Assignment II: CUDA Basics I

Wang, Chenyi
chenyiw@kth.se

Ahmed, Ouday
ouday@kth.se

November 26, 2023

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 Exercise 1 - Your first CUDA program and GPU performance metrics

1. We used Google's Collab to host and run our program. To compile the code, we used the nvcc compiler.
   `!nvcc -o ex1 ex1.cu` and run the code with `!./ex1 10000`

2. In the `vecAdd` kernel, two memory reads operation and one floating-point addition operation is performed for each element of the input vectors. Therefore, for a vector of length N, the number of floating-point operations is **N**, and the number of global memory reads is **2*N**.

3. For a vector of 1024:

   (a) The number of threads per block = 512. The number of thread blocks = (1024 + 512 - 1) / 512 = 2. The total number of threads = 512 * 2 = 1024.

   (b) The achieved Occupancy is **44.14%**.

4. Increase the vector length to 131070:

   (a) Yes, it still works without making any changes.

   (b) The number of threads per block = 512. The number of thread blocks = (131070 + 512 - 1) / 512 = 256. The total number of threads = 512 * 256 = 131072.

   (c) The achieved Occupancy is **76.94%**.

5. See Fig1

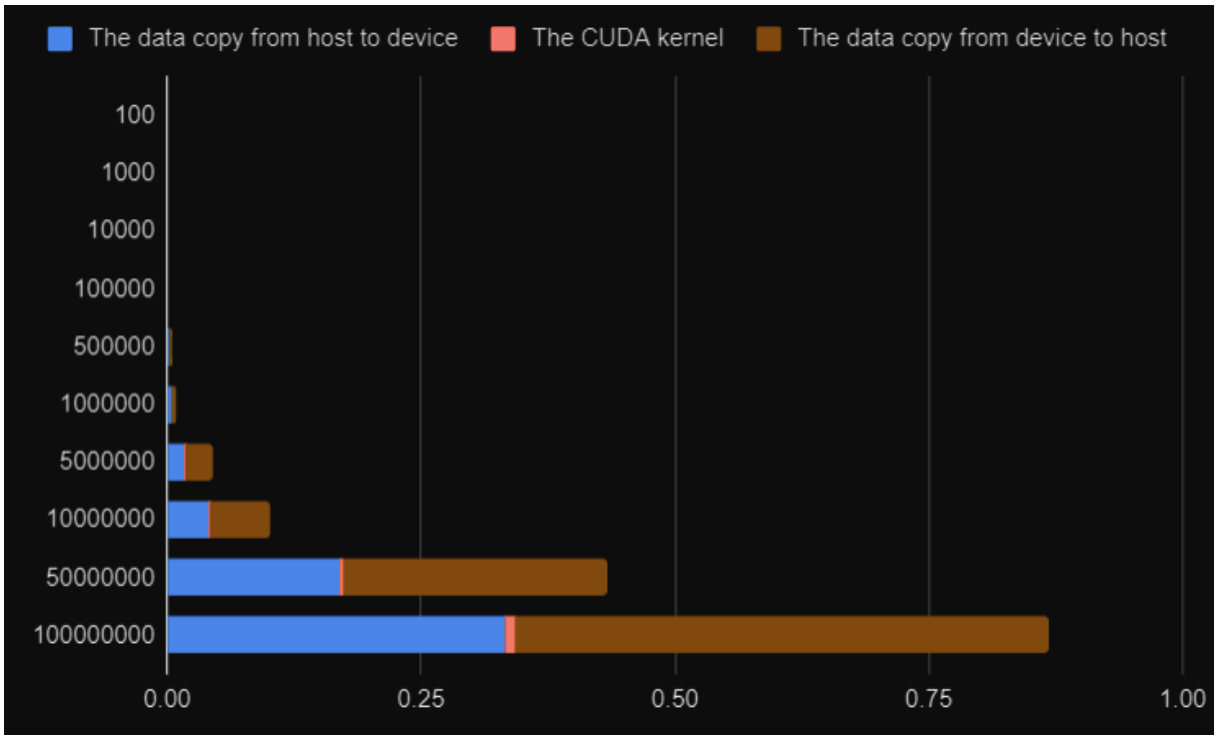

Figure 1: The time breakdown (data copy from host to device, CUDA kernel execution, and data copy from device to host) across different vector lengths.

