University of Puerto Rico

Mayaguez Campus

Department of Electrical and Computer Engineering

Project Assignment 2

Prof. Wilson Rivera

ICOM 4036

Sec: 020

Abisai Ramos Rodriguez

802-08-6528

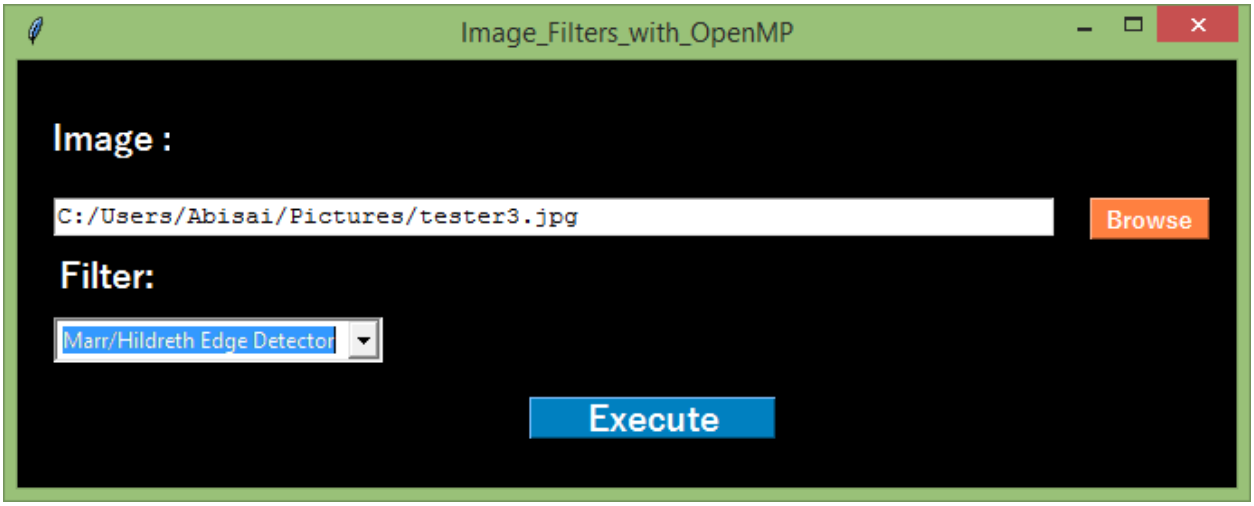
Introduction:

Parallel computing refers to the simultaneous use of multiple compute resources to solve a computational problem. In this assignment the tool used for parallelism was OpenMP. OpenMP is an API that may be used to explicitly direct multi-threaded, shared memory parallelism. The goal of this project was to develop a parallel image processing framework.

Implementations Overview:

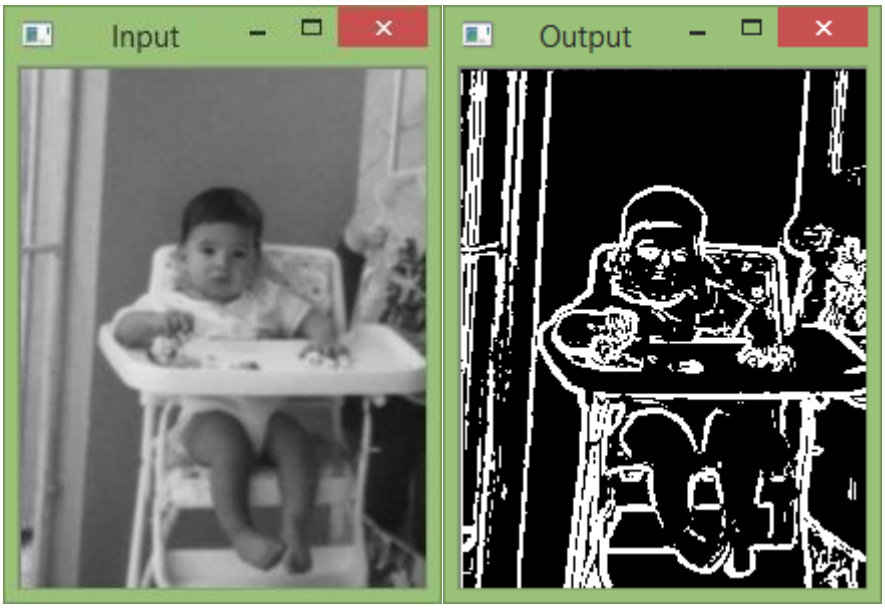
GUI:

The GUI of this assignment was developed in python. The modules names are pl.py and pl_support.py. They were generated with PAGE (Python Automatic GUI Generator).



The GUI allows the user to browse (by clicking the Browse button) for an image and select a filter (from the options inside the combo box). When the execute button is pressed the pl_support gives the given image path and filter to the function applyFilter (from FilterLibs.pyd). This function returns two image windows called input and output.

Example:



FilterLibs.pyd:

This module is the compiled version of imageFilters.cpp which allows python to call applyFilter(String path, int filter). The given image will be transformed in gray scale.

Helper Headers:

displayImage.h :

Implements the function:

```
void show(Mat in, Mat out){
	imshow("Input",in);
	imshow("Output", out);
}
```

imshow is a function from the opencv library, this function creates the input/output frames.

includes.h:

Includes the openMP library and the opencv library.

MarrHildrethED.h:

Implements the function `Mat marrhildreth(Mat src)`. This function applies the Marr-Hildreth Edge Detection algorithm to an input image and returns the output

image. The Marr-Hildreth edge detection algorithm consist of the Laplace of the Gaussian of the input image follow by an edge detection algorithm.

OrderStatsFilters.h:

Implements the following functions:

1. Min
2. Max
3. Min-Max
4. Mean
5. Midpoint
6. Median

All six functions applies the given image filter algorithm to an input image and returns the output image. For each algorithm we create a window (small array) of pixels, the given window is sorted in order to find the desired value (minimal, max, max-min, mean, midpoint, median), the pixel in the middle of the original window will be replaced by the desired value.

Example:

Window:

$$\begin{bmatrix} 9 & 8 & 3 \\ 6 & 2 & 1 \\ 4 & 6 & 5 \end{bmatrix}$$

If we sort this window the result is [1 2 3 4 5 6 6 8 9]. The value of the pixel in the middle of the sorted window is 4, then the median filter of this window will be:

$$\begin{bmatrix} 9 & 8 & 3 \\ 6 & 4 & 1 \\ 4 & 6 & 5 \end{bmatrix}$$

Results and Analysis:

The following analysis where done using an image of size 1920x1080 and applying the filter 5 times:

Filter:	Average execution time (ms) without OpenMP	Average execution time (ms) with OpenMP	OpenMP Performance (Times faster than none parallel algorithm)
Min	0.156	0.037	4.220
Max	0.156	0.039	4.000
Min-Max	0.157	0.034	4.620
Midpoint	0.155	0.032	4.844
Mean	0.0158	0.00604	2.616
Median	0.158	0.041	3.854

From the given table it can be observed that the parallel implementation for each algorithm in average was four times faster than it's none parallel implementation (except for the mean implementation that was 2.6 faster).

Conclusion:

The purpose of this assignment was to construct a parallel image processing frame work. The application consist of a graphical user interface developed with python and OpenMP implementations (developed in C/C++) of the several image processing algorithms. The parallel implementations of the image processing algorithms were in average four times more faster than their none parallel implementation proving that multiple processors (cores) provides increased performance when the algorithms are written to take advantage of parallelism.