Com4521/Com6521: Parallel Computing with GPUs

Assignment: Part 2

Deadline: 18th May 2020 15:00 (week 11)

Last Edited: 08/05/2020

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.

- 23/03/2020: Clarification of hand in requirements using Com4521 Windows GPU Accelerated Virtual Machine.
- 08/05/2020: Clarified that the deadline is in week 11.

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 NBody simulation. You have already implemented a serial and multi-core version in Assignment 1, you are expected to implement a GPU version of the same simulation. 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. 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 an N-Body simulation on the GPU (using the same problem definition from the Assignment 1 document). You should use the assignment 2 starting code which provides a CUDA project with the correct setting configured and an updated version of the following files.

- `NBody.h` - which adds a GPU mode and `nbody_soa` type to the MODE enumerator
- `NBodyVisualiser.h` – which has some clarifications and additional comments
- `NBodyVisualiser.cu` - which add fast GPU instanced rendering when in GPU mode.

The starting code is available from GitHub (https://github.com/mondus/com4521_assignment/tree/AssignmentPart2). You can either check it out using git or download the starting code as a zip file.

You should import your assignment 1 NBody.c file which should already perform IO and the CPU/OpenMP modes. To add the GPU mode you will need to rename this file to a CUDA extension (`*.cu`) before importing it to the new project. Your program should accept the additional GPU mode argument (see the definition of the `MODE` in `NBody.h`). The input file format for the N-Body initial states is the same CSV format. If your part 1 assignment did not handle file input correctly, you should use the feedback you have been given to update/improve this aspect to ensure that your code passes any tests used as part of the assessment for part 2.

Visualisation and Timing:

Your N-Body simulation should be able to run in either console mode (for a fixed number of iterations, `I`) or in visualisation mode (if the value of `I` is omitted as a program argument) for each of the three (CPU, OPENMP and GPU) modes. When running in console mode, you should use an appropriate timing method to measure the runtime of your simulate function's execution for the `I` number of iterations. This should be output in the form;

```
    "Execution time 123 seconds 456 milliseconds"
```

Writing the visualisation code is not required. An updated source module (`*.cu`) and header file have been provided which implements visualisation (which can be very helpful in checking that your simulation behaves as expected). You will need to use the functions defined in the `NBodyVisualiser.h` file to set pointers to your GPU data. Note: When executing your code in GPU mode, the pointers which you pass to the visualiser must be device pointers rather than host pointers (allocated using `malloc`).

Activity Heat-map:

As before, you are required to calculate an activity map. You should aim to do this on the GPU to avoid reading the bodies back to the CPU system memory each iteration. You will need to ensure that you avoid race conditions.

Parallel Implementation:

There are two possible ways to implement the NBody simulation on the GPU. You can either parallelise each interaction (a total of N*N threads) or you can parallelise each body (N threads). You may implement either method, however you must justify your decision within the document. You may consider implementing both to provide evidence as to which approach is favourable. You should consider a range of problem sizes ensuring that for either version you have enough threads to fully utilise the device. Larger values of N may be slower but will 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 console mode to compare alternative techniques (such as the use of various GPU memory caches where suitable) 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 of each body interaction vs parallelisation of each body?
- 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. It is not required to consider the use of opt-in L1 caching, instead use the read-only/texture cache.*
- Any GPU optimisations you have made to improve the main N-Body simulation loop. How has this affected performance?
- Any other interesting aspects of the implementation or optimisation techniques you have applied to the GPU version of your code.

Good benchmarking should always be done in console mode (to avoid visualisation overhead) with timing results for a single step averaged over a number of iterations (or separate runs of the program). Benchmarks should consider various values of `N` or `D` (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) on the instancehub.com Com4521 Windows GPU Accelerated Virtual Machine. 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 libraries provided in the labs (i.e. OpenGL).

Marking

The marks for part 1 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 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 simulation been implemented correctly and does it produce the correct behaviour?
2. Have you managed to use GPUs for all aspects of the simulation without having to read back

data to and from the device each iteration?

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 in calculating the Histogram/Activity Map?
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 simulation on the GPU (e.g. the activity map), 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 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** values of `N` and `D`. For values and input files which are incorrect (for example a grid width of negative 10 or an input file with the incorrect number of initial values) your code should exit elegantly with an 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.