2 Exercise 2 - 2D Dense Matrix Multiplication

1. Name three applications domains of matrix multiplication
   (a) In Computer Graphics, space transformations (e.g., scale, rotation)
   (b) In Machine Learning, doing convolutions
   (c) In Signal Processing, doing Fourier Transformations (see DFT matrix)
2. There is 2 * numAColumns * numARows * numBColumns floating operations are being performed in matrix multiply kernel.
3. There is 2 * numAColumns * numARows * numBColumns global memory reads are being performed by the kernel.
4. For a matrix A of (128x128) and B of (128x128):
   (a) The number of threads per block = 8 * 8 = 64. The number of thread blocks = 16 * 16 = 256. The total number of threads = 256 * 64 = 16384.
   (b) The achieved Occupancy is **37.29%**.
5. For a matrix A of (511x1023) and B of (1023x4094):
   (a) The program is still working. We just changed the block size to 32 * 32.
   (b) The number of threads per block = 32 * 32 = 1024. The number of thread blocks = 128 * 2048 = 256. The total number of threads = 1024 * 256= 2097152.

(c) The achieved Occupancy is **98.01%**.

6. We can see that the time taken for data copy from host to device increases as the size of the data grows. The time for data copy from device to host also increases with the size of the data, but the values are generally smaller than those for the host-to-device copy. The stacked bar chart indicates also that the majority of the execution time is consumed by the CUDA kernel, in comparison to the data copy operations. See Fig2.

7. The stacked bar chart for float data types indicates reduced execution times for both data copy and CUDA kernel operations compared to double data types. See Fig3.

