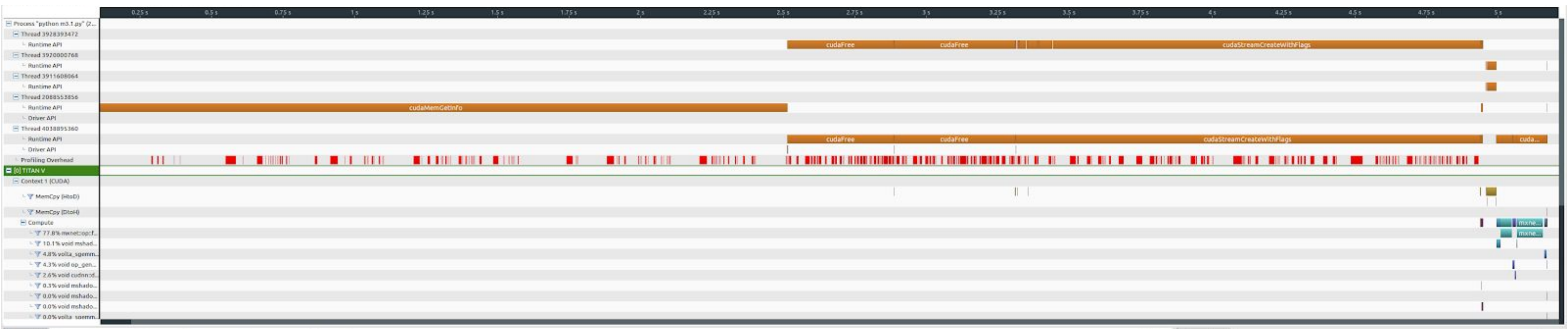
<u>Team: Undecided</u>

On Campus
Members: Shrey Chowdhary (shreyc2), Vijay Klein (vijaygk2), Alex Krysl (krysl2)

