ECE 408 Final Project Report

Team: ParallelCorn

Xiaoming Zhao (NetID: xz23)
Chieh Hsu (NetID: chielhh2)
Bohan Zhang (NetID: bohanz2)

March 2018

1 Milestone 1

1.1 Include a list of all kernels that collectively consume more than 90% of the program time

1. **34.05%** (118.44ms), 9 calls, 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. **26.98%** (93.871ms), 1 call, 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. **12.68%** (44.126ms), 9 calls, 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. **8.19%** (28.494ms), 1 call, sgemm_sm35_ldg_tn_128x8x256x16x32
5. **6.50%** (22.602ms), 14 calls, [CUDA memcpy HtoD]
6. **4.07%** (14.159ms), 2 calls, 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*)

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

Time(%)	Time	Calls	Name
43.62%	1.94235s	18	cudaStreamCreateWithFlags
27.21%	1.21127s	10	cudaFree
20.60%	917.27ms	27	cudaMemGetInfo

Table 1: CUDA API Calls

1.3 Include an explanation of the difference between kernels and API calls

Kernels are user-coded functions that are called by the host and executed on the device (GPU, typically), whereas API calls are invoking the functions that are provided by Cuda as interface.

1.4 Show output of rai running MXNet on the CPU

```
^[[32m✱Running python m1.1.py^[[0m
Loading fashion-mnist data...
done
Loading model...
done^M
New Inference
EvalMetric: {'accuracy': 0.8444}
^[[32m✱The build folder has been uploaded to http://s3.amazonaws.com/files.rai-project.com/userdata/build-bbdb2520-11a0-437b-af4c-f42e82
bf10e6.tar.gz. The data will be present for only a short duration of time.^[[0m
^[[32m✱Server has ended your request.^[[0m
```

Figure 1: MXNet CPU

1.5 List program run time

User: 12.67s; System: 6.27s

1.6 Show output of rai running MXNet on the GPU

```
^[[32m✱Running python m1.2.py^[[0m
Loading fashion-mnist data...
done
Loading model...
[09:21:00] src/operator/././cudnn_algoreg-inl.h:112: Running performance tests to find the best convolution algorithm, this can take a wh
ile... (setting env variable MXNET_CUDNN_AUTOTUNE_DEFAULT to 0 to disable)
done^M
New Inference
EvalMetric: {'accuracy': 0.8444}
^[[32m✱The build folder has been uploaded to http://s3.amazonaws.com/files.rai-project.com/userdata/build-56125cb6-ac27-4474-ab79-c93493
6d6d00.tar.gz. The data will be present for only a short duration of time.^[[0m
^[[32m✱Server has ended your request.^[[0m
```

Figure 2: MXNet GPU

1.7 List program run time

User: 2.30s; system: 1.10s

2 Milestone 2

2.1 Whole Program Execution Time

User: 30.48s; System: 1.48s

2.2 Op Times

First Layer Op Time: 6.570814s; Second Layer Op Time: 19.473800s

3 Milestone 3

3.1 nvprof Timeline API Calls

Time(%)	Time	Calls	Avg	Min	Max	Name
36.93%	1.93394s	18	107.44ms	23.882us	966.80ms	cudaStreamCreateWithFlags
22.91%	1.19950s	10	119.95ms	1.0020us	339.73ms	cudaFree
20.03%	1.04880s	6	174.80ms	13.403us	671.17ms	cudaDeviceSynchronize
17.80%	931.98ms	27	34.518ms	249.75us	923.94ms	cudaMemGetInfo
1.20%	62.583ms	29	2.1580ms	5.8340us	32.221ms	cudaStreamSynchronize
0.91%	47.487ms	9	5.2764ms	17.350us	22.964ms	cudaMemcpy2DAsync
0.13%	6.8965ms	45	153.26us	9.2670us	899.76us	cudaMalloc
0.03%	1.3578ms	4	339.46us	335.44us	348.66us	cuDeviceTotalMem
0.02%	1.1504ms	114	10.091us	956ns	425.89us	cudaEventCreateWithFlags
0.02%	978.26us	352	2.7790us	510ns	70.432us	cuDeviceGetAttribute
0.01%	591.66us	28	21.130us	9.3490us	76.754us	cudaLaunch
0.01%	363.96us	6	60.660us	30.285us	130.42us	cudaMemcpy
0.01%	278.61us	4	69.651us	55.444us	101.45us	cudaStreamCreate
0.00%	112.65us	168	670ns	527ns	1.6580us	cudaSetupArgument
0.00%	112.24us	104	1.0790us	854ns	1.9860us	cudaDeviceGetAttribute
0.00%	100.32us	4	25.080us	18.442us	29.777us	cuDeviceGetName
0.00%	88.815us	34	2.6120us	888ns	7.4090us	cudaSetDevice
0.00%	50.697us	2	25.348us	24.627us	26.070us	cudaStreamCreateWithPriority
0.00%	38.625us	28	1.3790us	691ns	2.4110us	cudaConfigureCall
0.00%	26.677us	10	2.6670us	1.4880us	8.6180us	cudaGetDevice
0.00%	14.908us	20	745ns	592ns	1.0340us	cudaPeekAtLastError
0.00%	6.4370us	6	1.0720us	546ns	2.4080us	cuDeviceGetCount
0.00%	5.8180us	2	2.9090us	2.8400us	2.9780us	cudaStreamWaitEvent
0.00%	5.2330us	6	872ns	635ns	1.2940us	cuDeviceGet
0.00%	5.2240us	2	2.6120us	2.5310us	2.6930us	cudaEventRecord
0.00%	4.7060us	2	2.3530us	2.0230us	2.6830us	cudaDeviceGetStreamPriorityRange
0.00%	4.4890us	5	897ns	654ns	1.1180us	cudaGetLastError
0.00%	3.4770us	3	1.1590us	1.0330us	1.2480us	cuInit
0.00%	3.4240us	1	3.4240us	3.4240us	3.4240us	cudaStreamGetPriority
0.00%	2.9860us	3	995ns	962ns	1.0470us	cuDriverGetVersion
0.00%	1.4480us	1	1.4480us	1.4480us	1.4480us	cudaGetDeviceCount

Table 2: CUDA API Calls

3.2 Top 3 Representative Profiling Result

Time(%)	Time	Calls	Avg	Min	Max	Name
90.42%	1.02679s	2	513.39ms	355.65ms	671.14ms	mxnet::op::forward_kernel
2.54%	28.823ms	1	28.823ms	28.823ms	28.823ms	sgemm_sm35_ldg_tn_128x8x256x16x32
2.08%	23.661ms	14	1.6901ms	1.5360us	22.812ms	[CUDA memcpy HtoD]

Table 3: Partial Profiling Result

3.3 Speedup with GPU

According to nvprof, the GPU convolution has the significant overall speedup when compared with the CPU implementation (0.355 on GPU vs 6.599 on CPU).

3.4 Individual Optimization

Inside the convolution kernel, the GPU code uses 16*16 tiles which enables every warp to access two consecutive memory sections, each consisting of 16 locations. This optimization utilizes 50 percent of the memory burst. On the other hand, given the relatively small block size, the kernel did not use shared memory. Thus the overhead introduced by barrier synchronization and the extra loading process is minimized for this small-block-sized convolution kernel.