ECE408 Final Project Report

Shriyaa Mittal (smittal6), Tiancheng Wu (twu54), Chuankai Zhao (czhao37)

Team Name: smartconvolutionteam

MileStone 3: Due November 16, 2018

3.1 Correctness and timing with 3 different dataset sizes

Dataset Size 100

```
Op Time: 0.000446
Op Time: 0.001617
Correctness: 0.85 Model: ece408

3.99user 2.37system 0:04.20elapsed 151%CPU
```

Dataset Size 1000

```
Op Time: 0.004267
Op Time: 0.016125
Correctness: 0.827 Model: ece408

4.03user 2.41system 0:03.99elapsed 161%CPU
```

Dataset Size 10,000

```
Op Time: 0.042534
Op Time: 0.152912
Correctness: 0.8171 Model: ece408

4.03user 2.84system 0:04.41elapsed 155%CPU
```

3.2 Demonstrate nvprof profiling the execution

Figure 1 shows a timeline for the execution of the code on dataset size 1,000 using NVIDIA Visual Profiler. The profiling overhead is highlighted in red.

In particular, the $forward\_kernel$ is called twice in the current execution. The first call takes shorter time and the the second takes longer time (for all dataset sizes). The exact start and end time can be seen in the Properties View of the profiler. For both calls to $forward\_kernel$, some important properties are summarized in Table 1.

Table 1: Kernel execution properties for dataset size 1000.

Property	**1st Call**	**2nd Call**
Grid Size	[1000,12,25]	[1000,24,4]
Block Size	[16,16,1]	[16,16,1]
Global Load Efficiency	57.4%	56.9%
Global Store Efficiency	68.6%	65.9%
Warp Execution Efficiency	83.9%	81.5%
Occupancy Acheived	84.9%	90.6%

Overall, both compute and memory are likely bottlenecks to performance for the *forward_kernel*, and latency is likely not the primary performance bottleneck for this kernel.

3.2.1 Compute analysis

3.2.1.1 Low warp execution efficiency

The warp execution efficiency for the *forward_kernel* is 83.9% during the first call and 81.5% during the second call, if predicted instructions are not taken into account. The kernel's not predicted off warp execution efficiency is less than 100% due to divergent branches and predicted instructions. It is suggested to reduce the amount of intra-warp divergence and prediction in the kernel.

3.2.1.2 Divergent Branches

In the first *forward_kernel* call, the divergence rate (line 42 in new-forward.cuh) is 16.5% (396,000 divergent executions out of 2,400,000 total executions). In the second *forward_kernel* call, the divergence rate (line 42 in new-forward.cuh) is 46.9% (360,000 divergent executions out of 768,000 total executions. Divergent branches lower warp execution efficiency which leads to inefficient use of the GPU's compute resources.

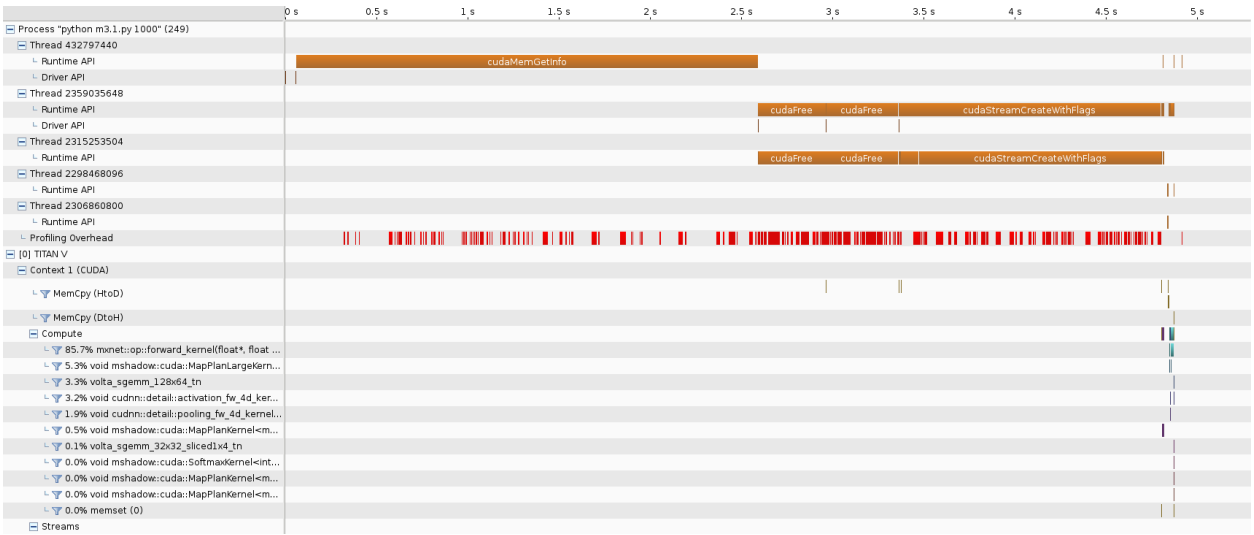

Figure 1: Execution timeline on dataset size 1000 as seen with NVIDIA Visual Profiler.

3.2.2 Memory bandwidth analysis

3.2.2.1 Inefficient global memory alignment and access pattern

It is suggested that the *forward_kernel* kernel has an efficient global memory alignment and access pattern (lines 48 and 49 in new-forward.cuh). As a result, memory bandwidth is not used most efficiently.