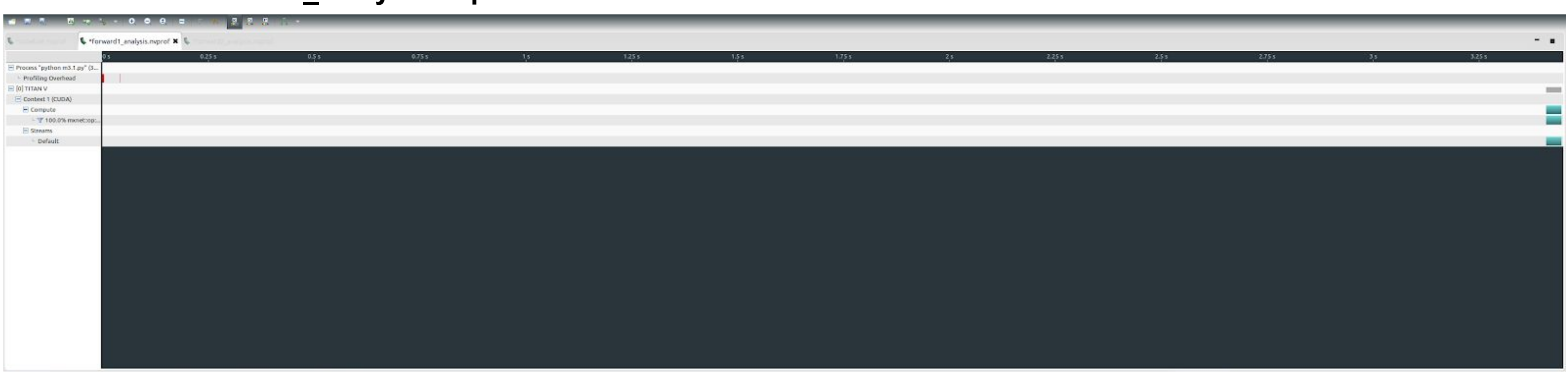
<u>Milestone 3</u>

Report: demonstrate nvprof profiling the execution

timeline.nvprof



forward1_analysis.nvprof



forward2_analysis.nvprof

According to NVVP we have low compute utilization meaning many of the SM were idling. Looking at the timeline trace it seems like computation on the GPU is taking very little time relative to the CUDA API calls on host code. Further, within our kernel, we also see a low global memory load efficiency. This metric is the number of bytes requested divided by the number of bytes transferred from device memory. The actual value for this metric was 62.5 percent which indicates that 37.5 percent of the memory coming out of the DRAM bursts was not requested and likely not used. Since memory is typically the slowest piece in the architecture, increasing this value certainly would be a viable optimization method in the future. Finally, we also noticed a fairly high divergence rate with 22.9 percent of threads diverging on the following line.

```
    if (h >= H_out || w >= W_out) {
        return;
    }
```

This source of this issue is the dimensions of the Grid/Blocks which obviously should not really be an issue for this milestone as we are prioritizing functionality, but again, in the future, this is certainly an area where we can improve. Still, this check is more or less required in some form, as we do not want to be writing in a thread that "falls out" of the bounds of the array.

Milestone 2

List of all kernels that collectively consume more than 90% of the program time.

Time (%)	**Name**
31.97%	**[CUDA memcpy HtoD]**
17.85%	**volta_scudnn_128x64_relu_interior_nn_v1**
17.16%	**volta_gcgemm_64x32_nt**
8.56%	**fft2d_c2r_32x32<float, bool=0, 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*, int2, int, int)**
7.79%	**volta_sgemm_128x128_tn**

6.50%	**op_generic_tensor_kernel<int=2, float, float, float, int=256, cudnnGenericOp_t=7, cudnnNanPropagation_t=0, cudnnDimOrder_t=0, int=1>(cudnnTensorStruct, float*, cudnnTensorStruct, float const *, cudnnTensorStruct, float const *, float, float, float, float, dimArray, reducedDivisorArray)**
5.70%	**void fft2d_r2c_32x32<float, bool=0, unsigned int=0, bool=0>(float2*, float const *, int, int, int, int, int, int, int, int, int, cudnn::reduced_divisor, bool, int2, int, int)**

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

Time (%)	Name
42.14%	**cudaStreamCreateWithFlags**
33.26%	**cudaMemGetInfo**
21.03%	**cudaFree**

Difference between kernels and API calls

Both kernels and API calls potentially interact with the device/GPU in the system; however, the way in which they do so is the key distinction.

Essentially, an API call is a function provided in the driver (run on the host) that can interact with the device, some examples of which are allocating and freeing device memory (cudaMalloc and cudaFree, respectively). The key note here is that while the code interacts with the device's hardware, the code itself is running on the host, and there are no threads spawning/computations being done on the GPU device (i.e. the majority of the code/functionality is occurring on the host, rather than the device).

This is in contrast to a kernel, where the kernel is called from the host, but directly spawns threads on the GPU device, which all then execute the kernel code <u>on the device</u>, in parallel.

Report: Show output of rai running MXNet on the CPU
Loading fashion-mnist data... done
Loading model... done
New Inference
EvalMetric: {'accuracy': 0.8154}

Report: List program run time
17.25user 4.45system 0:08.95elapsed 242%CPU (0avgtext+0avgdata 6045864maxres
ident)k
0inputs+2824outputs (0major+1600460minor)pagefaults 0swaps

So, 8.95 wall-clock seconds for a run on the CPU.

Report: Show output of rai running MXNet on the GPU
Loading fashion-mnist data... done
Loading model... done
New Inference
EvalMetric: {'accuracy': 0.8154}

Same accuracy makes sense and is worth noting.

Report: List program run time
5.20user 3.19system 0:04.63elapsed 181%CPU (0avgtext+0avgdata 2975644max
resident)k

So, 4.63 wall-clock seconds for a run on the GPU. Also makes sense this would run faster.
CPU whole program execution time
86.44user 7.99system 1:16.76elapsed
So, 1:16.76 wall-clock minutes for a run on the CPU.
CPU Op Times
Op Time: 11.020163
Op Time: 62.090408