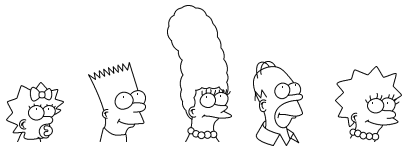
046278
Accelerators and Accelerated Systems
Assigment 1

Avraham Ayaso - 305036352, Yonatan Nakonechny - 200752061

April 26, 2020



1 Knowing the system

1.1 Report the CUDA version on your machine

release 10.2, V10.2.89

1.2 Report the GPU name

GeForce RTX 2080 SUPER

1.4 Examine the output proudly and report the number of SMs

There are 48 SM's in the GPU

3 Implement a task serial version

3.3 Explain why atomicAdd is required

atomicAdd is required since different threads will access the shared data (histogram) at unknown time and order. Therefore, in order to prevent race conditions, the shared data needs to be accessed serially.

3.4 Define the state needed for the task serial in the "task_serial_context" struct

The state needed is:

a. a pointer to an array of pointers, each points to a single input image.
b. a pointer to an array of pointers, each points to a single output image.

3.7 Use a reasonable number of threads when you invoke the kernel. Explain your choice

We've chosen to use 1024 threads since this is the maximum number of threads per block, and we couldn't use more than that because it would require using more than 1 block and therefore we wouldn't be able to use shared memory for histogram and syncthreads (for efficiency).

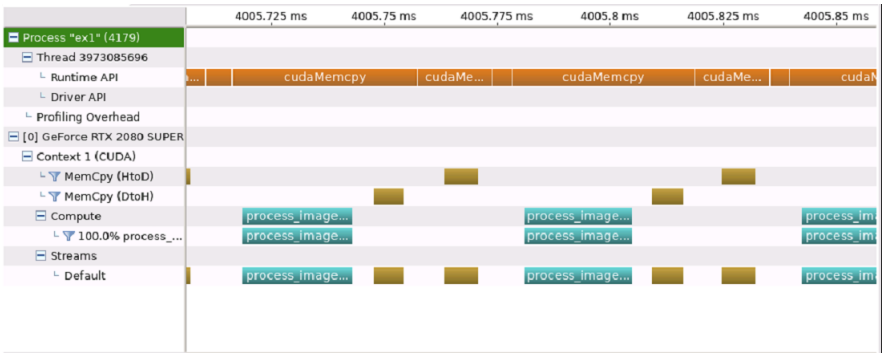
3.8 Report the total run time and the throughput (images/sec). memcpy-s should be included in this measurment

```
=== Randomizing images ===
total time 2562.107399 [msec]

=== CPU ===
total time 718.105308 [msec]

=== GPU Task Serial ===
total time 504.883753 [msec]  distance from baseline 0 (should be zero)
```

The total serial runtime on the GPU is 504.88ms, therefore the throughput is:

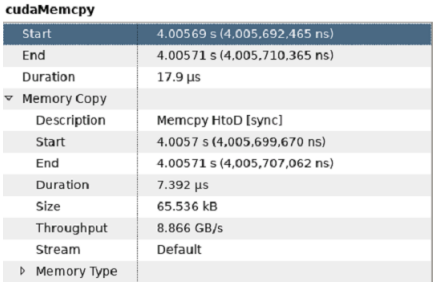
$$\frac{10000}{0.50488} = 19,806.69 \left[\frac{images}{sec}\right]$$

3.9 Use NVIDIA's visual profiler to examine the execution diagram of your code. Attach a clear screenshot to the report showing at least 2 kernels with their memory movements



3.10 Choose one of the memcpy's from CPU to GPU, and report its time (as appears in NVIDIA's visual profiler)

The memcpy duration is ~17.9us.



4 Implement a bulk synchronous version

4.2 Define the state needed for the bulk synchronous version in the "gpu_bulk_context" struct

The state needed is:

a. a single pointer to an array of all the input images.
b. a single pointer to an array of all the output images.

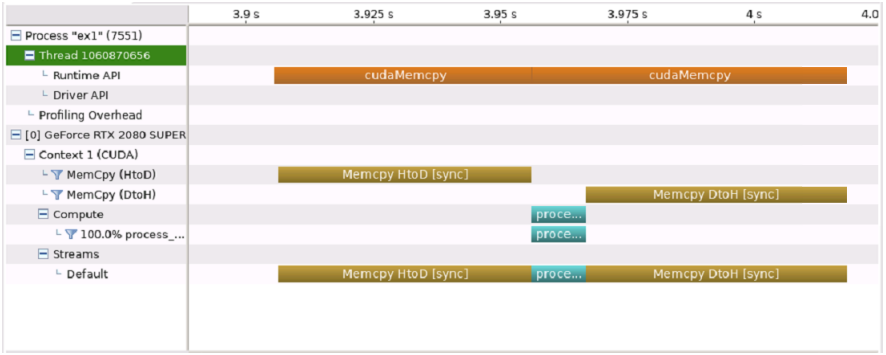
4.5 Report the execution time, and speedup compared to (3)

```
=== Randomizing images ===
total time 2562.107399 [msec]

=== CPU ===
total time 718.105308 [msec]

=== GPU Task Serial ===
total time 504.883753 [msec]  distance from baseline 0 (should be zero)

=== GPU Bulk ===
total time 110.977949 [msec]  distance from baseline 0 (should be zero)
```

The execution time is 110.98sms, therefore the speedup is:

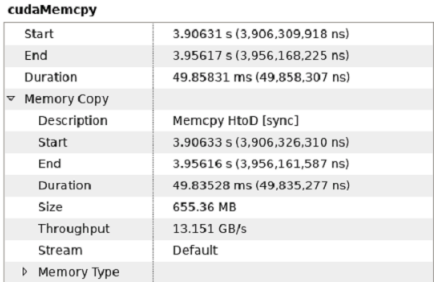
$$\frac{504.88}{110.98} = 4.55$$

4.6 Attach a clear screenshot of the execution diagram from NVIDIA's visual profiler



4.7 Check the times CPU-to-GPU memcpy in NVIDIA's visual profiler. Compare it to the memcpy time you measured in (2). Does the time grow linearly with size of the data being copied?

The memcpy duration now is ~49.86ms, which is indeed larger than the ~17.9us measured in the serial execution, but the growth is not linear with the size of the data being copied. As can be seen in our measurment, the duration was multiplied by ~3,000 (and not 10,000).



cudaMemcpy

Start	3.90631 s (3,906,309,918 ns)
End	3.95617 s (3,956,168,225 ns)
Duration	49.85831 ms (49,858,307 ns)
▽ Memory Copy	
Description	Memcpy HtoD [sync]
Start	3.90633 s (3,906,326,310 ns)
End	3.95616 s (3,956,161,587 ns)
Duration	49.83528 ms (49,835,277 ns)
Size	655.36 MB
Throughput	13.151 GB/s
Stream	Default
▷ Memory Type	