Com4521/Com6521: Parallel Computing with GPUs

Assignment: Part 1

Deadline: Tuesday 20th March 2018 17:00 (week 7)

Last Edited: 15/02/2018

Marks Allocated

Assignment Part 1 (of 2) is worth 30% of the total assignment mark. The total assignment mark (parts 1 and 2) is worth 80% of the total module mark.

Assignment 1 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 CPU. You will be expected to benchmark and optimise the implementation of a simple image processing problem. You will start by implementing a serial CPU version, you will then parallelise this version for multi-core processors using OpenMP. 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 Task (Overview)

The task is to implement a simple pixelate filter on a photographic image input to produce a photo mosaic. The mosaic should average the pixels within a square area defined by the input cell size (c) which should be any positive power of 2 number (i.e. $c = 2^n$ where n is any positive integer value). The value of c but must not exceed either the image width or height. Figure 1 shows the mosaic effect with varying cell sizes.

The implementation details of the mosaic filter are that a mosaic cell should consist of the average colour value of all pixels within the bounds of the mosaic cell. The number of pixels contributing to each cell (assuming that the image width and height are factors of c) is therefore c^2. There are no boundary conditions for the mosaic generation and a single pixel should contribute only to a single mosaic cell. The output image size should exactly match the input image size. Any output pixels within the same mosaic cell should have the same RGB colour values.

In addition to generating a mosaic image it is also required to generate an average colour RGB value for the entire image. For the example image this is the rgb value (156, 157, 164).

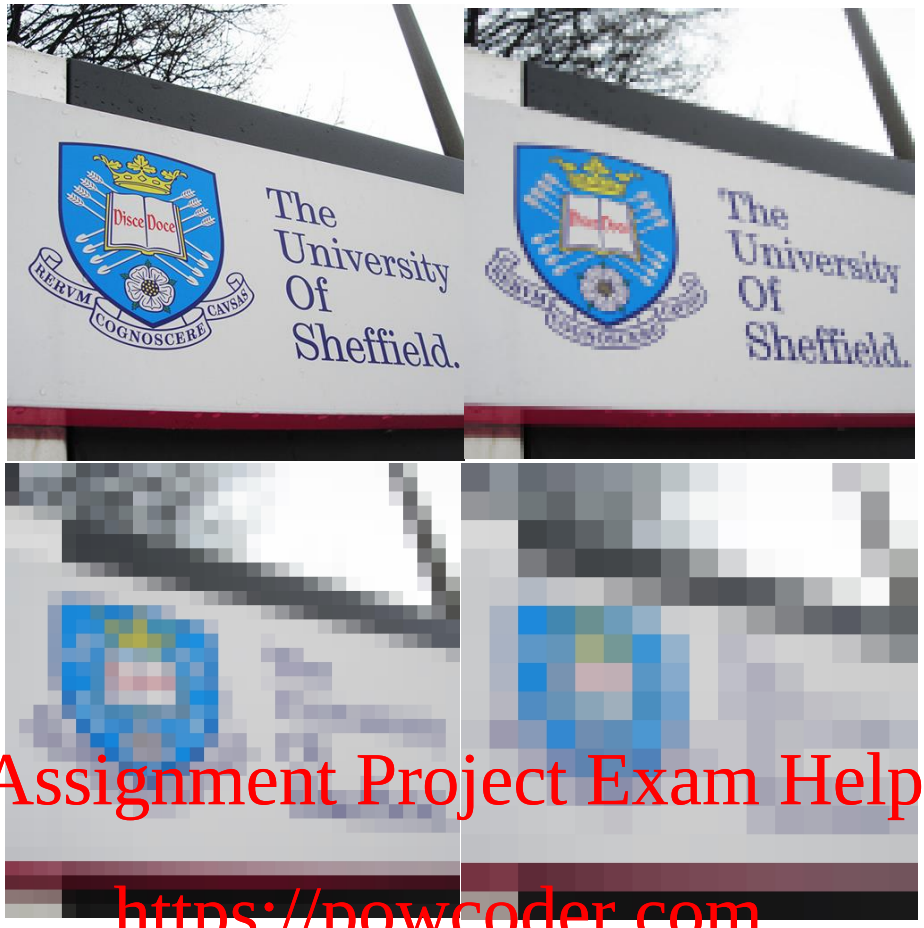

Figure 1 – Varying mosaic cell size. Top Left shows the original image, Top Right shows c = 4, Bottom Left shows c = 16, Bottom Right shows c =32.

Assignment Requirements (Code)

You are expected to implement the Mosaic task. You should start from the provided source code which is available from the following link.

Link to starting code, reference binary and image examples

You should complete the `TODO` sections. Your program should accept the arguments described by the existing `print_help()` function. The assignment has the following additional requirements;

Program Inputs and Output Format:

The required file format for the input and output images in your program is both binary and plain text PPM. A description of the formats is available online (http://netpbm.sourceforge.net/doc/ppm.html). For simplicity it can be assumed that the format will always use a line break to separate the magic number, width, height and maximum colour value. E.g. your code should read and write PPM headers in the following format

```
P3
# comments anywhere in the header
800
600
255
```

Where P3 is the magic number, in the above case indicating plain text format (P6 indicates binary data), 800 is the image width, 600 is the image height and 255 is the maximum colour value. Your code only needs to support maximum colour values of 255 (i.e. 8bits per colour channel). Comments can appear anywhere in the header but not after the maximum colour value.

