TRINITY COLLEGE DUBLIN, SCHOOL OF MATHEMATICS

Assignment #1, Module: MA5615

Gustavo Ramirez

March 2, 2017

1 Part 1: serial

After compilation, and when the program is executed (e.g. ./a.out), the calculation of 'max norm' is very fast, in comparison with the rest of methods; the other methods take, in average, around 7 or 8 times longer than max norm. These enlargements in execution time are due to jumps within the 1D array (form in which the data was stored), to read for example values of columns of the matrix, and also some operations as square root or raising to certain power, imply more execution time.
The full execution of this program is of the form: ./mat_norms_serial [-n N] [-m M] [-s] [-t]

2 Part 2: parallel

In this implementation, a use of the '-t' flag represents a considerable increase in information about timing.
As can be seen when the program is executed, a big decrease in execution time is encountered; e.g. (using 10 000 for n and m):

```
begin!
Frobenius norm: 4096.000000
*** execution time for Frobenius norm: 0.741819
——
cudaMemcpy time: 0.063989
gpu config time: 0.000001
norm processed!
copy-back-to-host time: 0.218801
Frobenius norm with CUDA: 5773.468262
*** execution time for Frobenius norm with CUDA (excluding cudaMalloc and cuda mem copy): 0.000017
*** and including cudaMalloc and mem copy timing: 0.283485
end!
```

As can be seen, most of the time accounts for the copy from device to host, whilst the execution within the device itself is very short.
On the other hand, it's very important to note the discrepancy in the result for the norm in this case; this is due to the precision. For example, if using 100 for both dimensions of the matrix:

begin!

```
Frobenius norm: 57.409531
*** execution time for Frobenius norm: 0.000117
——-
cudaMemcpy time: 0.000033
gpu config time: 0.000000
norm processed!
copy-back-to-host time: 0.000806
Frobenius norm with CUDA: 57.409561
*** execution time for Frobenius norm with CUDA (excluding cudaMalloc and cuda mem copy): 0.000010
*** and including cudaMalloc and mem copy timing: 0.001112
```

end!

The important point about precision is that, due to the 10^{-7} for floats, if $n \cdot m \approx 10^6$, then the accuracies start being unreliable.

3 Part 3: performance improvement

A couple of automated Python scripts were developed/used for testing in this part.
The script perf_tester.py performs all the executions (for all the matrix sizes and all the threads per block specified in the description of this assignment), then storing output results in the out.dat file. Then, the script plotter.py takes data from out_bu.dat (to use this, simply rename out.dat to out_bu.dat), which can be redirected to an output file or just be read from the terminal (in this case, it has been stored in the file results.txt).
As can be seen, precision becomes very important for larger matrices, and the transmission of data between RAM and global memory at the GPU, is an important aspect to be equilibrated in conjunction with how many threads to use in the calculation.
It's important to note that, from the data in part3/results.txt, 'max norm' and 'Frobenius norm' execution times become dangerously large.

4 Part 4: double precision testing

To make easier the transition from float to doable, the code implemented in part 2 contained the re-definition of float as VAR_TYPE, so that the change to doable was practically trivial.

From the file part4/tests/results.txt, it's noticeable how, although the execution times increase, the values of the norms for serial and parallel are much more coincident, as for larger matrices, the error propagates at much smaller decimal positions.