Assignment IV: Advanced CUDA

Wang, Chenyi
chenyiw@kth.se

Ahmed, Ouday
ouday@kth.se

January 7, 2024

0 Git Repository and Contributions

Git repository: https://github.com/OudayAhmed/DD2360HT23
Both group members completed the assignment task, and we arranged a meeting to discuss our solutions and write the report together.

1 Thread Scheduling and Execution Efficiency

1. 15200 wraps will be generated during the execution of the kernel, and no wraps will have control divergence. **Explain**:

   (a) The number of blocks per grid along the horizontal axis (X) is 50, and the number of blocks per grid along the vertical axis (Y) is 38. Each block has 16x16 threads, which converts to 8 wraps per block. Thus, the total wraps are 50 x 38 x 8 = 15200.

   (b) There will be a total of 6400 threads that are generated out of bounds. These 6400 threads will be divided into the last row of blocks, which means there will be 128 out-of-bounds threads in each of the 50 blocks. These 128 threads will be in exactly four wraps that all execute the else path, so there will be no divergence across these wraps.

2. There will be a total of 400 wraps that have control divergence. **Explain**: Since the image width (600 pixels) does not align perfectly with the block width (16 pixels), the last block in each row will have partially filled warps (where not all threads fulfill the (Col ¡ n) statement) which converts to 8 wraps. Since there are a total of 50 rows (blocks) in the grid vertically, the total wraps that have control divergence will be 50 x 8 = 400.

3. There will be a total of 50 (number of blocks along Y) * 8 (each block has 8 wraps) + 37 (number of blocks along X - 1) = 437 wraps that have control divergence. **Explain**:

   (a) The 50 x 8 wraps are exactly the same as the previous question

(b) Since the height of the image (799 pixels) does not align perfectly with the block height (16 pixels), one additional row will be added (50 blocks x 16 = 800, which causes one additional line). This additional row of pixel will be at the last wrap of each block, and it has control divergence. Therefore, we add 38 - 1 = 37 (minus one because the one block in the right corner has already been considered in the 400 wraps part) blocks to the final answer.

