A Report on

Parallel Implementation of Image Processing Algorithms

Submitted by

Aparna P L (14IT132)

Neha B (14IT224)

Prerana K R (14IT231)

S S Karan (14IT252)

V Sem B.Tech (IT)

in partial fulfilment of the course

of

Parallel Computing (IT300)

under the guidance of

Dr Geetha V (Asst. Professor)

as a part of

B. Tech in Information Technology

At

Department of Information Technology

National Institute of Technology Karnataka, Surathkal.

November 18, 2016

CERTIFICATE

This is to certify that the project entitled **"Parallel Implementation of Image Processing Algorithms"** is a bona fide work carried out as part of the course **Parallel Computing (IT300)**, under my guidance by

1. Aparna PL 14IT132

2. Neha B 14IT224

3. Prerana K R 14IT231

4. S S Karan 14IT252

students of V Sem B.Tech (IT) at the Department of Information Technology, National Institute of Technology Karnataka, Surathkal, during the academic semester Jul-Dec 2016 in partial fulfilment of the requirements for the award of the degree of Bachelor of Technology in Information Technology, at NITK Surathkal.

Place: Surathkal

Date: 18 Nov, 2016

Signature of the Instructor

Signature of the Examiner

ABSTRACT

Image segmentation is the process of dividing an image into multiple parts. It is typically used to identify objects or other relevant information in digital images. There are many ways to perform image segmentation including Thresholding methods, Colour-based segmentation, Transform methods among many others. Alternately edge detection can be used for image segmentation and data extraction in areas such as image processing, computer vision, and machine vision.

Image thresholding is a simple, yet effective, way of partitioning an image into a foreground and background. This image analysis technique is a type of image segmentation that isolates objects by converting grayscale images into binary images. Image thresholding is most effective in images with high levels of contrast. Otsu's method, named after Nobuyuki Otsu, is one such implementation of Image Thresholding which involves iterating through all the possible threshold values and calculating a measure of spread for the pixel levels each side of the threshold, i.e. the pixels that either fall in foreground or background. The aim is to find the threshold value where the sum of foreground and background spreads is at its minimum.

Edge detection is an image processing technique for finding the boundaries of objects within images. It works by detecting discontinuities in brightness. An image can have horizontal, vertical or diagonal edges. The Sobel operator is used to detect two kinds of edges in an image by making use of a derivative mask, one for the horizontal edges and one for the vertical edges.

ACKNOWLEDGEMENT

A successful and satisfactory completion of any significant task is the outcome of invaluable aggregate combination of different people in radial direction explicitly and implicitly. We would therefore take the opportunity to thank and express our gratitude to all those without whom the completion of our project would not be possible.

We extend our thanks to our project advisor and project co-ordinator, Dr Geetha V, Asst. Professor, Department of Information Technology, NITK, Surathkal, for her constant help and support. We express our heartfelt gratitude to our HoD, Dr. Ram Mohan Reddy, Department of Information Technology, NITK, Surathkal, who has extended support, guidance and assistance for the successful completion of the project.

Aparna PL

Neha B

Prerana K R

S S Karan

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

1.1 Problem Statement and Description

The aim of this project is to perform a comparative analysis of serial and parallel implementations of Gaussian Blurring, Otsu thresholding and Sobel edge detection algorithm in both C (using OpenMP) and Python (using PyMP).

Image thresholding is a simple, yet effective, way of partitioning an image into a foreground and background. Image thresholding is most effective in images with high levels of contrast. Otsu's method, named after Nobuyuki Otsu, is one such implementation of Image Thresholding which involves iterating through all the possible threshold values and calculating a measure of spread for the pixel levels each side of the threshold, i.e. the pixels that either fall in foreground or background.

Edge detection is an image processing technique for finding the boundaries of objects within images. It works by detecting discontinuities in brightness. An image can have horizontal, vertical or diagonal edges. The Sobel operator is used to detect two kinds of edges in an image by making use of a derivative mask, one for the horizontal edges and one for the vertical edges.lts from both

Inorder to obtain better results from the above two algorithms, the images can be de-noised by using the convolution process using a Gaussian mask.

1.2 Method of Parallelization

The sequential algorithms described in sections 2.1, 2.2 and 2.3 have been parallelised using threads. A thread of execution is the smallest sequence of programmed instructions that can be managed independently by a scheduler, which is typically a part of the operating system. A process can have many threads within it, but only one thread will be called the master thread. All these threads can execute concurrently while sharing either all or some of the resources which can be specified in the code. The implementation of resource sharing varies from one platform to another.

