Assignment II

Adam Orucu

November 26, 2023

Url to code: `https://github.com/adamorucu/gpu-programming-course`

Exercise 1

1. **Explain how the program is compiled and run.** The code is compiled and run the same way as in the example in the first assignment:

   ```
   nvcc lab_ex1.cu -o ex1 & ./ex1 32
   ```

2. **For a vector length of N:**
   - **How many floating operations are being performed in your vector add kernel?** One FLOP is performed in the kernel since one index of two vectors is added within it. This process is repeated N times.
   - **How many global memory reads are being performed by your kernel?** Each thread reads one element each from $in1$ and $in2$. Having N threads means there will be $2N$ global memory reads.

3. **For a vector length of 1024:**
   - **Explain how many CUDA threads and thread blocks you used.** I have set 32 threads per block (TPB). Additionally, I set the blocks per grid (BPG) in the following way,

     $$BPG = \frac{\text{inputLength} + \text{TPB} - 1}{\text{TPB}}.$$

     This means that there are $(1024 + 32 - 1)/32 = 32$ blocks.
   - **Profile your program with Nvidia Nsight. What Achieved Occupancy did you get?**

     ```
     The input length is 1024
     ==PROF== Connected to process 953 (/content/ex1)
     ==PROF== Profiling "vecAdd" - 0: 0%....50%....100% - 1 pass

     Reference:
     75.556 30.453 132.526 177.833 148.917 113.941 164.400 70.804 48.779 41.797 133.297 63.837 63.337 3.876 109.079 88.879 126.856

     GPU:
     75.556 30.453 132.526 177.833 148.917 113.941 164.400 70.804 48.779 41.797 133.297 63.837 63.337 3.876 109.079 88.879 126.856
     ==PROF== Disconnected from process 953
     ==WARNING== Found outstanding GPU clock reset, trying to revert...Success.
     [953] ex1@127.0.0.1
       vecAdd(double *, double *, double *, int), 2023-Nov-26 18:30:28, Context 1, Stream 7
         Section: Command line profiler metrics
         ---------------------------------------------------------------------- --------------- ------------------------------
         sm__warps_active.avg.pct_of_peak_sustained_active                                    %                           4.97
         ---------------------------------------------------------------------- --------------- ------------------------------
     ```

4. **Now increase the vector length to 131070:**
   - **Did your program still work? If not, what changes did you make?** Yes, it worked.

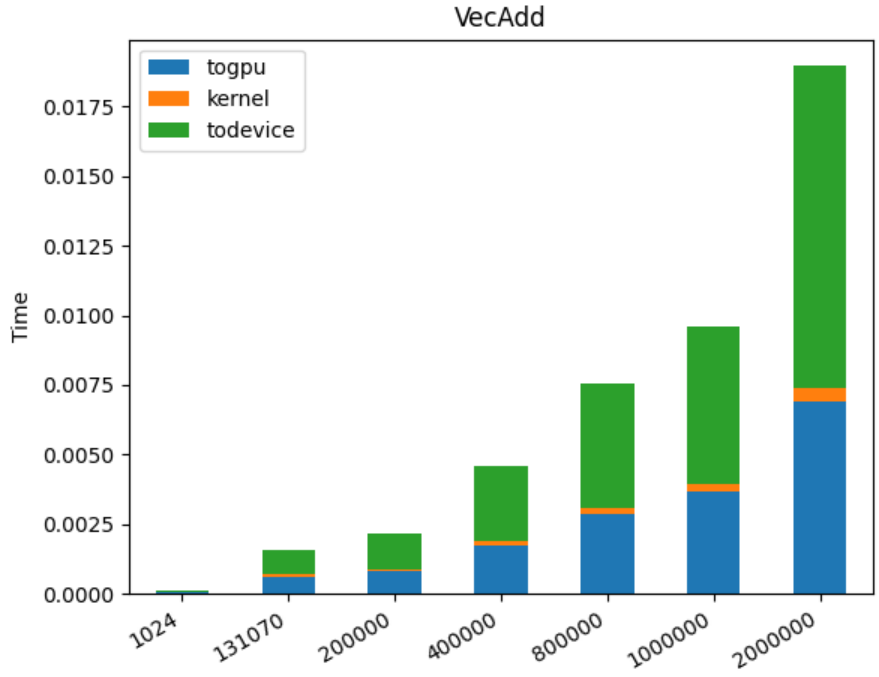
- **Explain how many CUDA threads and thread blocks you used.** TPB is left at 32. With the same equation we can calculate number of blocks to be $(131070 + 32 - 1)/32 = 4096$.
- **Profile your program with Nvidia Nsight. What Achieved Occupancy do you get now?**

```
The input length is 131070
==PROF== Connected to process 2035 (/content/ex1)
==PROF== Profiling "vecAdd" - 0: 0%....50%....100% - 1 pass

Reference:
75.871 87.644 126.087 50.453 157.643 112.186 114.621 176.387 123.089 192.387 62.460 76.776 136.182 137.511 117.633 35.135 148.6

GPU:
75.871 87.644 126.087 50.453 157.643 112.186 114.621 176.387 123.089 192.387 62.460 76.776 136.182 137.511 117.633 35.135 148.6
==PROF== Disconnected from process 2035
==WARNING== Found outstanding GPU clock reset, trying to revert...Success.
[2035] ex1@127.0.0.1
  vecAdd(double *, double *, double *, int), 2023-Nov-26 18:34:33, Context 1, Stream 7
    Section: Command line profiler metrics
    ---------------------------------------------------------------------- --------------- ------------------------------
    sm__warps_active.avg.pct_of_peak_sustained_active                                   %                          33.14
    ---------------------------------------------------------------------- --------------- ------------------------------
```

