Initial Plan

Author: Matthew Nunes
Supervisor: David Walker
Moderator: Xianfang Sun
Module Code: CM3203
Module Title: One Semester Individual Project
Credits: 40

Project Title: Parallel performance study of scientific codes using CUDA and OpenCL

Project Description:

Moore's law is one of the most well known laws amongst computer scientists. It asserts that the number of transistors on a chip will double every two years. This has been true in the past, however, chips are now beginning to reach their physical limits and it is predicted that Moore's law will stop showing this linear increase by 2020. Despite the inadequacies of Moore's law, computer speeds have continued to increase at an alarming rate. This has been achieved both by adding more processors per chip and employing co-processors in the form of Graphics Processing Units to assist in processing. The new hardware paradigms of parallel computers has led to a range of new software models to allow programmers to exploit the additional hardware. Two examples of these are Compute Unified Device Architecture (CUDA) and Open Computing Language (OpenCL).[1][2][5]

CUDA was developed by NVIDIA and is therefore aimed at NVIDIA's line of GPUs. Effictively, CUDA is an extension of C/C++. CUDA can only be run on NVIDIA GPUs and is optimised and written specifically for them. [3][4]

On the other hand, OpenCL is a cross-platform, open standard extension of C for writing portable parallel programs that can make use of CPUs, GPUs, FPGAs (Field Programmable Gate Arrays) and more. OpenCL is maintained by the Khronos Group and is supported by a large number of companies, a few which include Apple, Intel, Advanced Micro Devices (AMD) and NVIDIA. SDKs (Software Development Kit) have been developed for OpenCL by Intel, AMD and NVIDIA. [6][7][8][9]

The project I will be attempting is aimed at comparing the performance of OpenCL and CUDA on a variety of algorithms. The algorithms that will be implemented are:

1) The Matrix Multiplication algorithm
2) The Laplace equation
3) An image blurring algorithm
4) A molecular dynamics algorithm
5) A cellular automata algorithm

The Matrix multiplication algorithm is as its name suggests an algorithm that takes two matrices and produces the result of their product. The behaviour of this algorithm can be studied as the size of the matrix is increased.

The Laplace equation is used to determine the electric field surrounding a conducting object held at a fixed electrical potential inside a box which is also at a fixed electrical potential. This is quite a complex problem to parallelise due to the care that must be taken when choosing which values to modify and which not to touch. Therefore it will be interesting to compare how well it suits

OpenCL and CUDA.

The image blurring algorithm replaces each pixel value at a point with the average of its neighbour's values. Doing this blurs the image. This can be done more than once to further blur the image.

The molecular dynamics algorithm simulates the movement of molecules in a closed system where the molecules obey the following rules: 1) If the molecules are close to each other, they will repel one another 2) If the molecules are far apart, they will attract each other 3) If the molecules are extremely far apart, they will have no effect on one another. By following these rules, it is possible to simulate particle dynamics in a system.

Finally, the last algorithm that will also be implemented if time allows it is a cellular automata algorithm which simulates the reaction that occurs on the catalytic converter of a car. To put it simply, Carbon Monoxide reacts with Oxygen to give off the less harmful Carbon Dioxide. The end result of this algorithm can take many different forms depending on values of the various variables.

The main variable being measured and compared is the runtime. Data will be gathered by varying the problem sizes and language specific variables (such as chunk size, number of threads per block etc.) to see how changing the variables affects the runtime. All the experiments will be carried out on the same machine to reduce the chances of the results being skewed unintentionally. The data will be stored in a spreadsheet (such as Excel or OpenOffice Calc) and graphs will be generated in order to easily assess the trend in behaviour. Doing this should clearly show whether the OpenCL or CUDA implementation of an algorithm is more efficient

Time permitting, the algorithms will then be further optimised using a variety of techniques which include making use of shared memory and re-writing the algorithms to reduce the conditional statements within them. Furthermore, given that OpenCL should run on any AMD device, its performance for one of the algorithms could be analysed when running on a phone. Graphs will be plotted in order to fully comprehend the effect of these changes on the runtime.

Final Report Details:

The final report that I will work on during the Easter break will contain the following headings in its main body:

1. Introduction
2. Motivation and Background
3. Problem Description
4. Approach to problem
5. Code Description
6. Timing Experiments Performed
7. Results from Experiments
8. Analysis of Results
9. Future Considerations
10. Conclusion
11. Reflection

Essentially the final report will contain a description of how much I was able to achieve, an explanation of my implementation, the results from running tests (with graphs), additional areas in which the experiments could be further explored (future considerations) and a reflection on my experience with this project.

