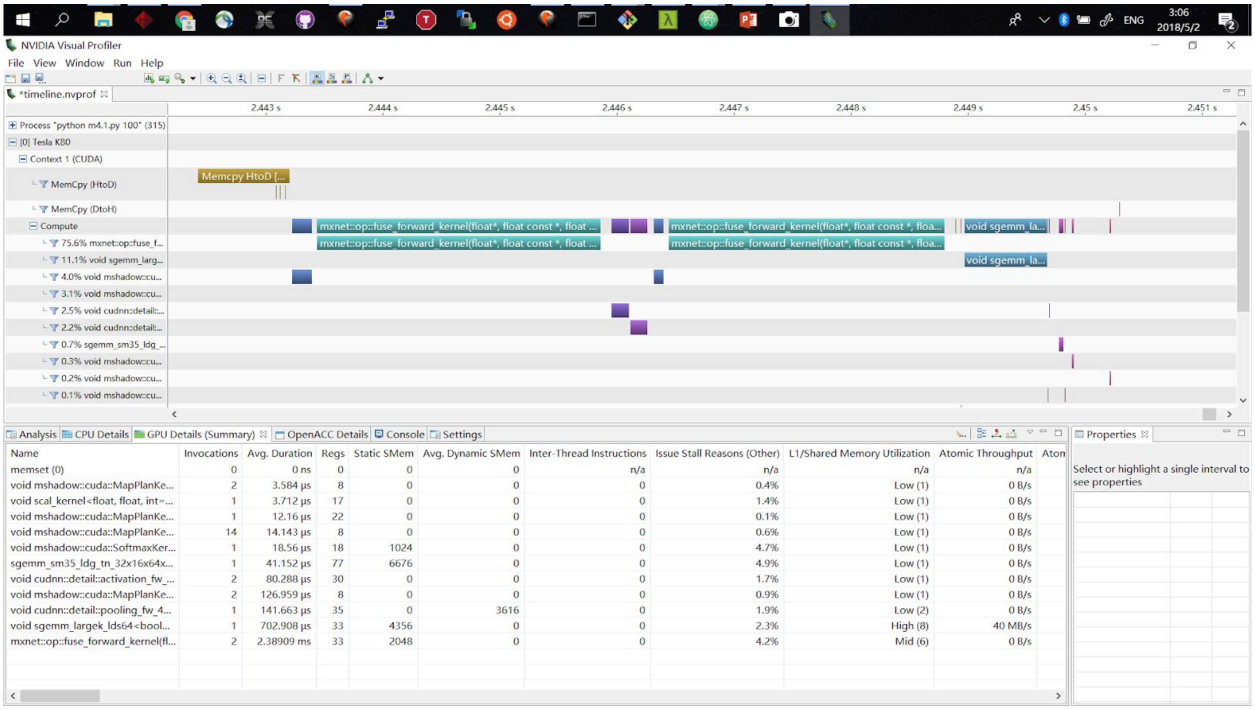
For convenience of comparison between CPU(in milestone 2) and GPU, we provide the actual OP time of GPU implementation with data size 10000.

```
* Running /usr/bin/time -f "%Uuser %Ssystem %eelapsed" python /eval-scripts/final.py
Loading fashion-mnist data... done
Loading model...
done
New Inference
Op Time: 0.290717
Op Time: 0.486178
Correctness: 0.8451 Model: ece408
2.56user 1.39system 3.40elapsed
```

We think this GPU optimization accelerates in a great amount compared with CPU version.

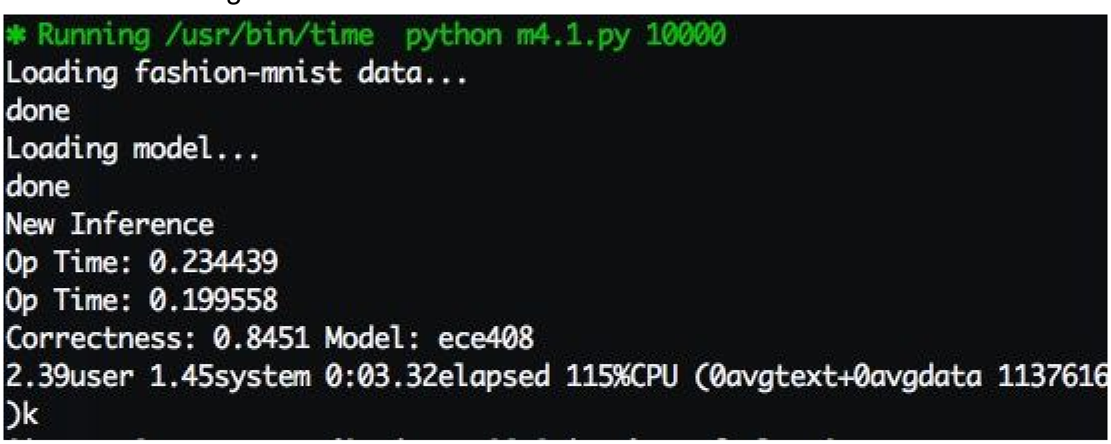
To be specific, we first unrolled the input image x and then change the convolution to matrix multiplication. This strategy reduces the redundant memory access caused by the convolution method and can be further optimized by advanced tiling matrix multiplication.

Milestone 4



Above is the nvprof performance analysis of milestone 4 with data size 100.

The actual running time of our final version code is around 430ms.



We actually already used unrolling and matrix multiplication in mile stone 3 so this time we changed it to unrolling and shared matrix multiplication. This strategy indeed improves our performance by 150ms (compared with the most basic processing method).

The second optimization we take is kernel fusion. We only call the fused matrix multiplication kernel and use different index representation to directly access the element in the input feature map. It turns out that this optimization improves our performance by 200ms.

The last optimization we use is tuning with restrict and loop unrolling. Basically we rewrite the for loop in the kernel and change each iteration to a single line of execution with certain parameters. This optimization improves our performance by around 20ms.