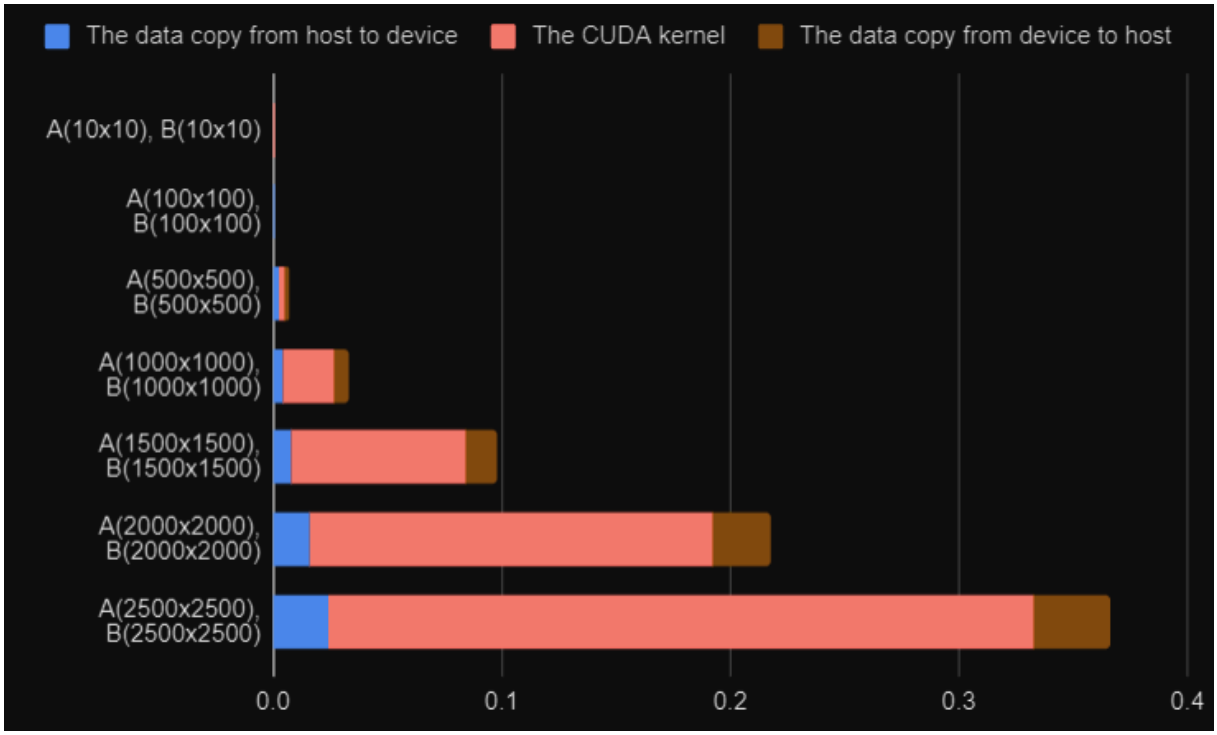

Figure 2: The time breakdown (data copy from host to device, CUDA kernel execution, and data copy from device to host) across different matrix sizes.

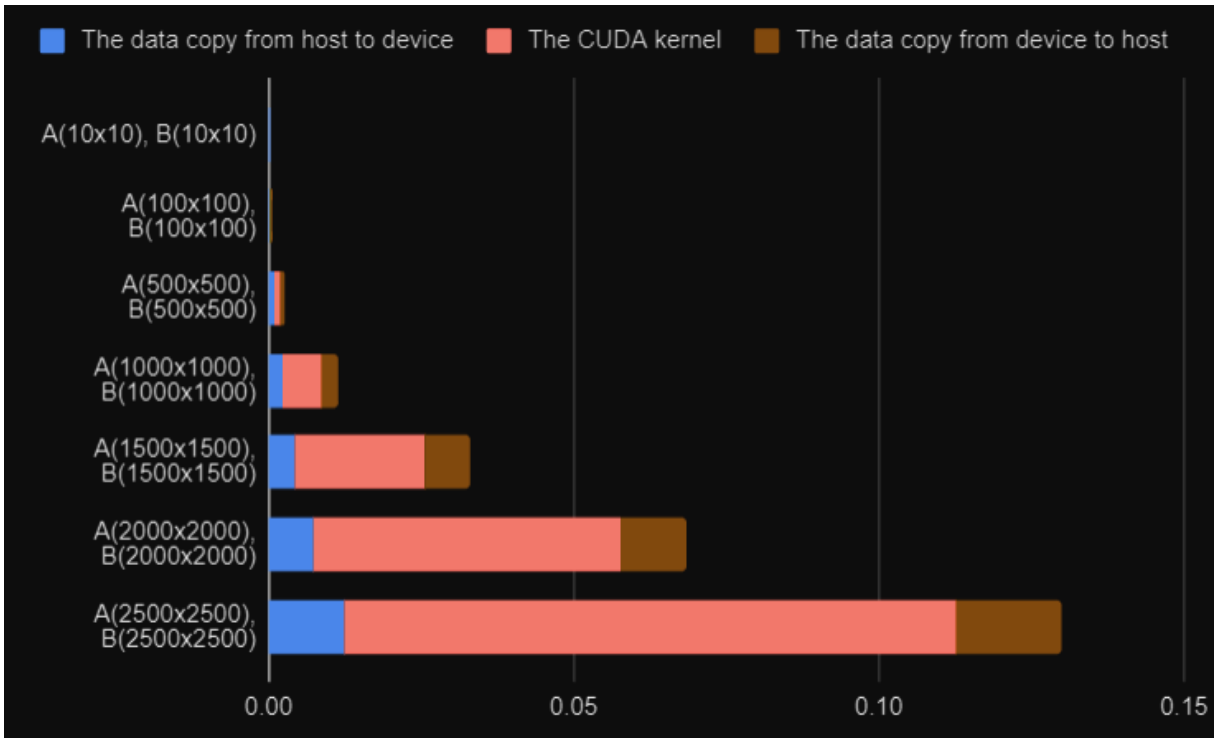

Figure 3: The time breakdown (data copy from host to device, CUDA kernel execution, and data copy from device to host) across different matrix sizes.