Name: *Trent Zhang*
NetID: *zz90*
Section: *ECE 408 AL1*

ECE 408/CS483 Milestone 2 Report

1. Show output of rai running Mini-DNN on the basic GPU convolution implementation for batch size of 1k images. This can either be a screen capture or a text copy of the running output. Please do not show the build output. (The running output should be everything including and after the line "*Loading fashion-mnist data...Done*").

```
Loading model...Done
Conv-GPU==
Layer Time: 99.5622 ms
Op Time: 5.10361 ms
Conv-GPU==
Layer Time: 90.2787 ms
Op Time: 16.8827 ms

Test Accuracy: 0.886


real    0m10.201s
user    0m9.799s
sys     0m0.372s
```

2. For the basic GPU implementation, list Op Times, whole program execution time, and accuracy for batch size of 100, 1k, and 10k images.

Batch Size	Op Time 1	Op Time 2	Total Execution Time	Accuracy
100	*2.88873 ms*	*3.97483 ms*	*0m1.271s*	*0.86*
1000	*5.10361 ms*	*16.8827 ms*	*0m10.201s*	*0.886*
10000	*108.997 ms*	*352.769 ms*	*1m41.396s*	*0.8714*

3. List all the kernels that collectively consumed more than 90% of the kernel time and what percentage of the kernel time each kernel did consume (start with the kernel that consumed the most time, then list the next kernel, until you reach 90% or more).

CUDA Kernel Statistics (nanoseconds)

Time(%)	*Total Time*	*Instances*	*Average*	*Minimum*	*Maximum*	*Name*
100.0	*461649332*	*2*	*230824666.0*	*161020454*	*300628878*	*conv_forward_kernel*

4. List all the CUDA API calls that collectively consumed more than 90% of the API time and what percentage of the API time each call did consume (start with the API call that consumed the most time, then list the next call, until you reach 90% or more).

CUDA API Statistics (nanoseconds)

Time(%)	*Total Time*	*Calls*	*Average*	*Minimum*	*Maximum*	*Name*
51.2	*1604593317*	*10*	*160459331.7*	*21922*	*632359372*	*cudaMemcpy*
28.8	*903978445*	*8*	*112997305.6*	*93667*	*897473663*	*cudaMalloc*
19.3	*606772901*	*10*	*60677290.1*	*1220*	*301588828*	*cudaDeviceSynchronize*

5. Explain the difference between kernels and CUDA API calls. Please give an example in your explanation for both.

How Kernels differ from CUDA API

1. When Kernels called, it execute N times in parallel by N different CUDA threads, as opposed to only once like regular C++ functions.
2. Kernel is defined using the __global__ declaration specifier
3. Number of CUDA threads that execute that kernel for a given kernel call is specified using a new <<<...>>>execution configuration syntax
4. The CUDA API provides an additional level of control by exposing lower-level concepts such as CUDA contexts - the analogue of host processes for the device - and CUDA modules - the analogue of dynamically loaded libraries for the device

Examples

1. *Kernels:*

```
// Kernel definition
__global__ void VecAdd(float* A, float* B, float* C)
{
   int i = threadIdx.x;
   C[i] = A[i] + B[i];
}

int main()
{
   ...
   // Kernel invocation with N threads
   VecAdd<<<1, N>>>(A, B, C);
```

2. *CUDA API:*

```
cudaMalloc((void **)& *device_x_ptr,  B * C * H * W * sizeof(float*));
  cudaMalloc((void **)& *device_y_ptr,  B * M * H_out * W_out * sizeof(float*));
   cudaMalloc((void **)& *device_k_ptr, C * M * K * K * sizeof(float*));

   cudaMemcpy(*device_x_ptr, host_x,  B * C * H * W * sizeof(float),
cudaMemcpyHostToDevice);
   cudaMemcpy(*device_y_ptr, host_y,  B * M * H_out * W_out * sizeof(float),
cudaMemcpyHostToDevice);
   cudaMemcpy(*device_k_ptr, host_k, C * M * K * K * sizeof(float),
cudaMemcpyHostToDevice);
```

6. Show a screenshot of the GPU SOL utilization