Aims and Objectives:

- Implement the Matrix Multiplication algorithm, Laplace equation algorithm, image blurring algorithm and the molecular dynamics simulation using both CUDA and OpenCL
- Compare the performance of the different implementations of the algorithms when problem sizes and language specific variables (such as chunk size, threads per block etc.) are modified
- Visualise the performance of the algorithms using graphs
- (Optional) Implement the cellular automata algorithm using CUDA and OpenCL.
  - Compare the performance of the different implementations for this algorithm as before
- (Optional) Run one of the algorithms implemented using OpenCL on a phone capable of running OpenCL.
  - Measure its performance on a phone.
- (Optional) Optimise the algorithms by exploiting elements of both the hardware and software such as using shared memory.
  - Plot graphs to assess to benefits from the optimisations

Work Plan:

Time Period	Tasks
Week 1 (27th January - 2nd February)	**1.** Write Initial Report **2.** Research OpenCL and CUDA compilers **3.** Learn OpenCL and CUDA **4.** Meet Supervisor
Week 2 (3rd February - 9th February)	**1.** Implement the matrix multiplication algorithm using OpenCL and CUDA **2.** Compare the performance of the algorithm written in OpenCL and CUDA **3.** Implement the Laplace equation using CUDA and OpenCL **4.** Compare the performance of the algorithm written in OpenCL and CUDA **5.** Meet Supervisor
Week 3 (10th February - 16th February)	**1.** Implement the image blurring algorithm using OpenCL and CUDA **2.** Compare the performance of the algorithm written in OpenCL and CUDA **3.** Meet Supervisor
Week 4 (17th February - 23st February)	**1.** Implement the molecular dynamics algorithm using CUDA and OpenCL **2.** Compare the performance of the CUDA and OpenCL implementations of the molecular dynamics algorithm **3.** First review meeting with Supervisor
Week 5 (24th February - 2nd March)	**1.** Implement the cellular automata algorithm using CUDA and OpenCL **2.** Meet Supervisor

Week 6 (3rd March - 9th March)	**1.** Compare the performance of the CUDA and OpenCL implementations of the cellular automata algorithm **2.** Meet Supervisor
Week 7 (10th March - 16th March)	**1.** Run and measure the performance of one of the OpenCL algorithms on a phone **2.** Optimise the OpenCL and CUDA algorithms for matrix multiplication **3.** Measure and compare the performance gain **4.** Meet Supervisor
Week 8 (17th March - 23st March)	**1.** Optimise the OpenCL and CUDA algorithms for the Laplace equation **2.** Measure and compare the performance gain **3.** Meet Supervisor
Week 9 (24th March - 30th March)	**1.** Optimise the OpenCL and CUDA algorithms for the image blurring algorithm **2.** Measure and compare the performance gain **3.** Meet Supervisor
Week 10 (31st March - 6th April)	**1.** Optimise the OpenCL and CUDA code for the molecular dynamics algorithm **2.** Measure and compare the performance gains **3.** Second review meeting with Supervisor
Week 11 (7th April - 13th April)	**1.** Optimise the OpenCL and CUDA code for the cellular automata algorithm **2.** Measure and compare the performance gains **3.** Second review meeting with Supervisor
Easter recess (14th April - 4th May)	**1.** Write report **2.** Meet Supervisor
Week 12 (5th May - 9th May)	**1.** Submit report **2.** Meet Supervisor

Work Plan Explained:

The table above shows an optimistic work plan in which all the optional objectives will be achieved. Since it is ambitious, it is also not set in stone. The work plan is subject to change as I will be following an Agile software development methodology (to the extent that is possible without a team). I chose this since it is flexible in that it allows requirements to change and it is suitable for small teams. While it is true that the Waterfall method is more popular and practiced, it is aimed more at large teams. In addition, it seems counter-productive to be chained to a methodology that doesn't encourage change despite the relatively low cost it would involve (in my case).

References

1. Blaise Barney, Lawrence Livermore National Laboratory. 2013. Introduction to Parallel Computing [Online]. Available at: https://computing.llnl.gov/tutorials/parallel_comp/ [Accessed: 29-01-2014]

2. Moore's Law [Online]. Available at: http://www.mooreslaw.org/ [Accessed: 29-01-2014]

3. NVIDIA. 2014. What is CUDA [Online]. Available at: https://developer.nvidia.com/what-cuda [Accessed: 29-01-2014]

4. Sarah Tariq, NVIDIA Corporation. 2011. An Introduction to GPU Computing and CUDA Architecture [Online]. Available at: http://on-demand.gputechconf.com/gtc-express/2011/presentations/GTC_Express_Sarah_Tariq_June2011.pdf [Accessed: 29-01-2014]

5. Michigan Technological University. 2008. Scientist models molecular switch [Online]. Nanowerk. Available at: http://www.nanowerk.com/news/newsid=6076.php [Accessed: 29-01-2014]

6. Khronos Group. 2014. The open standard for parallel programming of heterogeneous systems [Online]. Available at: http://www.khronos.org/opencl/ [Accessed: 29-01-2014]

7. NVIDIA. 2014. OpenCL [Online]. Available at: https://developer.nvidia.com/opencl [Accessed: 29-01-2014]

8. Intel. 2014. Getting Started With OpenCL* Applications [Online]. Available at: http://software.intel.com/en-us/vcsource/tools/opencl [Accessed: 29-01-2014]

9. Mahesh Doijade. 2013. What is OpenCL [Online]. TechDarting. Available at: http://www.techdarting.com/2013/06/what-is-opencl.html [Accessed: 29-01-2014]