The algorithms discussed in the above sections are embarrassingly parallel as they are inherently parallel by nature. The computation of one pixel’s intensity is independent of the computation of

intensity of another pixel. There by parallelizing the sequential algorithms, must lead to a speed up in theory.

The Single Instruction Multiple Data (SIMD) architecture of the Flynn's classification has been used as the same computation is to be performed on all the pixels. Each thread will execute the same instruction but on different pixels of the image.

1.3 Platforms and Tools Used

1.3.1 OpenMP

OpenMP (Open Multi-Processing) is an application programming interface (API) that supports multi-platform shared memory multiprocessing programming in C, C++, and Fortran, on most platforms, processor architectures and operating systems, including Solaris, Linux, OS X, and Windows. It consists of a set of compiler directives, library routines, and environment variables that influence run-time behaviour[1].

1.3.2 PyMP

This package brings OpenMP-like functionality to Python. It takes the good qualities of OpenMP such as minimal code changes and high efficiency and combines them with the Python Zen of code clarity and ease-of-use. Beaing a recent implementation, does not include all features that are provided by OpenMP, but considering the high use of Python in recent times, it is necessary to explore possibilities of parallelizing codes written in Python. Not only because of its excessive utility, but also because it is slow in comparison to basic languages like c/c++[2].

1.3.3 PyCharm

PyCharm is an Integrated Development Environment (IDE) used in computer programming, specifically for the Python language. It is developed by the Czech company JetBrains. It provides code analysis, a graphical debugger, an integrated unit tester, integration with version control systems (VCSes), and supports web development with Django.

CHAPTER 2

SEQUENTIAL ALGORITHM

In this project, parallel implementations of Otsu thresholding for segmentation, and Sobel edge detection algorithm along with their parallel implementations are presented. A pre-processing step of Gaussian blur is also performed for better segmentation results.

2.1 Gaussian Blurring

In image processing, a Gaussian blur (also known as Gaussian smoothing) is the result of blurring an image by a Gaussian function. It is a widely used effect in graphics software, typically to reduce image noise and reduce detail. Gaussian blurring is a convolution process. Convolution process basically, calculates for each pixel, the new value by adding weighted values of the neighbouring pixels. It is represented mathematically as shown below

$$R = \sum_{i=-1}^{1} \sum_{j=-1}^{1} P_{x+i,y+j} \; S_{1+i,1+j}$$

Here, P is the image matrix, S is the mask or the kernel being applied and R is the resultant matrix

The kernel typically used for Gaussian Blurring is shown in Fig 1.1

$$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

Fig 1.1: Kernel for Gaussian blur

This is used as a pre-processing step to segmentation reduce noise in the image to obtain better segmentation results.

The algorithm for applying convolution operation to an image is shown

Sequential Algorithm

```
for each image row in input image:
  for each pixel in image row:
    set accumulator to zero
    for each kernel row in kernel:
      for each element in kernel row:
        if element position  corresponding* to pixel position then
          multiply element value  corresponding* to pixel value
          add result to accumulator
        endif
    set output image pixel to accumulator.
```

2.2 Otsu thresholding

In computer vision and image processing, Otsu's method, named after Nobuyuki Otsu is used to automatically perform clustering-based image thresholding, or, the reduction of a gray-level image to a binary image. The algorithm assumes that the image contains two classes of pixels following bi-modal histogram (foreground pixels and background pixels), it then calculates the optimum threshold separating the two classes so that their combined spread (intra-class variance) is minimal, or equivalently (because the sum of pairwise squared distances is constant), so that their inter-class variance is maximal.

In Otsu's method we exhaustively search for the threshold that minimizes the intra-class variance (the variance within the class), defined as a weighted sum of variances of the two classes:

$$\sigma_w^2(t) = \omega_0(t)\sigma_0^2(t) + \omega_1(t)\sigma_1^2(t)$$

Weights $\omega_{0,1}$ are the probabilities of the two classes separated by a threshold t and

$\sigma_{0,1}^2$

are variances of these two classes. The class probability is $\omega_{0,1}(t)$ is computed from L histograms

$$\omega_0(t) = \sum_{i=0}^{t-1} p(i)$$

$$\omega_1(t) = \sum_{i=t}^{L-1} p(i)$$

$$\begin{aligned}\sigma_b^2(t) &= \sigma^2 - \sigma_w^2(t) = \omega_0(\mu_0 - \mu_T)^2 + \omega_1(\mu_1 - \mu_T)^2 \\ &= \omega_0(t)\omega_1(t)[\mu_0(t) - \mu_1(t)]^2\end{aligned}$$

