**Accelerators and Accelerated systems**

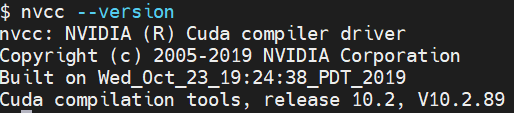
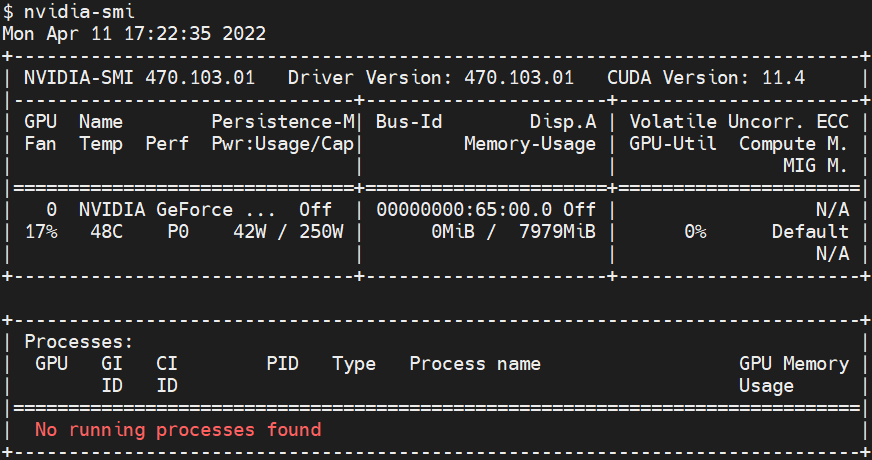
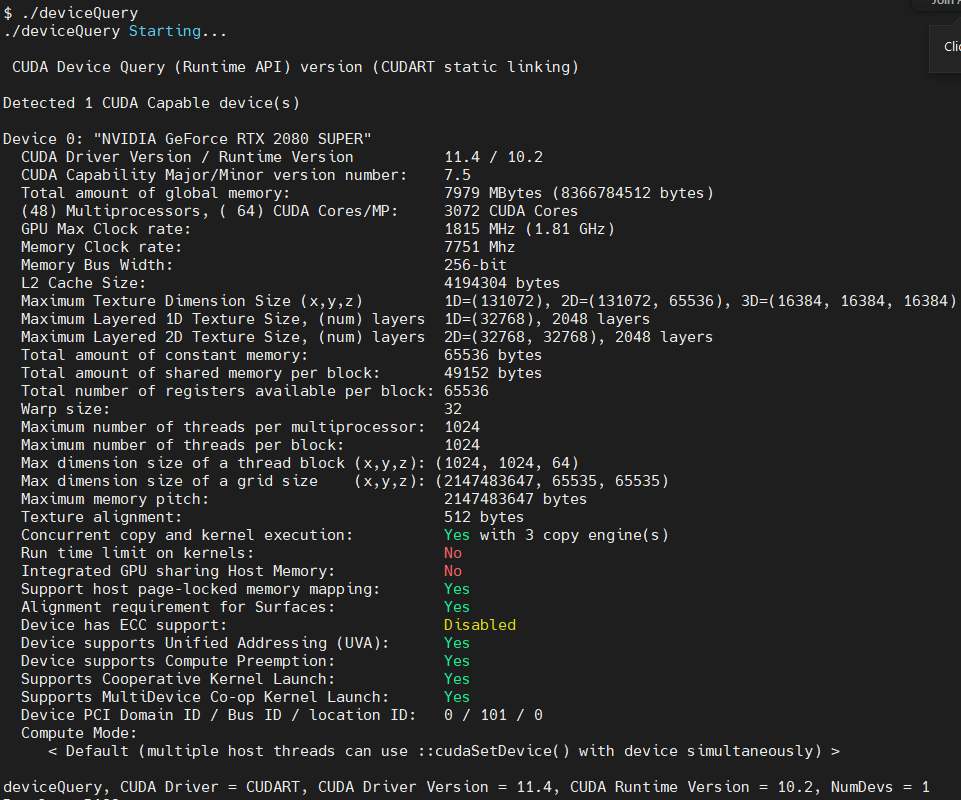
**046278**