Plain text PPMs should be in the format of "`r g b\t`" for each pixel value. I.e. each of the red, green and blue values should be separated by a space with a tab after each pixel value. A newline should be placed after each row of pixel values (you can ignore the 7- character line limit suggested by the early format guides). A number of sample plain text and binary PPMs is provided with the test program. You can use Photoshop or Gimp to generate your own binary PPM files.

You are expected to write an input and output function to handle this file format. You code should handle any size of image.

Mosaic Calculation:

You should implement the mosaic filter and image average value calculation; for a single CPU core in C and for multi core CPUs with OpenMP and C (be careful to ensure that OpenMP compiler flags are enabled when building your code). The pixel value for a mosaic cell requires averaging a number of cells. You should be careful to ensure that this is handled both efficiently and correctly.

When calculating the entire image average RGB value, this should represent an average of each pixel value, not of each mosaic value. This will ensure that for image dimensions not divisible by *c* the correct average value will always be obtained.

Program Timing and Command Line Outputs:

The command line output of your code should be the execution time of applying the mosaic filter and the average colour RGB value of the mosaic cells. Your code should execute either serially or with OpenMP based on the programs `mode` argument (see `print_help()` function). If the execution mode is *ALL,* timings and average values for each mode should be displayed to the console. The timing value of reading and writing to disk should be ignored in all cases.

Additional Comments:

You should be careful to ensure that all values of *c* work (excluding *c* values which are greater than either the width or height of the image). Your code should also handle images which are not multiples of *c*. For example, an image of size *500* should be handled when $c = 32$ by having a partial mosaic cell. Within the partial mosaic cell only valid pixels should contribute to the mosaic colour. You must be careful not to read beyond the bounds of the image (or valid memory). For example, if the original image from Figure 1 is cropped to *500x500* the result should be that shown in Figure 2. *Note: You must be careful when averaging values in partial mosaic cells.*

Figure 2: Correct handling of an image not divisible exactly by the mosaic cell size. Note: The Right and bottom edges which contain particle mosaic values.

Testing:

It is important that you ensure your code produces the correct result for each mode by validating the output. A compiled executable is provided as part of the assignment hand-out which is able to provide the expected outputs given an input image. It is expected that your program will pass a number of blind tests during marking and assessment. These tests are blind in that you do not know what they will be other than they will consist of a number images of varying size and for different mosaic cell sizes. For each blind test there is a correct expected output image which has been produced by the reference program. Due to possible rounding errors a margin of 1 will be acceptable when assessing the result of your code.

Documentation Requirements

You are expected to document the implementation of your code. More specifically you are expected to compare and contrast various techniques to show how you have converged on a particular implementation. You should benchmark your code and/or give explanation as to why one implementation technique (or optimisation you have made) is better than another. The following should be considered;

- The choice of data structure used to store pixel and mosaic values.
- A comparison of implementation approaches. Is it best to loop over pixels or mosaic cells in calculating the mosaic cell values?
- A comparison of OpenMP parallelisation strategies for the mosaic calculation. Is parallelising over inner or outer loops (where they exist within your code) more effective?
- A comparison of OpenMP techniques which could be used for calculating average values. How have you avoided race conditions?

- Any optimisations you have made to improve the mosaic calculation loop. How has this effected performance?
- Any other interesting aspects of the implementation or optimisation techniques you have applied to either the serial or OpenMP version of your code.

Good benchmarking should be done over various large (at least *2048 × 2048*) images sizes and values of *c*. 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 optimisation 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 (or makefile) and any project files. Your code should build in Visual Studio release mode configuration without errors or warnings (or for Linux with an appropriate Makefile). You should submit whatever you have completed even if you have not completed the entire assignment. Your code should not rely on any third party libraries or tools.

If you use un-modified code from any of the lab solutions you should make it clear that the code is not your own with comments.

Marking

The marks for part 1 of assignment will be distributed as follows:

- *50%* of the assignment is for the coding aspect.
  - *50% of the coding marks* are for the quality of the programming and performance of your code.
  - *50% of the coding marks* are 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 code functionally correct? I.e. Has the simulation been implemented correctly and does it produce the correct behaviour?
2. Has iterative improvement of the code yielded a sufficiently optimised final program for each of the CPU and OpenMP implementations?
3. Does the code make good use of memory bandwidth?
4. Does the OpenMP implementation avoid race conditions?
5. Are the OpenMP variables correctly scoped? Is their excessive/unnecessary use of shared variables?
6. Are there any compiler warnings or dangerous memory accesses (beyond the bounds of the memory allocated)? Does your program free all memory which is allocated?
7. Is your handling of input files correct and robust? Does the program deal with incorrect input file formatting elegantly (e.g. raising an error and exiting rather than crashing)?
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.

2. Does your document describe optimisations to your code and show the impact of these?

3. Have a range of OpenMP approaches been considered for parallelising the problem? Have appropriate investigations been made into scheduling? Are good explanations given to benchmarking results?
4. Have a range of OpenMP approaches been explored to implement the calculation of mosaic and average image RGB value? Are good explanations given to why a particular approach has been chosen?

Tips for Developing Your Code and Documentation

You should start with a simple serial implementation and describe how you iteratively improve the performance in your documentation. If you are unable to complete some specific part (e.g. the activity map with OpenMP) then you should default back to using the simple CPU version for that part so that your code correctly 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 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.

Feedback

You will receive feedback on your assignment after the Easter holiday which you should consider when handing in Assignment Part 2.