Otsu shows that minimizing the intra-class variance is the same as maximizing inter-class variance which is expressed in terms of class probabilities ω and class means μ.

while the class mean $\mu_{0,1,T}(t)$ is:

$$\mu_0(t) = \sum_{i=0}^{t-1} i\frac{p(i)}{\omega_0}$$

$$\mu_1(t) = \sum_{i=t}^{L-1} i\frac{p(i)}{\omega_1}$$

$$\mu_T = \sum_{i=0}^{L-1} ip(i)$$

Sequential Algorithm:

1. Compute histogram and probabilities of each intensity level
2. Set up initial $\omega_i(0)$ and $\mu_i(0)$
3. Step through all possible thresholds $t = 1, \ldots$ maximum intensity
    1. Update ω_i and μ_i
    2. Compute $\sigma_b^2(t)$
4. Desired threshold corresponds to the maximum $\sigma_b^2(t)$

2.3 Sobel Edge Detection

The Sobel operator, sometimes called the Sobel–Feldman operator or Sobel filter, is used in image processing and computer vision, particularly within edge detection algorithms where it creates an image emphasising edges. It is named after Irwin Sobel and Gary Feldman[3]. It is a discrete differentiation operator, computing an approximation of the gradient of the image intensity function. At each point in the image, the result of the Sobel–Feldman operator is either the corresponding gradient vector or the norm of this vector. The Sobel–Feldman operator is based on convolving the image with a small, separable, and integer-valued filter in the horizontal and vertical directions and is therefore relatively inexpensive in terms of computations.

The Sobel operator is used to detect two kinds of edges namely, horizontal and vertical edges. The Sobel operator when applied to an image basically uses the convolution process to obtain the edges.

It works on the principle that a change in the gradient of intensity of pixels almost certainly will be an edge of the image, i.e. the areas where there is a significant change in the intensity of the pixels marks a different object in the image.

The horizontal edges are identified using the horizontal mask as shown below, where A represents the image matrix

$$\mathbf{G}_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} * \mathbf{A}$$

The vertical edges are identified using the horizontal mask as shown below, where A represents the image matrix

$$\mathbf{G}_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * \mathbf{A}$$

Now, to obtain the final image that combines both horizontal and vertical images, the following equation is applied to the G_x and G_y matrices.

$$\mathbf{G} = \sqrt{\mathbf{G}_x^2 + \mathbf{G}_y^2}$$

The serial algorithm for applying Sobel edge detection is shown below

Sequential Algorithm:

1. For each pixel in image
    a. Compute the convolved value of the pixel using the horizontal Sobel kernel

   End For

2. For each pixel in image
    a. Compute the convolved value of the pixel using the vertical Sobel kernel

   End For

3. For each pixel in image
    a. Compute the resultant gradient approximation

   End For

CHAPTER 3

WORK DONE

All 3 algorithms i.e

(i) Gaussian Blurring

(ii) Otsu thresholding and

(iii) Edge detection

were successfully implemented in serial and parallel in C, using OpenMP and Python, using PyMP.

3.1 Gaussian Blurring

Convolution operation involves pixel wise computation for the input image. Computation for each pixel are independent of each other, i.e they are embarrassingly parallel and hence the computation on each pixel can be performed simultaneously. For achieving this parallelism, OpenMP for C and PyMP for Python were used which use thread-level parallelism.

Part of the sequential code parallelized:

```
for each image row in input image:
   for each pixel in image row:
     //do some computation
```

This nested for loops are parallelized to compute new pixel values simultaneously.