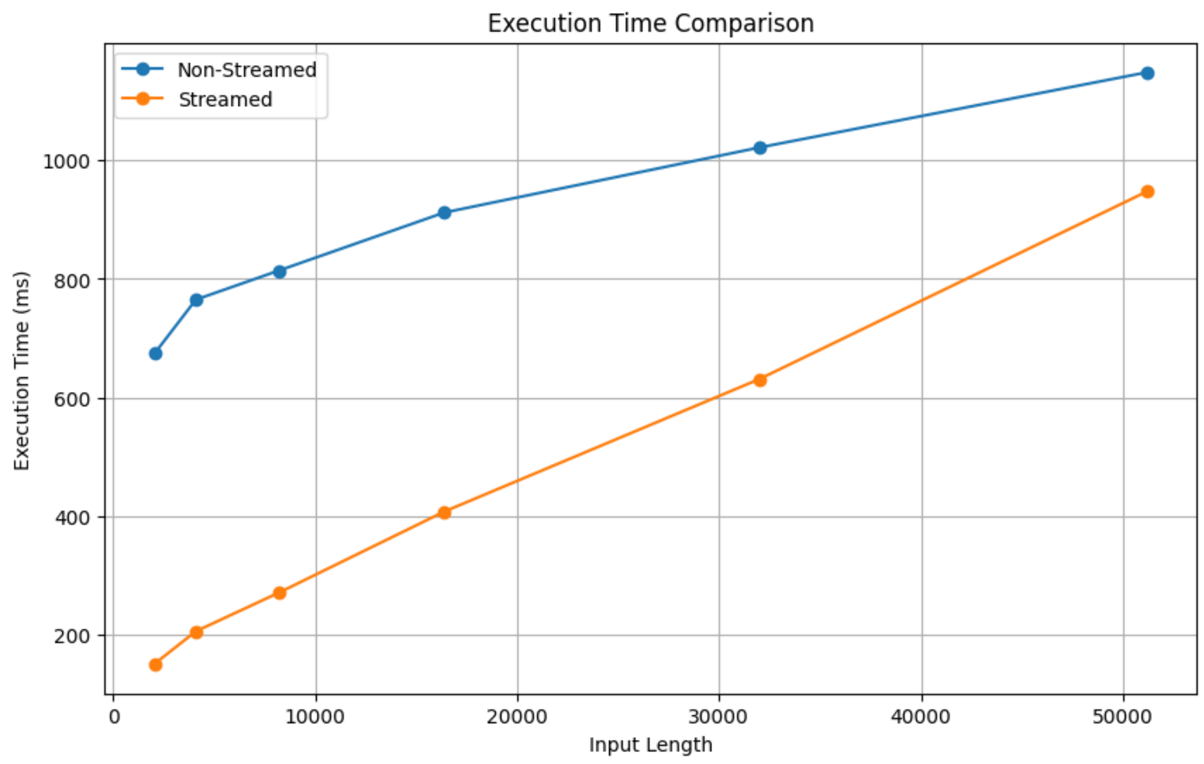


Figure 1: Execution time comparison with non-streamed version

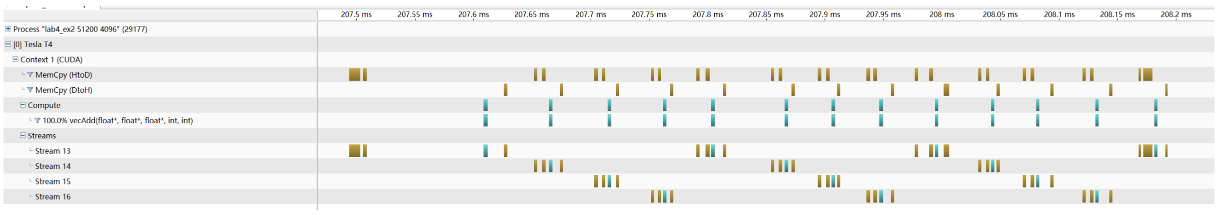


Figure 2: Visualization of the overlap of communication and computation

2 CUDA Streams

1. With a segment size of 2048, the performance gain compared with the non-streamed version is bigger when the input length is relatively small but gets lower as the input length increases. See fig 1 for results in the plot.

