Com4521/Com6521: Parallel Computing with GPUs

Assignment: Part 2

Deadline: Monday 13th May 2019 17:00 (week 12)

Last Edited: 09/03/2018

Marking

Assignment 2 (of 2) is worth 70% of the total assignment mark. The total assignment mark (both parts 1 and 2) is worth 80% of the total module mark.

Assignment 2 marks will be weighted as 50% for code functionality and performance and 50% for demonstrating understanding via a written report.

Document Changes

This is Version 1 of the assignment document. Any corrections or changes to this document will be noted here and an update will be sent out to the course google group mailing list.

Introduction

The aim of the assignment is to test your understanding and technical ability to implement efficient code on the GPU. You will be expected to benchmark and optimise the implementation of a simple image processing problem. You have already implemented a serial and multi-core version in Assignment 1, you are expected to implement a GPU version of the same task. The emphasis of this assignment is on your ability to progressively improve your code to converge on an efficient implementation. In order to demonstrate this, you are expected to document (in a written report) any design consideration you have made in order to improve the performance (and ensure correctness). For example, you should use benchmarking to demonstrate how changes to your implementation have resulted in performance improvements, or explain through theory why a particular method was chosen. Handing in just a piece of code with excellent performance will not score highly in the assessment, unless you have also demonstrated an understanding in the written report, of how you have progressively refined your implementation to reach the final solution.

The Assignment Requirements (Code)

You are expected to implement the pixelate filter on the GPU (using the same problem definition from the Assignment 1 document). You should use your hand-in for assignment 1 as your starting code which should already perform IO and the CPU/OpenMP modes. Your assignment 1 code will already have a placeholder for when the program `mode` is `CUDA`. In order to add GPU code to your assignment 1 hand-in you will need to rename your existing C file to a CUDA extension (`*.cu`) before importing it to a new Visual Studio, NVIDIA CUDA project. Your program should accept the additional `CUDA` mode argument and you should update the `print_help()` function to reflect the new CUDA option. You should use the feedback from your assignment 1 hand-in to ensure your GPU code is efficient and works correctly. You may need to update your file IO code if you have previously implemented this incorrectly (otherwise your GPU code may produce incorrect results). Your CPU and OpenMP code should still be able to be run using the `mode` program argument but this code will not be re-assessed

as part of assignment 2.

Timing:

The program arguments for the GPU version of your code are the same as the previous assignment. You should take care to ensure that you accurately time the GPU code using an appropriate technique.

Parallel Implementation:

There are two obvious methods to implement the pixelate technique on the GPU. You can either parallelise each mosaic cell or you can parallelise each input pixel. Each approach may be more suited to certain sizes of the mosaic cell size C. You may implement either method (or a hybrid of both), however, you must justify your decision within the document. You may consider implementing both techniques to provide evidence as to which approach is favourable when considering different C values. You should consider a range of input image sizes ensuring that for either version you have enough threads to fully utilise the device. Larger input images may be slower but may give better device utilisation.

Documentation Requirements

You are expected to document the implementation of your code. More specifically, you are expected to compare and contrast various implementation techniques to show how you have converged on a particular implementation. In particular, you should benchmark your code in Release mode to compare alternative techniques (such as the use of various GPU memory caches where appropriate) and give an explanation as to why one implementation technique (or optimisation you have made) is better than the other. Some examples of interesting benchmarks or discussions include;

- Parallelisation approach?
- Different methods to layout or represent your data in GPU memory to ensure coalesced access patterns. E.g. Arrays of Structures vs Structures of Arrays.
- The use of various GPU memory caches (texture/read-only, constant, shared memory) to reduce the number of global memory reads? *Note: Not all will be suitable for your problem but should discuss why not.*
- Any GPU optimisations you have made to improve the performance? A description of any investigations into performance through benchmarking or profiling.
- Any other interesting aspects of the implementation or optimisation techniques you have applied to the GPU version of your code.

Benchmarking should always be done in Release mode within Visual Studio with timing results for a single run of your program averaged over a number of independent program runs. Benchmarks should consider various values of Image size and the mosaic filter size C (as defined in part 1 document) to demonstrate performance scaling. For each significant improvement to your code try to show the performance of your code before and after changes. You should highlight (with short code samples) any novel aspects or optimisations you have made.

Project Hand In

You should hand in your program code via MOLE with the documentation as a single pdf within a single zip file. You should also include the Visual Studio solution and any project files. Your code should build

in the Release mode configuration without errors or warnings (other than those caused by IntelliSense) on Diamond computer room 4 (lab) machines. You should submit whatever you have done if you have not completed the entire assignment. Your code should not rely on any third party libraries or tools (other than those included with CUDA or OpenMP).

Marking

The marks for part 2 of assignment will be distributed as follows:

- *50%* of the assignment is for the coding aspect. Half of this percentage is for the quality of the programming and optimisation (including the performance of your code) and the other half is for satisfying the requirements.
- *50%* of the assignment is for the production of a document describing the processes you have undertaken to implement and optimise your code. This should include benchmarking and iterative refinement of approaches as described in the documentation requirements.

In assessing your work, the following requirements will be considered for the code aspect.

1. Is the GPU code functionally correct? I.e. Has the technique been implemented correctly and does it produce the correct result? A number of test cases will be used to evaluate this against the reference implementation.
2. Have you managed to use GPUs appropriately ensuring that the device has sufficient levels of parallelism for all mosaic cell sizes?
3. Has iterative improvement of the code yielded a sufficiently optimised final GPU program?
4. Does the code make good use of memory bandwidth?
5. Does the GPU code avoid race conditions when reducing value?
6. Does the GPU code use effective caching to reduce the number of global memory loads?
7. Are there any compiler warnings or dangerous memory accesses (beyond the bounds of the memory allocated)? Does your program free any memory which is allocated?
8. Is your code structured clearly and well commented?

In assessing your documentation, the following will be considered and should act as a guideline for discussing incremental improvements to your code.

1. Description of the technique and how it is implemented. Is a good justification given for the choice of parallelisation method?
2. Have appropriate investigations been made into using a good memory access pattern and suitable caching technique? Are good explanations given for the benchmarking results?
3. Does your document describe optimisations to your code and show the impact of these?
4. Is there benchmarking and discussion about the performance difference between all three version of the code?

Tips for Developing Your Code and Documentation

If you are unable to implement all aspects of the technique on the GPU, then you should default back to using the CPU or OpenMP versions for that part so that your code builds and executes producing the correct result. Similarly, if you apply a technique that does not improve the performance, you should include this in your documentation and explain your belief/understanding as to why it did not work as expected. You can use `#define` to allow your code to be built in different versions to make

a comparison of techniques more straight forwards.

You should comment your code to make it clear what you have done. You should test your code to make sure that it works for **all** images sizes, values of c and image input files. For values and input files which are incorrect (for example $c = -10$ or an input file with the incorrect number of pixel values) your code should exit elegantly with a helpful error message. Your code should never read or write beyond allocated memory.

Assignment Help

Help for your assignment will be available in general lab classes. For specific questions outside of the labs you should use the course google discussion group.