3.2 Otsu Thresholding

Part of the sequential code parallelized:

Parallel Implementation using OpenMP:

The histogram generation is parallelised.

```
#pragma omp for schedule(dynamic,20)
for (int n=0 ; n<y_size1;n++ )
{
    for (int m=0; m<x_size1; m++){
        h1[ithread*GRAYLEVEL+image1[n][m]]++;
    }
}
#pragma omp for schedule(dynamic,20)
```

```
    for(int i=0; i<GRAYLEVEL; i++) {
        for(int t=0; t<nthreads; t++) {
            hist[i] += h1[GRAYLEVEL*t + i];
        }
    }
```

Maximum Variance Computation is parallelised

```
  #pragma omp for schedule(dynamic,20) reduction(max:max_sigma)
for (i = 0; i < GRAYLEVEL-1; i++) {
    if (omega[i] != 0.0 && omega[i] != 1.0)
sigma[i] = pow(myu[GRAYLEVEL-1]*omega[i] - myu[i], 2) /(omega[i]*(1.0 -
omega[i]));
    else
      sigma[i] = 0.0;
    if (sigma[i] > max_sigma) {
      max_sigma = sigma[i];
      threshold = i;
    }
  }
```

Binarization Output into Image2 is parallelised

```
#pragma omp for collapse(2) private(x,y) schedule(dynamic,500)
for (y = 0;y<y_size2; y++)
  for (x = 0; x < x_size2; x++)
    if (image1[y][x] > threshold)
     image2[y][x] = MAX_BRIGHTNESS;
    else
     image2[y][x] = 0;
```

The method makes an array with dimentions GRAYLEVEL*nthreads. Fills this array in parallel and then merges it into “hist” array without using a critical section. This way the possible data discrepancy is removed.

Figure 3.1 Input image for Otsu (OpenMP)

Figure 3.2 Output image on Otsu

Parallel implementation of Otsu in Python using pymp is as follows:

Parallel code:

```
#/* binarization output into image2 */
x_size2 = x_size1
y_size2 = y_size1

with pymp.Parallel(2) as p1:
    with pymp.Parallel(2) as p2:
        for y in p1.range(0,y_size2):
          for x in p2.range(0,x_size2):
            if (image1[y][x] > threshold):
                      image2[y][x] = MAX_BRIGHTNESS
            else:
                      image2[y][x] = 0
```

The pymp library has limitations as it is a very recent implementation. It does not offer constructs like “critical”. Also performing reduction on arrays is not possible. Hence the code is parallelized where the segmented output image is generated. This is computationally expensive as for a large image of size say 2048x3072, the number of iterations rise to 60,00,000. This region is parallelized and significant improvement is observed.

Figure 3.3 Input image for Otsu (pymp)

Figure 3.4 Output image on Otsu

3.3 Sobel Edge Detection

Parts of the code parallelized were:

1. Nested loops

```
with pymp.Parallel(2) as p1:
   with pymp.Parallel(2) as p2:
         for i in p1.range(1,l+1):
               for j in p2.range(1,b+1):
```

2 . Making sections for computing Gx and Gy simultaneously

CHAPTER 4

ANALYSIS AND DISCUSSIONS

4.1 Gaussian Blurring – OpenMP

Gaussian Blurring algorithm was parallelized using OpenMP and following results were observed.

Time taken sequentially: 0.0058

Time taken in parallel: 0.0115

Speed up: ts/tp = 0.504

4.2 Gaussian Blurring – PyMP

Gaussian Blurring algorithm was parallelized using PyMP and following results were observed.

Time taken sequentially: 0.8796

Time taken in parallel: 0.5658

Speed up: ts/tp = 1.554

4.3 Otsu - OpenMP

It was found that for an image of size around 6MB (size around 2048x3072), the serial implementation was faster than the parallel on OpenMP, reason being the cost to create and maintain threads being more than the time saved by parallel execution of instructions. The Figure 4.1 below shows the time taken for serial and parallel implementation of Otsu using OpenMP.