2. See fig 2 for the result (with input length = 51200, segment size = 4096).

3. With an input vector length of 51200, see fig 3 for plot results.

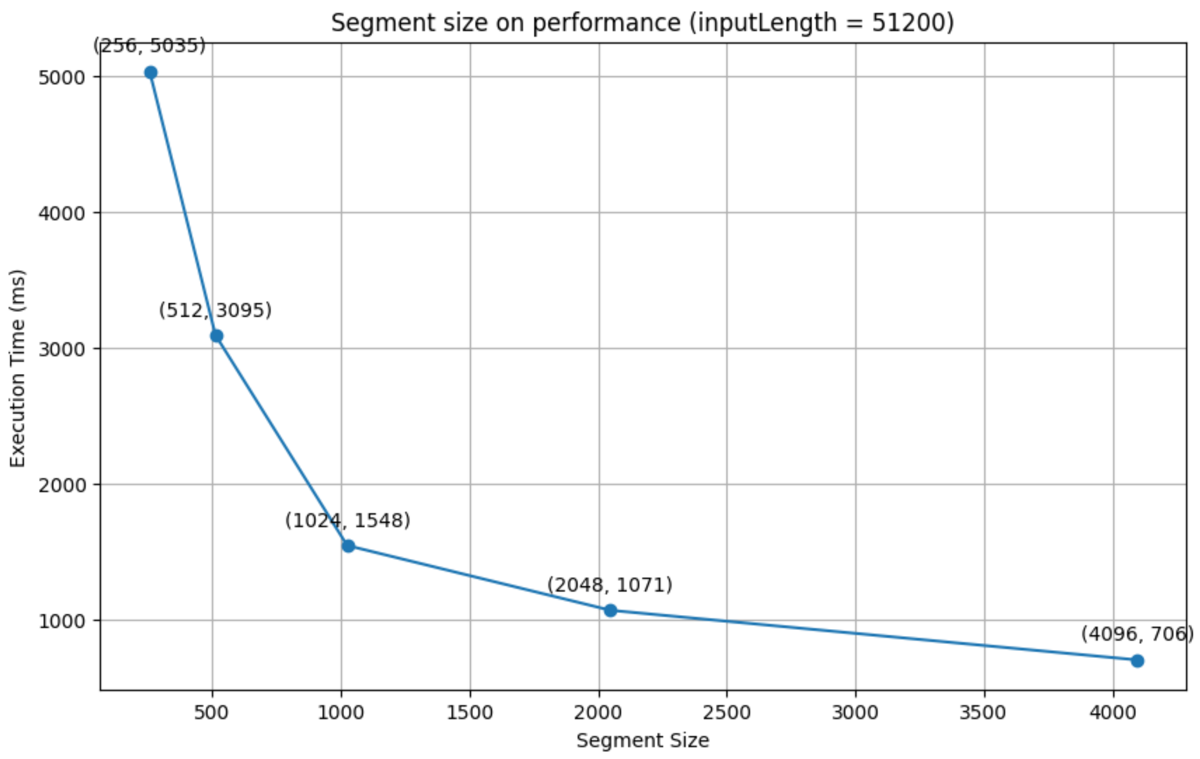


Figure 3: Segment size on performance

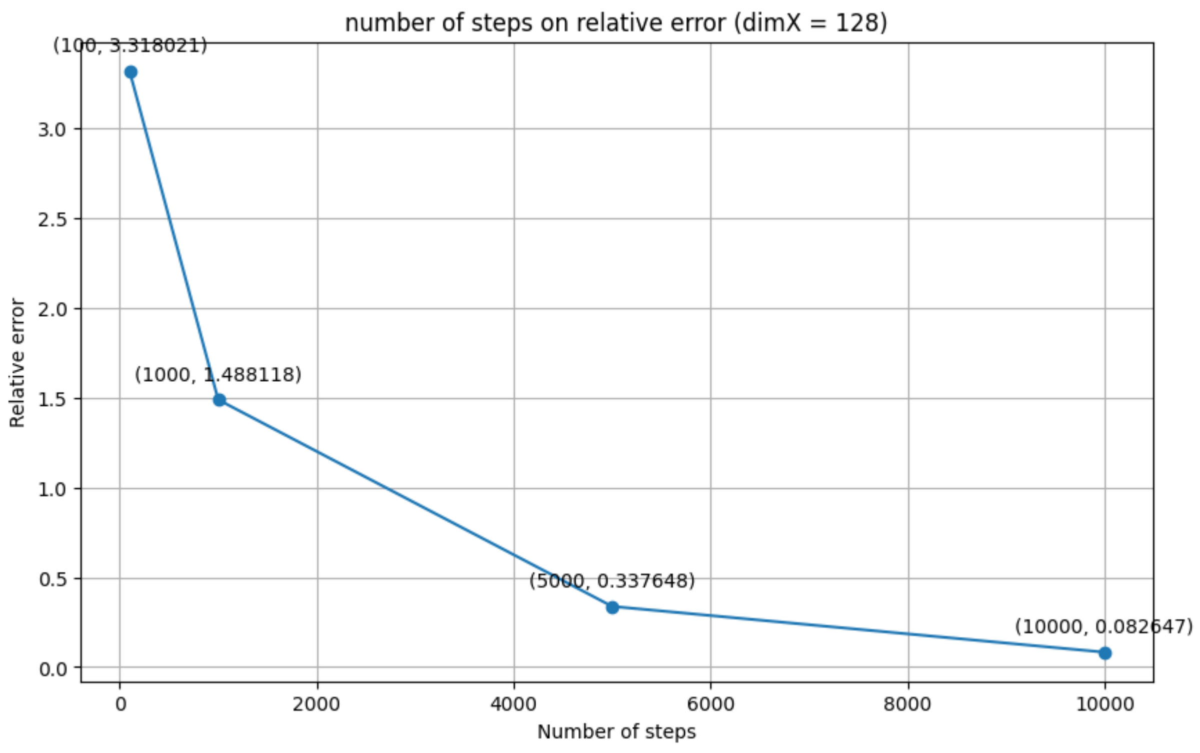


Figure 4: Number of steps on relative error

3 Heat Equations using Nvdia libraries

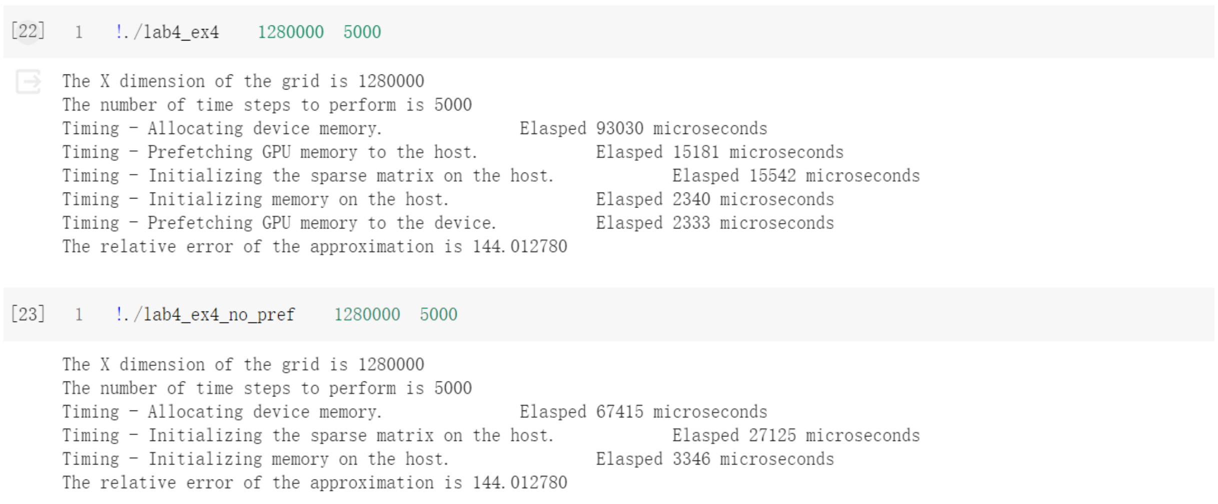
1. See fig6 for plot result. We approximate flops using this formula:

