Parallel Implementation of Image Processing Algorithms

Aparna PL (14IT132)

Neha B (14IT224)

Prerana K R (14IT231)

S S Karan (14IT252)

Introduction

- In the simplest sense, ***parallel computing*** is the simultaneous use of multiple compute resources to solve a computational problem
- A problem is broken into discrete parts that can be solved concurrently
- Each part is further broken down to a series of instructions
- Instructions from each part execute simultaneously on different processors
- An overall control/coordination mechanism is employed

Problem Statement

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).

Otsu Serial Implementation:

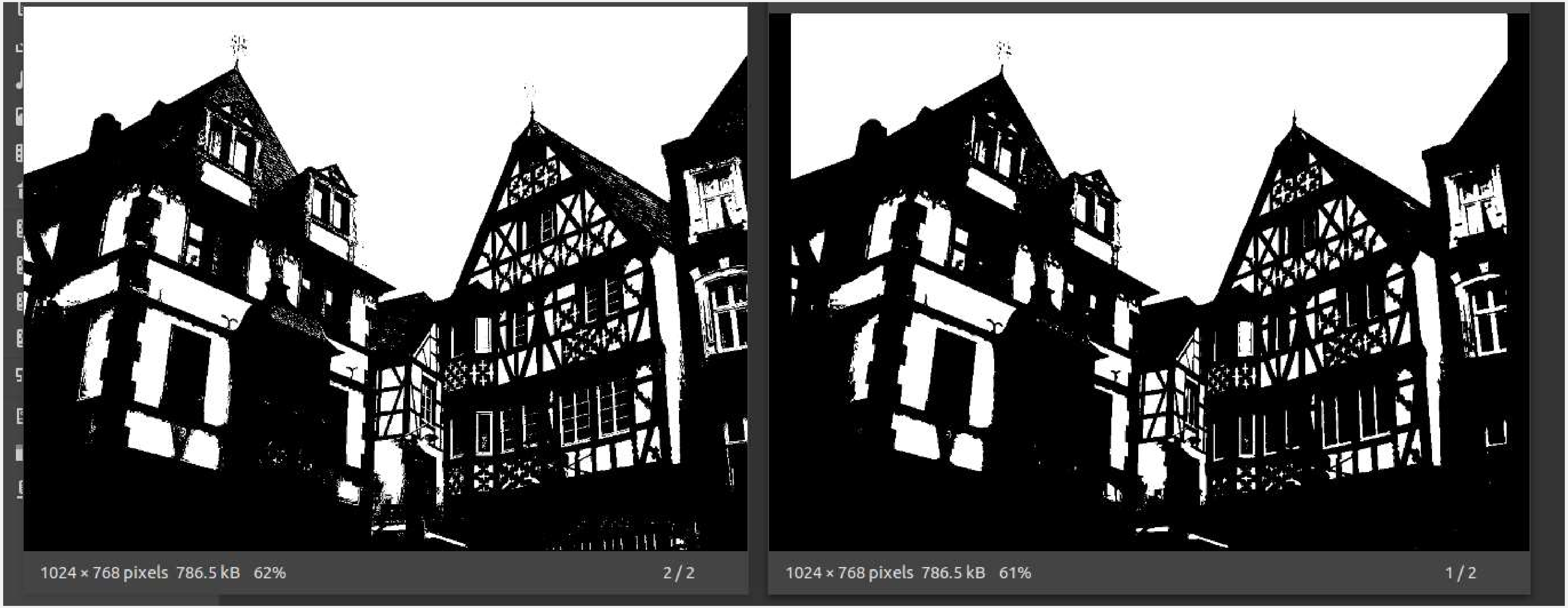
$$\sigma_b^2(t) = \sigma^2 - \sigma_w^2(t) = \omega_0(\mu_0 - \mu_T)^2 + \omega_1(\mu_1 - \mu_T)^2$$
$$= \omega_0(t)\omega_1(t)[\mu_0(t) - \mu_1(t)]^2$$

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 $\sigma_b^2(t)$

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 $\omega_{0,1}(t)$ is computed from the L histograms:

Gaussian Blur (Pre-processing)



Otsu using OpenMP

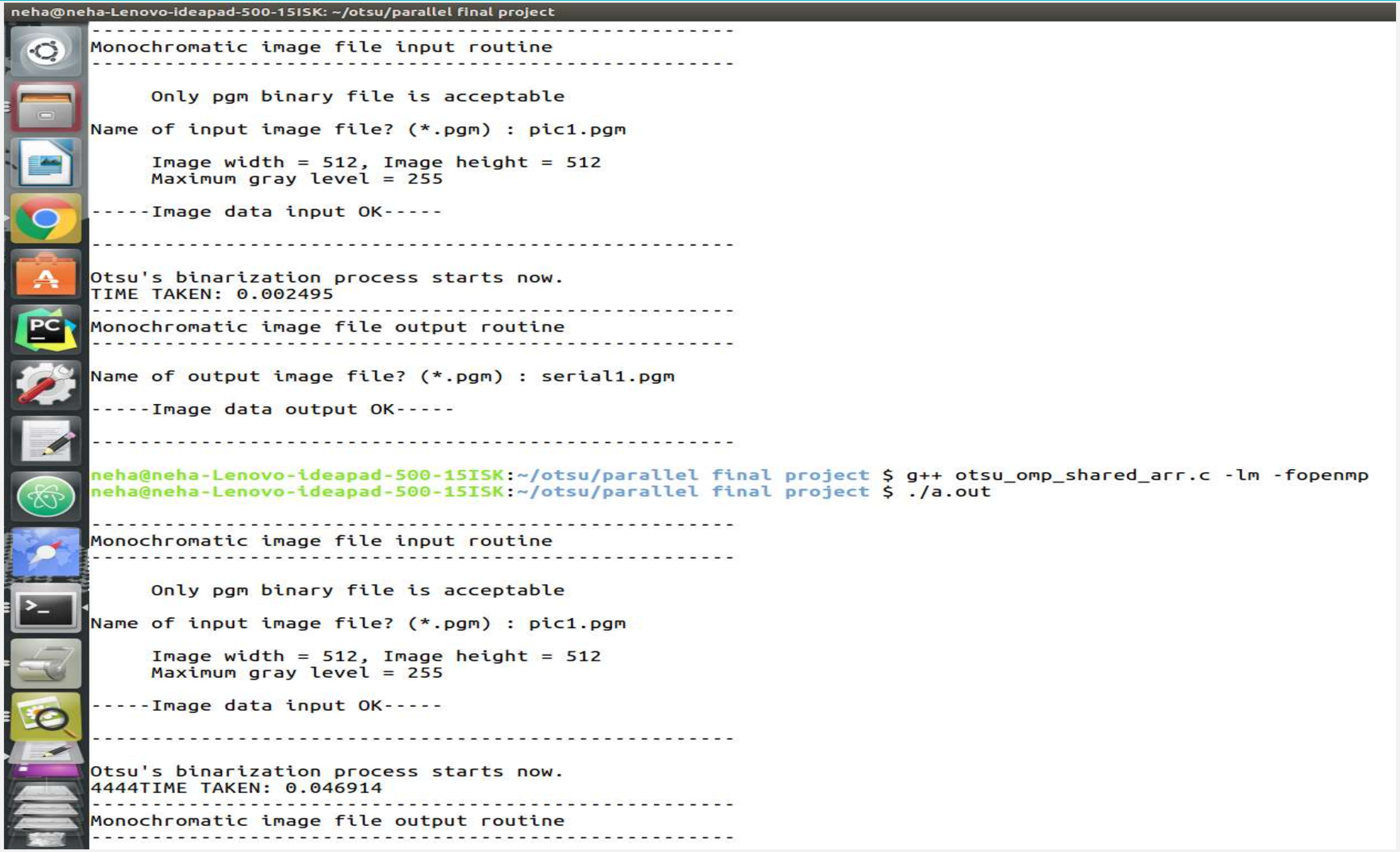
```c
    int* h1;
  #pragma omp parallel
{

    const int nthreads = omp_get_num_threads();
    const int ithread = omp_get_thread_num();
    printf("%d",nthreads);
    #pragma omp single
    {
       h1 = new int[GRAYLEVEL*nthreads];
        for(int i=0; i<(GRAYLEVEL*nthreads); i++) h1[i] = 0;
    }
    #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];
        }
    }
    #pragma omp for schedule(dynamic,20)
      for ( i = 0; i < GRAYLEVEL; i ++ ) {
        prob[i] = (double)hist[i] / (x_size1 * y_size1);
      }
    #pragma omp single
{  omega[0] = prob[0];
  myu[0] = 0.0;         /* 0.0 times prob[0] equals zero */
  for (i = 1; i < GRAYLEVEL; i++) {
    omega[i] = omega[i-1] + prob[i];
    myu[i] = myu[i-1] + i*prob[i];
  }}
  threshold = 0;
  max_sigma = 0.0;
  #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;
```

