GPGPU - Project report

Nicolas Portal - Alexandre Yvart - Geoffrey Jount

13 juin 2020

Table des matières

1 Introduction

For educational purpose in the GPGPU subject, we are asked to work on a project using a GPU parallel method. We will use the CUDA library.

2 Problem description

This project's goal is to implement an image segmentation algorithm : the max-flow/min-cut algorithm. A picture containing numerous pixels however, processing one after another in a simple `for` loop would show to be very slow. We thus need to make the process parallel, and implement it on GPU for an even faster result : the *push-relabel* algorithm, parallel version of the first mentioned, is what we seek.

3 CPU implementation

We followed the guidelines found on the NVIDIA presentation.
All in all there are eight matrices :
- for the heights of each node
- for the excess flow of each node
- for the links between the source and each node
- for the links between the sink and each nodes

And the four last matrices for the links between each nodes : top, bottom, right and left neighbours matrices were labeled with the inverse of the euclidean distance in RGB space.
The links between the source and each node and the links between the sink and each node were labelled with the inverse of the average distance to the marked pixels in RGB space. Each node were linked to the source and sink. The height of each node was 0 except for the source which had a height equals to the number of nodes. The `std_load` library was used to load `jpg` images both in grayscale and RGB color space. Images were stored as one dimensional vectors with contiguous red, green and blue pixels. the size of these vectors were three times as large as the number of pixels in an image. The `ppm` file format was used to generate the output segmented image.

3.1 Issue 1

Our implementation soon ran into an infinite loop which remained one of the core issue for several weeks. As a result, the code was changed to make it possible for each node to push towards the sink or the source. This did not solved the issue. The weights of links between the source ans sink on one hand and every node on the other hand was changed using the histogram on red, green and blue channel of the marked pixels to compute a probability. This did not solved the issue as well.

It was found that there was actually no infinite loop but the algorithm was very long to terminate as two nodes would often exchange flow between them waiting for one of them to reach the maximum height. To solve this glitch, it was decided to manually lower the maximum height. The maximum height was no longer set to the number of nodes but rather to a low number such as five or ten. This made it possible for our algorithm to terminate.

3.2 Issue 2

An other issue quickly appeared. Using a depth first search to find nodes yielded bad results as every pixel would be set to white.

When it comes to the way to get the output image, it was decided to set every pixel with a height above zero to one. This did not generate optimal results but it was good enough to make out the segmentation.

A small eight by eight image was used to make the debug step simpler. This image was a ppm image with every pixels on the top part of the image set to blue and every pixel on the lower part of the image set to red. One pixel was set to black and white in each region.

3.3 Issue 3

Although the maximum height was decrease to five or ten, the run time was still to high. It took 15 to 20 minutes for the algorithm to terminate.

Lowering the maximum height resulted in a 15, 20 minute running time. This was still too high. It was found out that calling the push and relabel function on every nodes would decrease the run time significantly. The check for active nodes was now done by the push and relabel function. It was no longer needed to write a separate function to retrieve the right active nodes. It was also no longer required to loop on every pixels to find active nodes. This decreased the run time to 10 seconds which was a huge improvement.

3.4 Issue 4

For a long time we simply checked the height value of each pixel to get the min cut of the graph and thus decide the color of each pixel.

Since we were not satisfied with our output segmented image, it was decided to change the way we recovered the min cut. Although there were very little explanation about the way the BFS algorithm should have been implemented in NVIDIA slides, the algorithm was successfully implemented. It was a little bit of a disapointment though since this did not change the look of the output segmented image. This led us to understand that this non optimal segmented

image resulted from a flaw in the core algorithm (maybe the weights of the links).

4 GPU implementation

5 Bottlenecks

6 Enhancements

6.1 Enhancement 1

6.1.1 Justification

6.1.2 Description

6.1.3 Performance comparisons

7 Benchmarks recapitulation

8 Tasks repartition

Nicolas
- CPU implementation and debug
- Report
- GPU debug

Alexandre
- CPU debug
- GPU implementation

Geoffrey
- CPU debug
- Benchmarking implementation