$$FLOPS = \frac{Number\ of\ Operations}{ExecutionTime(seconds)} = \frac{2NNZ}{t} \tag{1}$$

   where NNZ is the number of non-zero elements.

2. See fig 4 for plot result.

3. Without prefetching data, initializing the sparse matrix on the host takes significantly longer. See fig. 5



```
[22]  !./lab4_ex4   1280000  5000
```

```
The X dimension of the grid is 1280000
The number of time steps to perform is 5000
Timing - Allocating device memory.              Elasped 93030 microseconds
Timing - Prefetching GPU memory to the host.            Elasped 15181 microseconds
Timing - Initializing the sparse matrix on the host.            Elasped 15542 microseconds
Timing - Initializing memory on the host.               Elasped 2340 microseconds
Timing - Prefetching GPU memory to the device.          Elasped 2333 microseconds
The relative error of the approximation is 144.012780
```

```
[23]  !./lab4_ex4_no_pref   1280000  5000
```

```
The X dimension of the grid is 1280000
The number of time steps to perform is 5000
Timing - Allocating device memory.              Elasped 67415 microseconds
Timing - Initializing the sparse matrix on the host.            Elasped 27125 microseconds
Timing - Initializing memory on the host.               Elasped 3346 microseconds
The relative error of the approximation is 144.012780
```

Figure 5: Comparison with non-prefetching unified memory

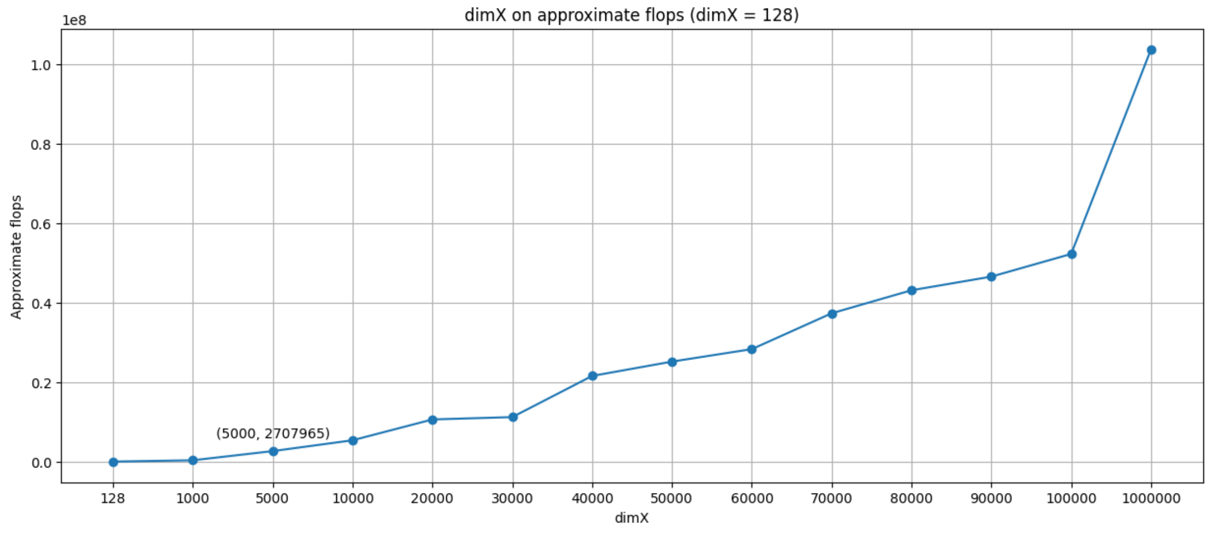


Figure 6: different dimX on approximate flops of SpMV