tsu_omp_shared_arr.c | Windows (CR+LF) | WINDOWS-125

```c
    if (sigma[i] > max_sigma) {
      max_sigma = sigma[i];
      threshold = i;
    }
  }}
/* binarization output into image2 */
  x_size2 = x_size1;
  y_size2 = y_size1;
    #pragma omp for collapse(2) private(x,y) schedule(dynamic,5000)
        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;

/*
    gettimeofday(&TimeValue_Final, &TimeZone_Final);
    time_start = TimeValue_Start.tv_sec * 1000000 + TimeValue_Start.tv_usec;
    time_end = TimeValue_Final.tv_sec * 1000000 + TimeValue_Final.tv_usec;
    time_overhead = (time_end - time_start)/1000000.0;
    printf("\n\n\t\t Time in Seconds (T) : %lf",time_overhead);*/
}


main( )
{
  load_image_data( ); /* input image1 */
  clock_t begin = clock();
  otsu_th( );           /* Otsu's binarization method is applied */
```

otsu_omp_shared_arr.c | Windows (CR+LF)

Otsu : C Analysis



```
neha@neha-Lenovo-ideapad-500-15ISK: ~/otsu/parallel final project
-----------------------------------------------------
Monochromatic image file input routine
-----------------------------------------------------

     Only pgm binary file is acceptable

Name of input image file? (*.pgm) : pic1.pgm

     Image width = 512, Image height = 512
     Maximum gray level = 255

-----Image data input OK-----

-----------------------------------------------------

Otsu's binarization process starts now.
TIME TAKEN: 0.002495
-----------------------------------------------------
Monochromatic image file output routine
-----------------------------------------------------

Name of output image file? (*.pgm) : serial1.pgm

-----Image data output OK-----

-----------------------------------------------------

neha@neha-Lenovo-ideapad-500-15ISK:~/otsu/parallel final project $ g++ otsu_omp_shared_arr.c -lm -fopenmp
neha@neha-Lenovo-ideapad-500-15ISK:~/otsu/parallel final project $ ./a.out

-----------------------------------------------------
Monochromatic image file input routine
-----------------------------------------------------

     Only pgm binary file is acceptable

Name of input image file? (*.pgm) : pic1.pgm

     Image width = 512, Image height = 512
     Maximum gray level = 255

-----Image data input OK-----

-----------------------------------------------------

Otsu's binarization process starts now.
4444TIME TAKEN: 0.046914
-----------------------------------------------------
Monochromatic image file output routine
-----------------------------------------------------
```

