ECE 408 Course Project Report

Team Name: cudnn_think_of_one
School Affiliation: UIUC

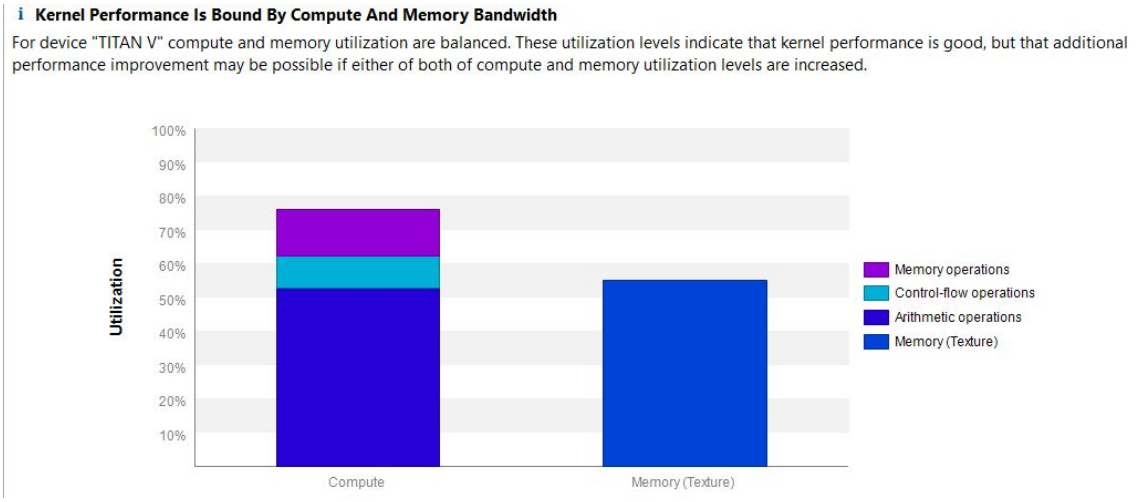
Team Members:
Ayush Agarwal (ayusha4)
Shivam Bharuka (bharuka2)
Vandana Kulkarni (vandana2)

Milestone 3

Dataset	Correctness	Op Time 1 (s)	Op Time 2 (s)	User + System Time (s)
100	0.85	0.000592	0.001602	6.49
1000	0.827	0.005725	0.015483	6.60
10000	0.8171	0.056734	0.139802	6.61

Nvprof was used to profile the dataset with 100 images to get an overview of the kernels. The following properties were studied to determine performance limiting factors:

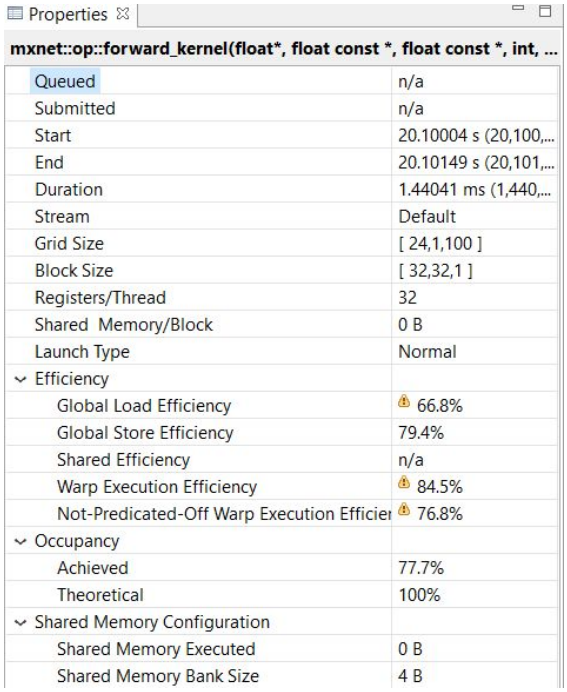
1. Global memory efficiency
2. Occupancy
3. Thread divergence
4. PC sampling



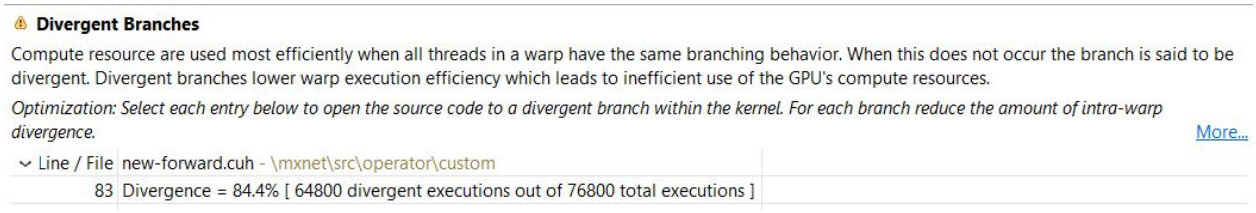
Since the current kernel is a naive GPU implementation of convolution, the aim of this exercise was to understand the different properties that can be observed and correspondingly performance optimizations can be targeted.