**Authors:**

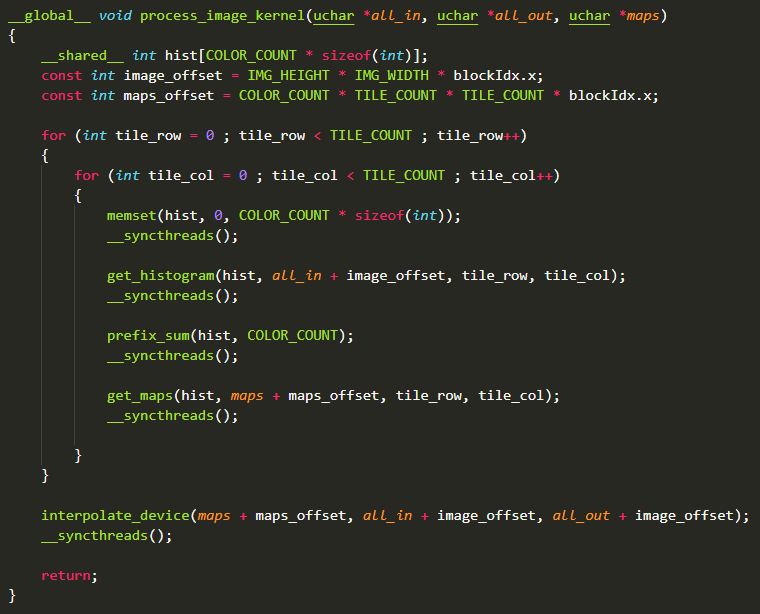
**Chris Shakkour, 208157826, Christian.s@campus.technion.ac.il**

**Itay Snir, 208726992, itaysnir@campus.technion.ac.il**

**Tasks**

1. Knowing the system:
2. Cuda Version: 
3. GPU Name:  
   
4. GPU:   
   
5. Number of SMs: **48** SMs (multiprocessors), each containing **64** CUDA Cores.

2. In **ex1.cu**

3. **Task Serial Version**  
  
a. Implementation of the kernel (can be found within **ex1.cu**):  


b. The histogram calculation method contains an atomic operation.  
**atomicAdd** is required due to the race that may occur between different threads:

The “increment by one” operation (eg, histogram[color]++) is actually made of three operations: **read** of the previous value from memory, **arithmetic incrementation,** and finally a **write** of the updated value into memory. For example, two threads may race - thread #1 reads an old value but hasn't written the updated value yet. In the meantime, thread #2 does all three operations (to the same memory location). Once thread #1 completes the write operation, the memory location contains an invalid value, that doesn’t takes into account the addition that was made by thread #2. Atomic operations ensure all of the “sub operations” are made all at once, without any interfering thread in the meanwhile.

c. The global memory accesses are coalesced mainly because of the important need to reduce memory access count. Since global memory accesses are very costly, an efficient access mechanism is mandatory for decent performance. Instead of fetching a single byte per access, 32 Bytes fetching is much more efficient, assuming all the fetched data will be processed soon (that's why the continuous access by the various threads is required).