3.2.2.2 GPU utilization limited by memory bandwidth

Figure 2 shows that the performance of the *forward_kernel* is potentially limited by the bandwidth available from one or more of the memories on the device. In particular, the bandwidth available to the unified cache that holds texture, global, and local data limits the kernel performance, highlighted in red. Accordingly, a series of optimizations are suggested for the memory with high bandwidth utilization, which will be explored in the future.

1. Shared memory: If possible use 64-bit accesses to shared memory and 8-byte bank mode to achieve 2 times throughput.
2. L2 cache: Align and block kernel data to maximize L2 cache efficiency.
3. Unified cache: Reallocate texture data to shared or global memory.
4. Device memory: Resolve alignment and access pattern issues for global loads and stores.
5. System memory: Make sure performance critical data is placed in device or shared memory.

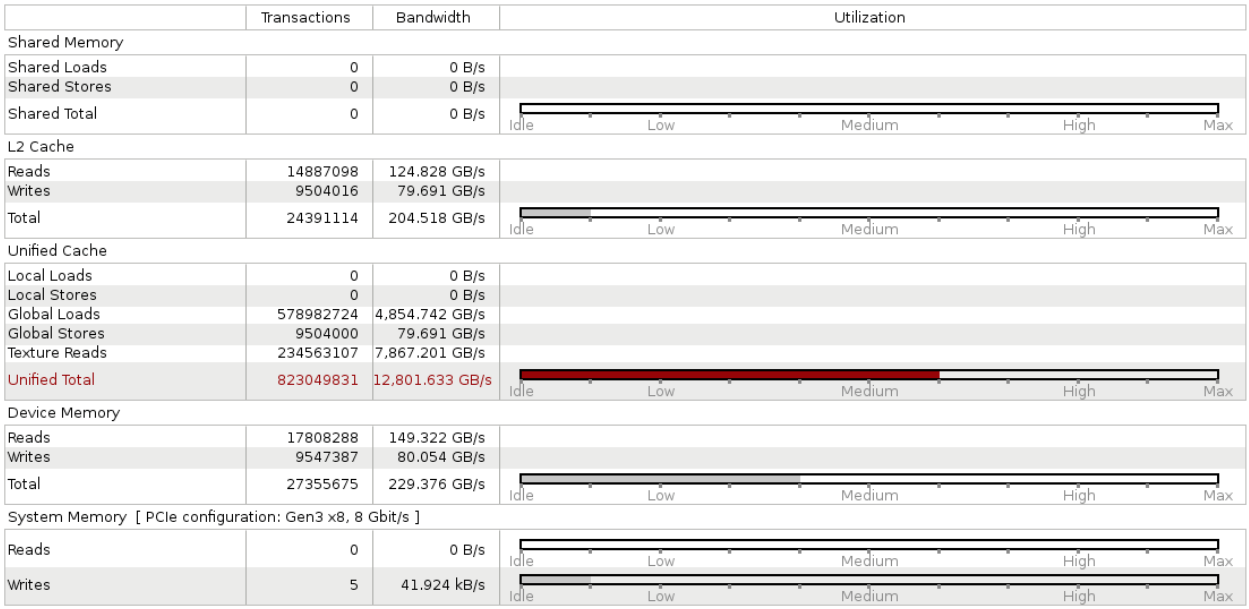

Figure 2: Memory bandwidth used by the *forward_kernel* for the various types of device memory.

3.2.3 Latency analysis

From the profiling analysis, the kernel's block size, register usage, and shared memory usage allow the kernel to fully utilize all warps on the GPU.

3.2.4 Comparing GPU implementation to CPU implementation

We also observed that the performance using GPU is better as compared to the CPU implementation (from Milestone 2). Previously, the execution time was 2 min 38 sec whereas, with the current GPU kernel implementation the execution time is 4.03 sec. This is a 97.5% increase. We hope to improve the kernel code with the suggested optimizations in future milestone deliverables.

MileStone 2: Due October 29, 2018

2.1 List whole program execution time

```
157.56user 4.42system 2:31.72elapsed 106%CPU
```

2.2 List Op times

```
Op Time: 28.737308
Op Time: 118.727850
Correctness: 0.8171 Model: ece408
```

Milestone 1: Due October 24, 2018

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

Top 10 kernels are as below:

1. volta_scudnn_128x32_relu_interior_nn_v1
2. Implicit_convolve_sgemm
3. volta_sgemm_128x128_tn
4. activation_fw_4d_kernel
5. pooling_fw_4d_kernel
6. MapPlanLargeKernel
7. SoftmaxKernel
8. MapPlanKernel
9. volta_sgemm_32x32_sliced1x4_tn
10. computeOffsetsKernel

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

Top 10 CUDA API calls are as below:

1. cudaStreamCreateWithFlags
2. cudaMemGetInfo
3. cudaFree
4. cudaEventCreateWithFlags
5. cudaMemcpy2DAsync
6. cudaFuncSetAttribute
7. cudaStreamSynchronize
8. cudaMalloc
9. cudaGetDeviceProperties
10. cudaMemcpy

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

Kernels are programmer defined functions, while API calls are built-in.

1.4 Show 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}
```

1.5 List program run time

```
19.48user 4.09system 0:13.30elapsed 177%CPU
```

1.6 Show 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}
```

1.7 List program run time

```
4.05user 2.67system 0:04.63elapsed 145%CPU
```