$$\text{Speed up} = t_s/t_p < 1 \sim 0.05318$$

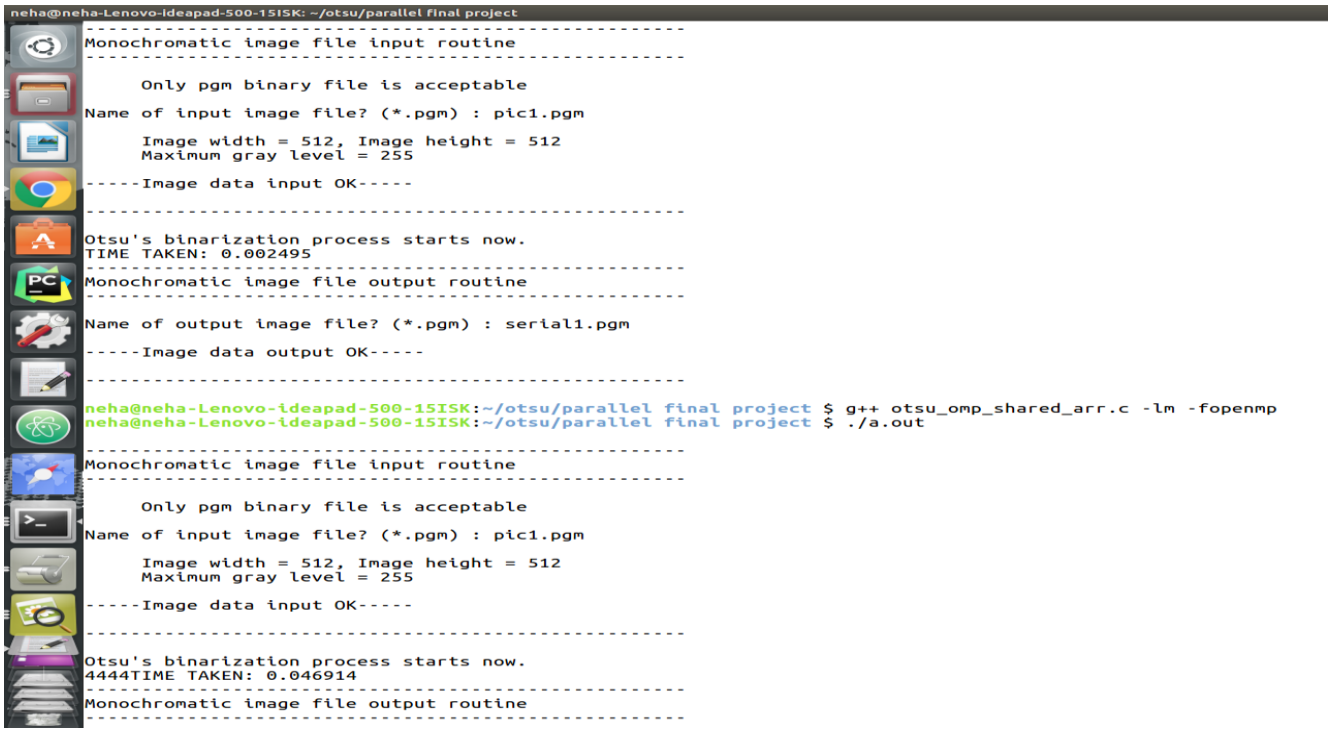

Figure 4.1 Time Taken by Serial and Parallel Implementation of Otsu in C

4.4 Otsu - pymp

Despite the limitations of pymp, the performance improvement of the parallel code in comparison with the serial implementation is very impressive. The screen shots (Figure 4.2 and Figure 4.3) below show the time taken for the execution of the serial (otsu.py) and parallel (otsup.py) on giving two different images as input. One fairly large and another small.

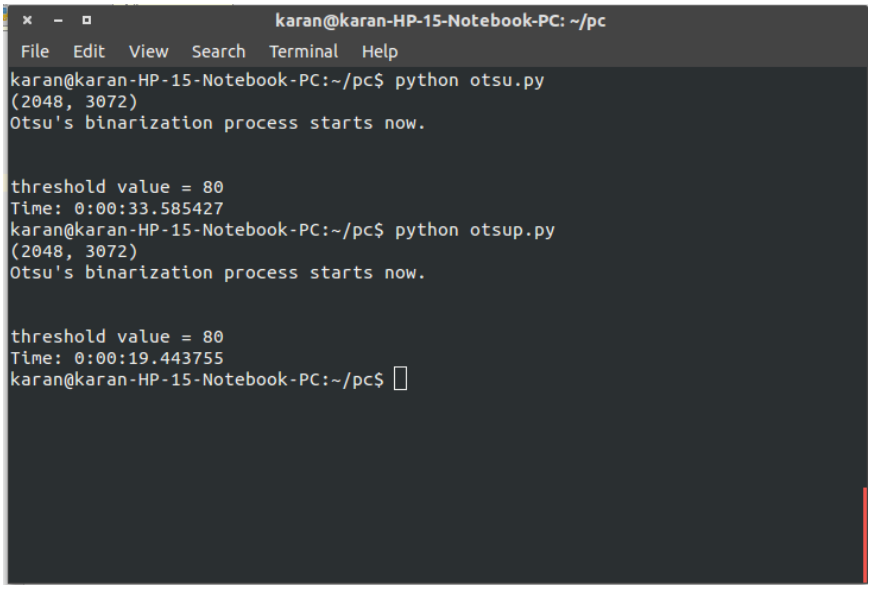

Fig 4.2: Performance Comparison for 2048x3072 Size Image (pymp)