The following observations were made for the second instance of the forward kernel:

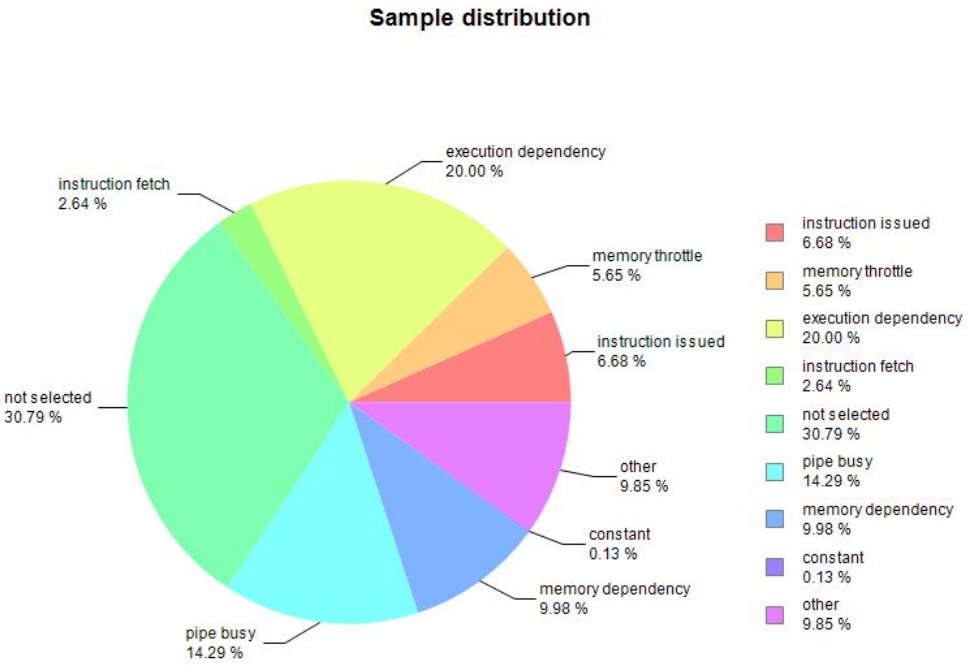
1. Global load and store efficiencies were 66.8% and 79.4% respectively. There is room for optimization in the way memory is accessed.
2. While the theoretical occupancy is 100%, only 77.7% occupancy was actually achieved, providing room for optimization here as well.
3. In this milestone we haven't implemented shared memory optimization. Hence we observe the shared efficiency is n/a and the shared memory executed is 0B as shown in the figure. Hence there is room for improvement and increase in the performance using shared memory for optimizing the convolution layers.
4. We also observe that the duration of execution of kernel is 1.44041 ms as shown in the figure.



5. The current implementation shows 84.8% divergence. Defining better thread blocks will help alleviate this problem and a boost in performance is expected.



6. PC sampling was studied to understand the distribution of time spent by the kernel in different operations like memory operations, execution operations, and so on. Since it is well distributed, the kernel is performing equally good (or bad) in each of the operations.



Milestone 2

Program execution time:

```
133.47user 4.61system 2:07.56elapsed
Program run time: 138.08 s
```

Op Times:

```
Op Time: 21.291906 s
Op Time: 101.988109 s
```

Milestone 1

1. Kernels that collectively consume more than 90% of the program time

36.82% [CUDA memcpy HtoD]

22.74% volta_scudnn_128x32_relu_interior_nn_v1

20.76% void cudnn::detail::implicit_convolve_sgemm<float, 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, float*, cudnn::detail::implicit_convolve_sgemm<float, float, int=1024, int=5, int=5, int=3, int=3, int=3, int=1, bool=1, bool=0, bool=1>*, kernel_conv_params, int, float, float, int, float, float, int, int)

7.39% volta_sgemm_128x128_tn

7.25% 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*)
32% void cudnn::detail::pooling_fw_4d_kernel<float, float, cudnn::detail::maxpooling_func<float, cudnnNanPropagation_t=0>, int=0, bool=0>(cudnnTensorStruct, float const *, cudnn::detail::pooling_fw_4d_kernel<float, float, cudnn::detail::maxpooling_func<float, cudnnNanPropagation_t=0>, int=0, bool=0>, cudnnTensorStruct*, cudnnPoolingStruct, float, cudnnPoolingStruct, int, cudnn::reduced_divisor, float)

0.52% void mshadow::cuda::MapPlanLargeKernel<mshadow::sv::saveto, int=8, int=1024, shadow::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, int)

0.07% 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)

0.06% 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)

0.03% volta_sgemm_32x32_sliced1x4_tn
```

2. CUDA API calls that collectively consume more than 90% of the program time

```
38.66%  cudaStreamCreateWithFlags
34.05%  cudaMemGetInfo
21.64%  cudaFree
1.74%  cudaFuncSetAttribute
1.33%  cudaMalloc
1.10%  cudaMemcpy2DAsync
0.85%  cudaStreamSynchronize
0.28%  cudaEventCreateWithFlags
0.18%  cudaEventCreate
0.07%  cudaGetDeviceProperties
```

3. Difference between kernel and API calls

   Kernels are automatically loaded during initialization and stay loaded for as long as the program runs whereas with the API calls it is possible to only load modules that are currently needed or load them dynamically during runtime as well.

   Kernel functions are defined by the user to run computation on a GPU device called by the host using the __global__ declaration whereas API calls are defined by the CUDA library to perform predefined functions.

   Kernel is executed N time parallelly where N is the total number of threads whereas API calls are executed once.

4. Output of rai running MXNet on the CPU

```
EvalMetric: {'accuracy': 0.8177}
20.01user 4.13system 0:13.60elapsed 177%CPU (0avgtext+0avgdata
5954888maxresident)k
0inputs+2856out
puts (0major+1585429minor)pagefaults 0swaps
```

5. CPU program run time

```
20.01user 4.13system 0:13.60elapsed
Program run time : 24.14 s
```

6. Output of rai running MXNet on the GPU

```
EvalMetric: {'accuracy': 0.8177}
4.00user 2.59system 0:04.56elapsed 144%CPU (0avgtext+0avgdata
2841584maxresident)k
8inputs+1712outputs (0major+704309minor)pagefaults 0swaps
```

7. GPU program run time

```
4.00user 2.59system 0:04.56elapsed
Program run time: 6.59 s
```