d. In **ex1.cu**

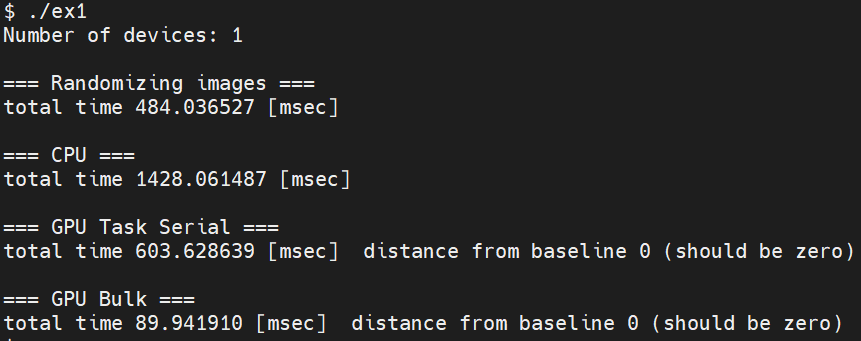
e. In **ex1.cu**

f. In **ex1.cu**

g. For the serial version, we were asked to invoke the kernel with a single thread block.  
This single thread block runs **256 threads**. The main reasoning is simplicity, as the maps calculation function (**get\_maps()**) requires exactly 256 threads.

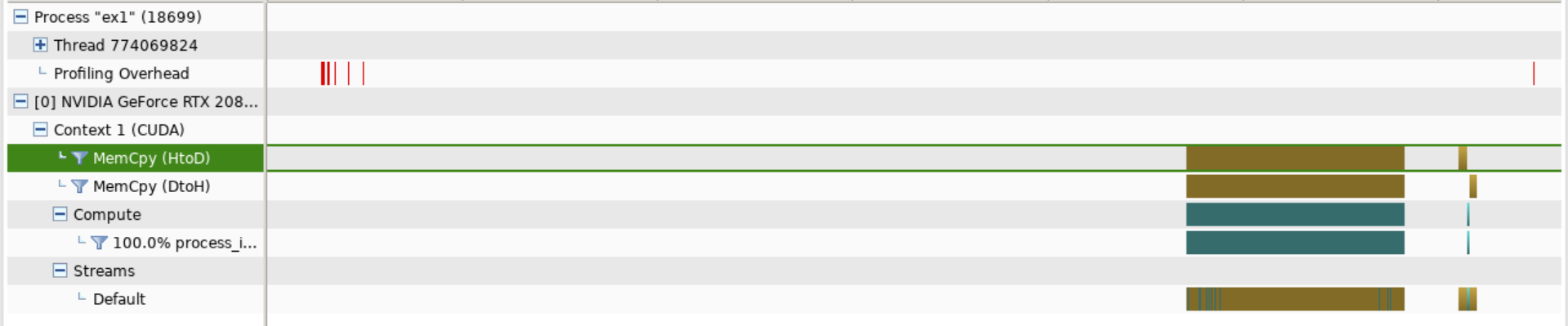
We can further increase the amount of threads (up to 1024), and that will indeed improve the performance of the histogram and the prefix sum calculations.

h. We got the following results:

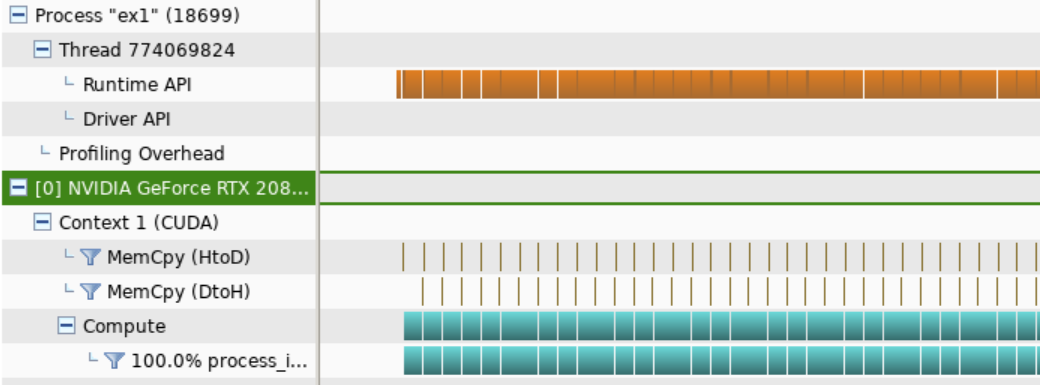


As we can see, the total run time for the GPU serial processing is **603.63 msec**.   
Since **N\_IMAGES = 1000**, the throughput is about **1657 images / sec.**

i. Execution diagram:



