An Introduction to HPC and Scientific Computing

CWM, Department of Engineering Science

University of Oxford

Wes Armour

Assignment: CUDA programming.

This assignment will test the skills that you have developed over the last week. You are to choose one part (part A or Part B) below and complete the tasks listed. Marks will be given for:

Good coding practices	- 2 marks.
Using a build system	- 1 mark.
Correct use of C/CUDA	- 1 mark.
Working code	- 1 mark.

So a possible 5 marks in total. You may help each other or work in groups if you would like to. You have both morning and afternoon practical sessions to complete these tasks. The lab will also be available during lunchtime.

All practicals for this course will be carried out on the Universities ARCUS-B computer. To understand how to use ARCUS-B see the slides from lecture 3. As a reminder log in using ssh as follows:

ssh –CX teachingXY@arcus-b.arc.ox.ac.uk

Where teachingXY is the account that we have issued you with.

When logged into the Arcus-B head node, you can create an interactive session on one of the K80 GPU compute nodes by issuing the following command:

export SALLOC_RESERVATION=cuda-openmp-thu

salloc -pgpu --ntasks-per-node=1 srun --pty --x11 --preserve-env /bin/bash -l

Once your interactive session begins, issue the commands:

module load gpu/cuda

and

export CUDA_VISIBLE_DEVICES=0,1,2,3

If you have not done so clone the github repo for this CWM. To do this, at the command prompt type:

$ git clone https://github.com/wesarmour/CWM-in-HPC-and-Scientific-Computing.git

Or

```
$git pull
```

To update your local repo.

Instructions for this assignment

Part A

1. Study the reduction code that I've supplied. Ensure that you understand how it works. You might find it helpful to draw out a small example for a small number of threads (say eight).
2. Add dynamic memory allocation to the code.
3. Extend the code to work with any power of two (hint, you'll need extra thread blocks, then take care regarding the final reduction across thread blocks).
4. Take input from the user to determine the number of random numbers to work with.
5. Add the cuRAND library to replace the rand() function.
6. Extend the code so it works with any number supplied by the user (a non-power of two).
7. Add OpenMP to the supplied C code.
8. Produce a plot of GPU speed vs CPU speed for a varying reduction length.
9. Use the code to output a mean value for different random number distributions.
10. Provide a make file.

Part B

1. Convert the Monte-Carlo code provided to calculate pi into a CUDA code.
2. Add a function to estimate the accuracy of the output as the number of randoms increases.
3. Produce a plot of the above for increasing number of randoms.
4. Convert the CPU code provided into an OpenMP code.
5. Produce a plot of GPU speed vs CPU speed for an increasing number of randoms used.
6. To increase the speed of your code consider using the reduction code supplied.
7. Compare the speed of the code that uses the reduction algorithm to your original code.

8. Change your code so that it now counts the number of summed uniform randoms that are required to exceed the value 1. e.g random[0]=0.123 random[1]=0.345 random[2]=0.789. So the sum that exceeds 1 has 3 elements in this case.

9. Modify your code so that it calculates the average of all of these sums. What number does it produc*e*

10. Provide a make file.

At 17:00 (28th May 2018) you must email me your code. This can be attached to an email with plots if you've managed to produce them, or as a link to your code and plots in your github repo.

Email: wes.armour@eng.ox.ac.uk