5. **Further increase the vector length (try 6-10 different vector length), plot a stacked bar chart showing the breakdown of time including (1) data copy from host to device (2) the CUDA kernel (3) data copy from device to host. For this, you will need to add simple CPU timers to your code regions.**



Exercise 2

1. **Name three applications domains of matrix multiplication.**

   (1) Deep learning, (2) Computer graphics, and (3) scientific simulations

2. **How many floating operations are being performed in your matrix multiply kernel?**

   Each iteration of the loop inside the kernel has one multiplication and one addition operation. Additionally, this kernel operation is done for all calculations during matrix multiplication. This corresponds to the number of elements in the output matrix C. Therefore the total number of FLOPs in this matrix multiplication is `numCRows * numCColumns * 2 * numAColumns`.

3. **How many global memory reads are being performed by your kernel?**

   For each thread there are `numAColumns` reads for matrix A and matrix B each. Therefore, in total `numCRows * numCColumns * 2 * numAColumns`.

4. **For a matrix A of (128x128) and B of (128x128):**

   - **Explain how many CUDA threads and thread blocks you used.** Block size is set to 8 by 8 which is 64 threads per block. A similar method to the one in the first exercise is used to calculate one dimmention of the number of blocks (also same as in the lecture). This gives the grid size of 16 by 16 which is 256.
   - **Profile your program with Nvidia Nsight. What Achieved Occupancy did you get?**

```
[13326] matmul@127.0.0.1
  gemm(double *, double *, double *, int, int, int, int), 2023-Nov-26 19:18:50, Context 1, Stream 7
    Section: Command line profiler metrics
    ---------------------------------------------------------------------- --------------- ------------------------------
    sm__warps_active.avg.pct_of_peak_sustained_active                                   %                          39.84
    ---------------------------------------------------------------------- --------------- ------------------------------
```

5. **For a matrix A of (511x1023) and B of (1023x4094):**

   - **Did your program still work? If not, what changes did you make?** Yes, it worked.
   - **Explain how many CUDA threads and thread blocks you used.** Block size again is 64 threads. With the same calculation grid is $64 * 512 = 32768$.
   - **Profile your program with Nvidia Nsight. What Achieved Occupancy do you get now?**