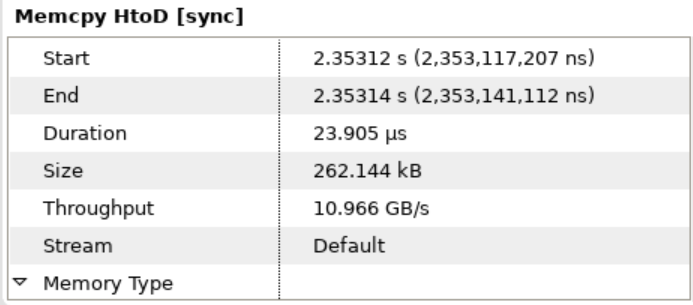
Zoom in …



j. Memcpy from CPU to GPU time(Host to Device):

We will choose the first memcpy from CPU to GPU, following task duration.





4. **Bulk Synchronous Version**

a. In **ex1.cu**

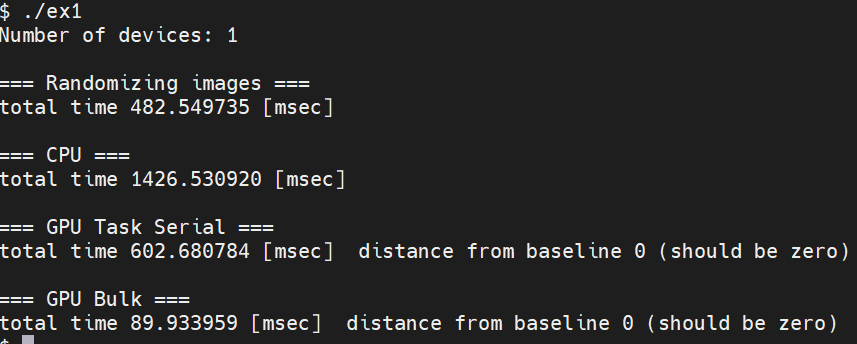
b. We’ve modified the kernel presented in (3), so that it can easily support bulk execution for multiple thread blocks. In **ex1.cu**

c. In **ex1.cu**

d. In **ex1.cu**

e. In **ex1.cu**

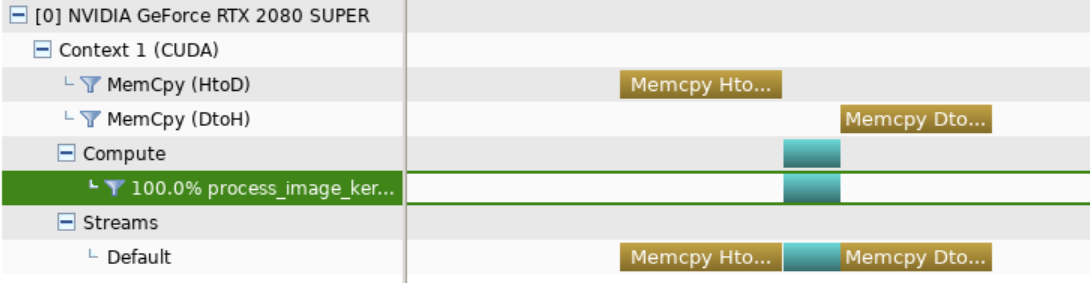
f. Another execution of the results:



As we can see, the execution time for the bulk version is **89.93 msec**.

The resulting speedup is **x6.7!** The bulk version is faster by 6.7 times.

g. Execution diagram:



h. Memcpy from CPU to GPU time(Host to Device):

