ME759
High Performance Computing for Engineering Applications
Assignment 7

Problem 1.

1.1. Run an analysis to gauge how much time it takes you to move data down into the GPU at a location A in device global memory. To this end, transfer from the host to address A in device global memory 2^0 bytes; 2^1 bytes; 2^2 bytes; …; 2^{30} bytes. Generate a **png** plot that shows the amount of time required by each of these transfers. Do not register the amount of time necessary to allocate memory on host and/or device. You might want to allocate memory on the device once, for the most demanding case (2^{30} bytes) and then use the address A for all the other data transfer cases. Make sure you (*i*) run your code several times to get a good idea about the average amount of time you'd encounter in real-life applications; and (*ii*) free the memory you allocate on the device before you finish.

1.2. In a different **png** plot, compare the non-pinned vs. pinned memory transfer speed. Run the analysis for 2^0 bytes, 2^1 bytes,…, all the way to 16 KB. What needs to be plotted is the ratio $\frac{t_N^{NP}}{t_N^P}$, where t_N^P stands for the time it took to transfer N bytes in pinned memory mode; the NP stands for non-pinned memory mode; i.e., like in 1.1. Make sure you free the memory you allocate on the device before you finish.

1.3. Do the same thing as before but now time both a copy to device, then a copy from the device back to host. This would simulate the typical scenario you have in a real-life application: you copy data to the device, do something to it there, copy back a result (here you don't do anything to the data on the device, you only want to get an idea of the absolute time it takes to move data around). To this end, allocate 2^{25} bytes on the device, say at location A. Then allocate the same amount on the device, say at location B. Time the following sequence of two operations: copy N bytes from host to A, then copy back N bytes from B to host. Run this analysis for $N = 2^0, \ldots, 2^{25}$ bytes and generate a **png** plot with timing results for each value of N. Make sure you free the memory you allocate on the device before you finish. How are your results correlating with the results at 1.1 above and why?

1.4. Repeat the same experiment as for 1.2 above but using also the scenario of 1.3; i.e., data transmission both ways, and generate a **png** plot based on your observations. Make sure you free the memory you allocate on the device before you finish. How are your results correlating with the results at 1.1 above and why?

For this problem, upload the four **png** plots to the forum (under topic "Assignment 7: MemCpy overhead plots") and provide answers to the questions above in a separate document. The timing should be based on using CUDA events as described in an early lecture this semester.

Problem 2.
Revisit the array reduction code you wrote a couple of weeks ago. Specifically, write a piece of code that does the following:

a) Takes as command line arguments two positive integers: the first one, call it N, indicates the size of the array to be reduced; the second one, call it M, indicates the max absolute value of any entry in the array to be generated. Example:
```
>> myProgr 100000 5
```
b) Generates on the host an array of N random <u>double precision</u> numbers in the range $-M$ to M (to get a random number in double precision you might first generate an integer then multiply it by 1.0).

c) Uses CUDA code to sum up all the double precision numbers in the array and compares the result against a version of the code that runs on the CPU. A message should be output to report the reduction result on the CPU and on the GPU. Note: there might be small differences in the CPU and GPU results owing to finite precision arithmetic aspects.

d) Presents a timing report to indicate the amount of time spent on the CPU and GPU to reduce the array. The GPU should report inclusive time.

Report on the forum (under topic "Timing Results Assignment 7: Reduction Operation") the CPU and GPU times you get when you run your executable using the following input:
```
>> myProgr 50000000 5
```

For this problem

i) You should have some checks in place to indicate whether the memory allocation on the device was successful (the TA will try to use your code for an array of one trillion entries). In case of error, the code should exit graciously with a message to indicate the cause of failure

ii) You can use any material covered in class, recycle any code discussed in class, use any information you read from any external source (including CUDA SDK examples). However, no copy-and-paste of code from other sources. Again, it's ok to copy and paste code from the lecture notes.

For this problem, the TA will list the winner of the speed competition on the forum. There will be two champions:

1) Winner of the fastest CPU solution competition
2) Winner of the fastest GPU solution competition

The TA will use an array larger than $N = 1,000,000$ entries to evaluate your code, and $M \leq 10$.