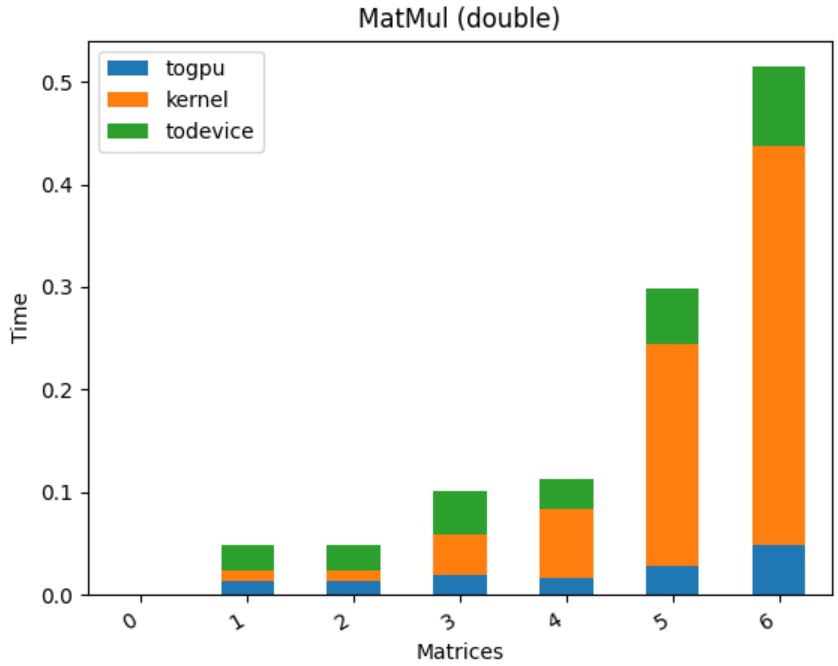
```
==WARNING== Found outstanding GPU clock reset, trying to revert...Success.
[15530] matmul@127.0.0.1
  gemm(double *, double *, double *, int, int, int, int), 2023-Nov-26 19:27:29, Context 1, Stream 7
    Section: Command line profiler metrics
    ---------------------------------------------------------------------- --------------- ------------------------------
    sm__warps_active.avg.pct_of_peak_sustained_active                                   %                          97.01
    ---------------------------------------------------------------------- --------------- ------------------------------
```

6. **Further increase the size of matrix A and B, plot a stacked bar chart showing the breakdown of time including (1) data copy from host to device (2) the CUDA kernel (3) data copy from device to host. For this, you will need to add simple CPU timers to your code regions. Explain what you observe.**

   The difference in matrix multiplication compared to vector addition is that the kernel now takes a bigger portion of the total time. This is expacted since the number of operations has grown much more than the amount of data that is read/written. The matrices correcponding to the indeces provided in the figures are provided in Table 1.

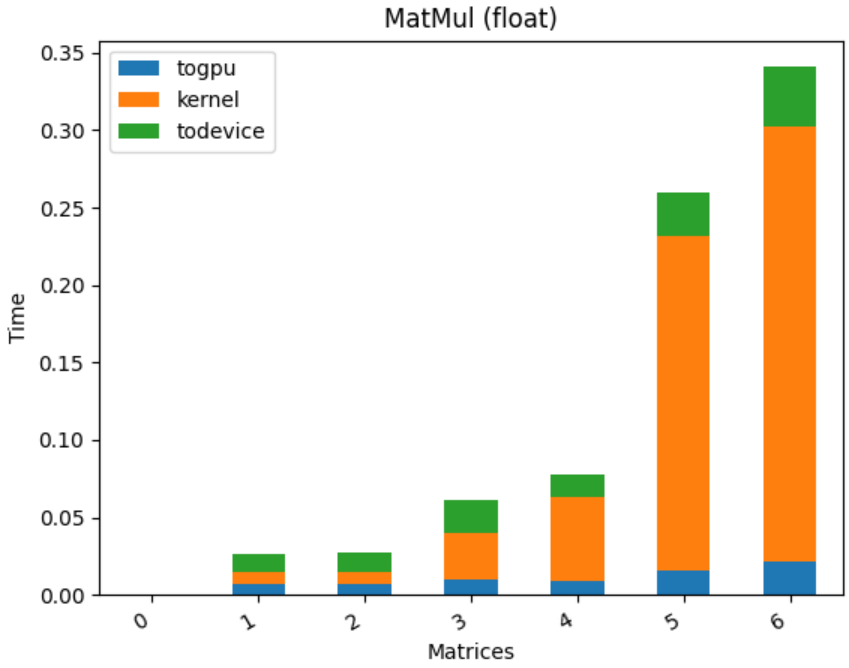
Table 1: Tested matrices

Index	Arow	Acol	Brow	Bcol
0	128	128	128	128
1	511	1023	1023	4094
2	512	1024	1024	4094
3	1024	1024	1024	4094
4	1024	2048	2048	2048
5	2048	2048	2048	2048
6	2048	4094	4094	2048



7. **Now, change DataType from double to float, re-plot the a stacked bar chart showing the time breakdown. Explain what you observe.**

   Total time decreases when the data type is changed to float because float has a lower precision and therefore it is an ”easier” task.


MatMul (float)
togpu
kernel
todevice
Time
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
0
1
2
3
4
5
6
Matrices