CS3211 project 1 - Image processing: implementation and analysis

February 1, 2017

Due 5pm on Friday March 3rd, and worth 20% of your course mark.

Part 1: Write up your laboratories in a short-essay style. You will explain the results you got with respect to the hardware, the speedup, and accuracy results you achieved. We expect this write-up to take no more than (say) 5-10 pages.

Part 2: Develop a set of original image processing functions, which run both on a GPU and CPU platform, in a browser. You will be doing this using a subset of Javascript, and using GPU.js (GPU accelerated Javascript `http://gpu.rocks/`). In the assessment of your program, we will give marks for range (i.e. how much you completed), and for your use of any specific techniques that you use to improve the behaviour of your program (beyond the obvious one-pixel-per-thread in the demo program given to you). We expect your code to be no more than 150-1500 commented lines.

1 Part 1

In part 1, we want you to show the (tabulated) results you got from your timing experiments in laboratory 1 and laboratory 2. You should then explain why the results are the way they are. In your explanation, we expect to see the following items:

- Hardware: Discussions on the hardware you used (the lab machine and Tembusu), and how the hardware contributes to the results obtained.
- Speedup: The tabulated results from (Lab01) Tasks 3 and 4, along with a discussion on interpreting these tables with respect to the two aspects of speedup from the lectures. For example: if you hold the problem constant, how does the time taken scale with cores? If you hold the time constant, how does the problem scale with cores?
- Memory effects: The tabulated results from (Lab01) Task 6, along with a discussion on the results. What is causing the behaviour you see?
- Accuracy: Tabulate your results from the (Lab02) Tasks 1-4, along with a discussion on the results. What is causing the behaviour you see?
- Communication, speedup: Tabulate your results from the (Lab02) openMPI mm-mpi.c program, Task 7, along with a discussion on the results. What is causing the behaviour you see? How does the openMPI implementation scale when you run on multiple processors?

2 Part 2

We want you to develop GPU.js based implementations of image processing functions, with well commented source code. Your program/web page should demonstrate a range of image processing functions, perhaps edge detections of various kinds, low pass and high pass filtering and so on. It is up to you how much you do. Your main program should allow the user to try out your kernels on either the supplied animation, or perhaps a movie file, displaying the result in a web browser, and allowing easy comparison with the CPU version, as in this sample demo:

`http://www.comp.nus.edu.sg/~hugh/cs3211/project/demo.html`

This sample program has buttons to switch between CPU and GPU, display the before-filtered image, and turn on and off a simple animation. It also displays an approximate number of frames per second. For some reason, it does not seem to work properly under Firefox, but works fine in Chrome, on my Mac, Windows or Android. So perhaps best to use Chrome. Your program may look nicer, and have extra options, but this task is not about the look of your web page, it is about the parallelized components of your program.

You can grab the sources of `demo1.html`. The core part is that GPU.js supplies a restricted mechanism to define functions which compute a single float (32 bit) value. This mechanism is supplied options:

1. If you call the mechanism with the option "`mode: cpu`", you get a function which will run on a CPU. With "`mode: gpu`", you get a function which runs multiple threads on the GPU (a *kernel*).
2. Another option defines a target matrix into which results are to be placed. If you call the mechanism with the option "`dimensions: [800,600]`", you get a function which will iterate over all elements of an 800x600 matrix if run on the CPU. If run on the GPU, each thread operates on a single element (of the whole matrix), with co-ordinates (`this.thread.x`,`this.thread.y`).
3. A third option specifies if you are going to be treating the output matrix as an image, in which case you should set "`graphical: true`".

In the example given, the nameless function inside `gpu.createKernel(functionDef,options)` provides this mechanism. The function definition `functionDef` looks like Javascript, but instead is a restricted subset of Javascript, limited by the WebGL mechanism used.

2.1 Properties of GPU.js

Documentation for GPU.js may be found at

`http://gpu.rocks/getting-started/`

GPU.js was developed at NUS during Hack&Roll last year, and so there is a fair amount of local knowledge about it. To be more precise... all knowledge about it is local! The source is freely available from the website.

The subset of Javascript used by GPU.js has some restrictions that are a little confusing, beginning with the restriction that the data types in the kernel functions can only be `float32` (a restriction imposed by the OpenGL shader architecture). In the example, three `float32` arrays are fed to the `functionDef`, and the output is a 2D `float32` matrix, which we treat as an rgba image. Loop bounds must be fixed at compile time, so the `constants` option helps you import constant values into your GPU function. There is no recursion. (!) ... GPUs generally do not do recursion.

The functions you can call within a kernel, can only be ones written in the restricted subset allowed by GPU.js, and a mechanism is provided to attach these functions to the kernel, to help you structure your program. For example, the example program calls a function `sqr(x)`, implemented in this way:

```
function sqr(x) {
   return x*x;
}
gpu.addFunction(sqr);
```

2.2 Part 2 completion?

There precise definition of which functions you implement has been left undefined. In general, the more different kinds of image manipulation functions you try, the better. In your project paper presentation, you should clearly state at the beginning which functions you implemented. There is an almost endless list of possible things to do, or not do:

- Edge detection.
- Image processing filters, blurring, sharpening, highlighting, antialiasing and so on.
- Did you implement a pipeline of filters?
- Did you do anything special for accuracy? Speedup? Workload reduction?
- Did you use any other special techniques to improve the system?
- Can you adjust the parameters to specify different methods of operation? Did you implement other (not provided) animations or backgrounds animate your scene? Did you provide some sort of camera motion/fly-through technique?
- Did you provide evidence of testing?

Your documentation for Part 2 should include a tabulation and brief discussion of your results, along with a discussion about any machine and language specific aspects that may affect these results.

3 Assessment

The assessment below is only a guideline, and is indicative of the project assessment. However, the assessment for individual projects may deviate from this in some ways, dependant on the form of the delivered project. On **Friday 3rd March 17:00**, the project is due. It will be worth **20/100** of your final mark. The assessment will be done as follows:

- **4/20** Speedup: An assessment of your writeup in Part 1, with respect to your analysis of the hardware effects, and the two aspects of speedup.
- **2/20** Accuracy: An assessment of your writeup in Part 1, with respect to your analysis of the accuracy effects.
- **2/20** Communication, speedup: An assessment of your writeup in Part 1, with respect to your analysis of the two aspects of speedup on a multi-processor machine.
- **12/20** Implementation: An assessment of the implementation and associated documentation of Part 2.

In general, better assessments will be given to writeups which encapsulate more complex ideas.

3.1 Finally...

Your final completed project should be submitted to the Project1 IVLE workbin in the form of a single zipped up file name A00XXXXXX.zip (i.e. your-student-number.zip), containing your writeup, commented sources and any other supporting files, on or before Friday 3rd March 17:00. You must also submit a readable **hardcopy** to your lecturer.

You are allowed to discuss the problems with your friends, and to study any background material with them, but the project *should be your own work*. **Copying** and **cheating** will be grounds for failing the project.