CS/SE 4F03 Assignment 3

18 February 2016

Due date: 8 March

1 Introduction

In a genetic algorithm, a population of individuals evolves over time. The stronger individuals reproduce, and the weaker ones die. The surviving ones can also mutate. With each individual, there is a fitness number which indicates how strong that individual is. For more details, see https://en.wikipedia.org/wiki/Genetic_algorithm. A genetic algorithm terminates when a given number of iterations is reached or when a solution is found, that is, an individual with certain fitness.

The goal of this assignment is to compute an image (using a genetic algorithm approach) that is close to a given image. The idea is from
http://jdevh.com/2012/05/19/evoart-creating-art-using-genetic-algorithms/.

2 Details

You are given C code
http://www.cas.mcmaster.ca/~nedialk/COURSES/4f03/private/A3code.zip
that takes as input an image in P3 ppm format, number of generations, and population size, and generates an image in P3 ppm format; see `main.c`. The population size must be a multiple of 4.

The main work is done by the function `compImage` in `compimage.cc`. This file is self explanatory. The function for generating a random image is in `randimage.cc`.

Computing fitness. Denote the input image by A and an arbitrary image by B. Denote a pixel (i, j) in A by $A(i, j)$ and in B by $B(i, j)$. The distance between these two pixels is

$$d(i,j) := [A(i,j).r - B(i,j).r]^2 + [A(i,j).g - B(i,j).g]^2 + [A(i,j).b - B(i,j).b]^2$$

where r, g, b denote the values for red, green, and blue. This distance is computed by `pixelDistance` in `fitness.c`. The fitness of B is

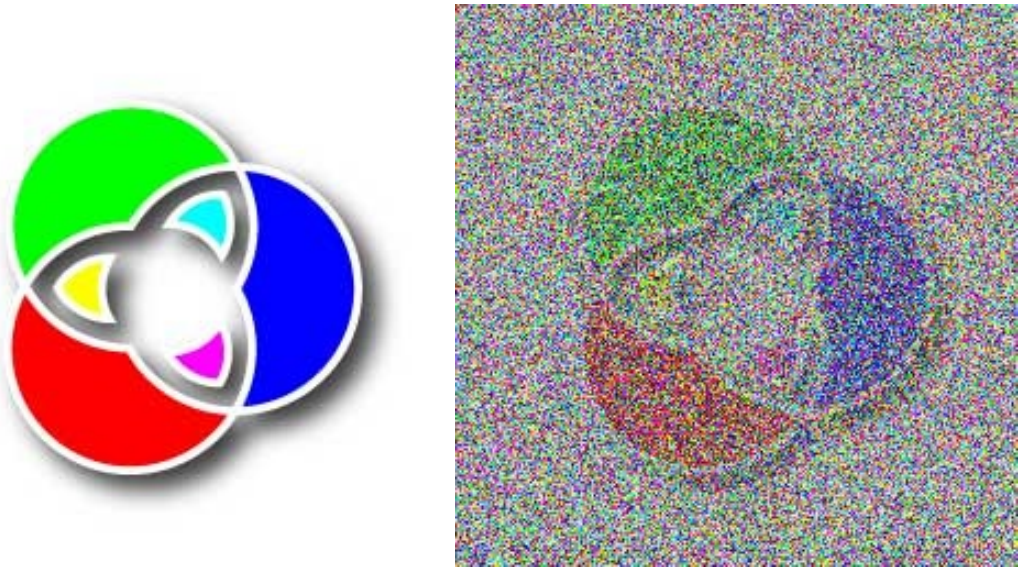
$$f(A, B) = \sum_{i,j} d(i, j).$$

This is comuted by `compFitness` in the same file.

Reproduction. Two parents "split" at a random point to produce two children. This is done in the function `mate` in `mate.c`.

Mutation. File `mutate.c` contains the code for mutation. We select some number of pixels (`rate` pixels) at random and change their RGB values randomly.

Examples Below on the left is an input image and on the right is an image produced by `./genimg image2.ppm image2_10000_1000.ppm 10000 1000`



That is, using 10,000 generations and a population size of 1000. (These images are in `A3code.zip`.)

3 Work to be done

3.1 Code optimization

The given makefile produces an executable with name `genimg`. Run this executable through a profiler and try to optimize the code as much as you can. In Linux, you can use `gprof`; on Mac's, Instruments is very easy to use.

3.2 Parallelization

Implement a shared-memory version of the given code using OpenMP. Then implement a GPU version using OpenACC.

Create a makefile, such that when `make omp` is typed, the OpenMP executable is created. Name it `genimg_omp`.

When `make acc` is typed, the executable `genimg_acc` must be created. Ensure that you create this executable on `tesla.mcmaster.ca`. (We will be testing your code on `tesla`.)

3.3 Reporting results

See the list of servers at http://www.cas.mcmaster.ca/support/index.php/Servers.

Run

```
./genimg_omp image2.ppm image2_10000_100.ppm 10000 100
```

on a server with at least 8 cores. Also execute the same but with `genimg_acc` on the GPUs below.

7 points. Complete the following table; give the server name you have used.

server name cores	CPU time	speedup
1		
2		
4		
8		
Quadro FX4800		
Quadro K2000		
Quadro K5000		
K40		

10 points. The best speedup on 8 cores receives 10 points. A speedup of < 2 on 8 cores receives 0 points. Your mark will depend on how your speed up compares to the best one.

10 points. The best time on K40 receives 10 points, and your mark will depend on how your time compares to the best one on K40.

3.4 Discussion

5 points Discuss your speed up results. In particular, can you explain the different timings on the GPUs?

4 What to submit

- Hardcopy containing
  - the above table
  - the above discussion
  - the C files that are parallelized
- Your code to SVN under directory with name `A3`. Ensure that when `make omp` and `make acc` are typed, the corresponding executables are created.

5 Bonus

≤ 5 **points.** The implemented algorithm is not efficient in terms of finding a pleasing image. Could you implement a better algorithm? If so, explain clearly what you have done. Also, submit the code and ensure that when `make` is typed, an executable with name `genimg2_acc` is created. This version should compile and run with OpenACC (OpenMP is not required).

≤ 5 **points.** As the iterations proceed, you can output intermediate images in ppm format. You can convert them immediately to jpg format and put them together into a video that shows the evolution of the pixels. See the utility `convert` from
http://www.imagemagick.org/script/convert.php.

Use the fastest GPU, likely K40, to produce these images. You can use either the given code or your improved version of it (if you produce such).

If you manage to create an attractive video, please put on the opening frame your name (but not student number), the name of the GPU card you have used, and the course name. (This may help you in your job searches). You can upload this video on SVN or/and on youtube and send the link in an e-mail to the instructor and TAs.

Finally, you may want to try this image
http://www.cas.mcmaster.ca/~nedialk/COURSES/4f03/private/mm.ppm.

6 Logistics

You can work in groups of at most two. Only one group member should submit. Please type your explanation part, as we have had difficulties reading some of the previous handwritten submissions.