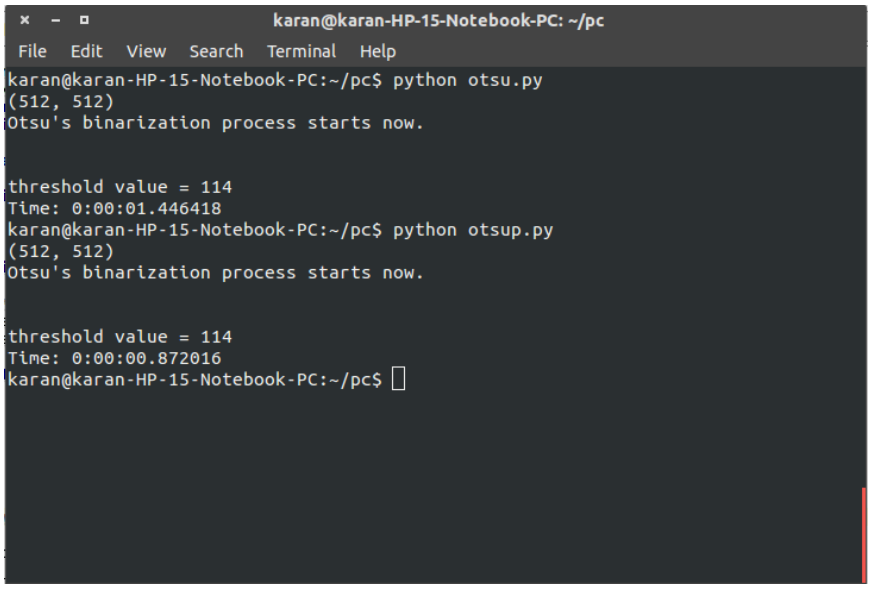

Fig 4.3: Performance Comparison for 512x512 Size Image (pymp)

Python being an interpreted and a language of high level abstraction is very slow in comparison to low level languages like c, which are very fast. Parallelization becomes a more important criteria in this case. We observe that there is a significant reduction in time taken for execution. In case of the small image from 1.45 seconds to 0.87 seconds, and in case of the larger image 33.58 seconds to 19.44 seconds.

For 512X512 image:

Time taken sequentially: 1.45

Time taken in parallel: 0.87

Speed up: ts/tp = 1.67

For 2048X3072 image:

Time taken sequentially: 33.58

Time taken in parallel: 19.44

Speed up: ts/tp = 1.73

4.5 Sobel Edge Detection – PyMP

Sobel Edge detection algorithm was parallelized using PyMP and following results were observed.

Time taken sequentially: 6.22

Time taken in parallel: 3.23

Speed up: ts/tp = 1.86

4.6 Comparison of Python implementations:

4.6.1 Otsu Thresholding

Image Size	*Serial(in seconds)*	*Parallel(in seconds)*	*Speed up*
512X512	1.44	0.88	1.64
768X1024	4.29	2.54	1.69
2048X3072	33.59	19.45	1.73
3464X5200	98.2	60.2	1.63

4.6.2 Sobel Edge Detection

Image Size	*Serial(in seconds)*	*Parallel(in seconds)*	*Speed up*
512X512	22.67	11.07	2.05
768X1024	67.69	32.9	2.06
2048X3072	584.62	283.8	2.06
3464X5200	1678.1	822.6	2.04

4.6.3 Gaussian Blurring

Image Size	*Serial(in seconds)*	*Parallel(in seconds)*	*Speed up*
512X512	3.14	1.77	1.77
768X1024	9.22	5.21	1.77
2048X3072	72.30	41.08	1.76
3464X5200	214.34	122.48	1.75

CHAPTER 5

CONCLUSION

Gaussian blurring, Otsu thresholding and Sobel edge detection algorithms were implemented in C using OpenMP and Python using PyMP. It was found that for C, the speedup was less than one and for python, significant speed up was observed. Python being an interpreted and a language of high level abstraction is very slow in comparison to low level languages like c, which are very fast. Parallelization becomes a more important criterion in this case. We observe that there is a significant reduction in time taken for execution.

REFERENCES

[1] https://computing.llnl.gov/tutorials/openMP/

[2] https://github.com/classner/pymp

[3] https://en.wikipedia.org/wiki/Sobel_operator