Output


512 × 512 pixels 262.2 kB 100%
13 / 14

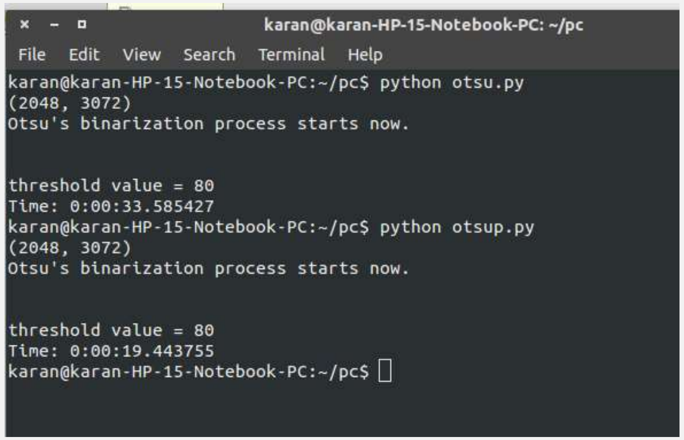

512 × 512 pixels 262.2 kB 100%
10 / 14

Otsu using pymp

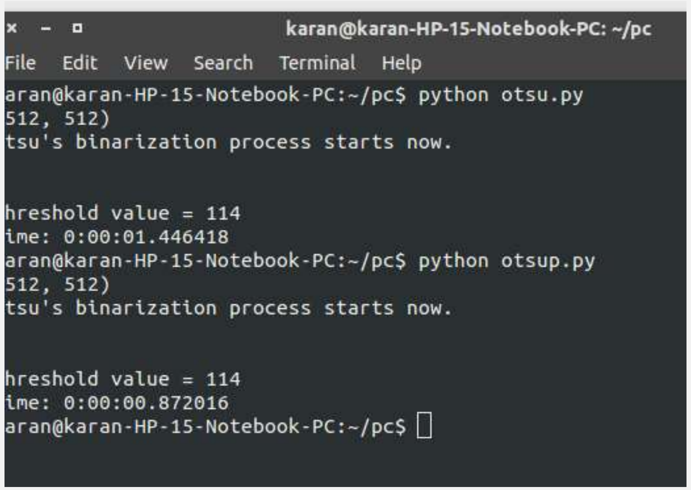
```
  and determines optimal threshold value */'''
threshold = 0
max_sigma = 0.0
for i in range(0,GRAYLEVEL-1):
  if (omega[i] != 0.0 and omega[i] != 1.0):
    sigma[i] = ((myu[GRAYLEVEL-1]*omega[i] - myu[i])**2) / (omega[i]*(1.0 - omega[i]))
  else:
    sigma[i] = 0.0
  if (sigma[i] > max_sigma):
    max_sigma = sigma[i]
    threshold = i

print("\nthreshold value = "+ str(threshold))

#/* binarization output into image2 */
x_size2 = x_size1
y_size2 = y_size1
image2 = pymp.shared.array((y_size1, x_size1), dtype='uint8')
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
print("hi")

a = datetime.datetime.now()
otsu_th()
b = datetime.datetime.now()
print("Time: "+str(b-a))
img = Image.fromarray(image2)
img.save('my.pgm')
img.show()
```

```
#image2=face
print (face.shape)
y_size1=face.shape[0]
x_size1=face.shape[1]
image2 = pymp.shared.array((y_size1,x_size1), dtype='uint8')
hist=pymp.shared.array((256), dtype='uint8')
with pymp.Parallel(4) as p1:
  for i in p1.range(0,GRAYLEVEL):
      hist[i]=0;
prob=[0.0]*256
myu=[0.0]*256
omega=[0.0]*256
sigma=[0.0]*256

def otsu_th():
  print("Otsu's binarization process starts now.\n")
  #/* Histogram generation */
  with pymp.Parallel(4) as p1:
    with pymp.Parallel(4) as p2:
      for y in p1.range(0,y_size1):
        for x in p2.range(0,x_size1):
          hist[image1[y][x]] += 1

  #/* calculation of probability density */
  for i in range(0,GRAYLEVEL):
    prob[i] = float(hist[i]) / (x_size1 * y_size1)
  for i in range(0, 256):
      print("Serial: " + str(prob[i]))
  #/* omega & myu generation */
  omega[0] = prob[0]
  myu[0] = 0.0       #/* 0.0 times prob[0] equals zero */
  for i in range(1,GRAYLEVEL):import datetime

a = datetime.datetime.now()
    omega[i] = omega[i-1] + prob[i]
    myu[i] = myu[i-1] + i*prob[i]
```

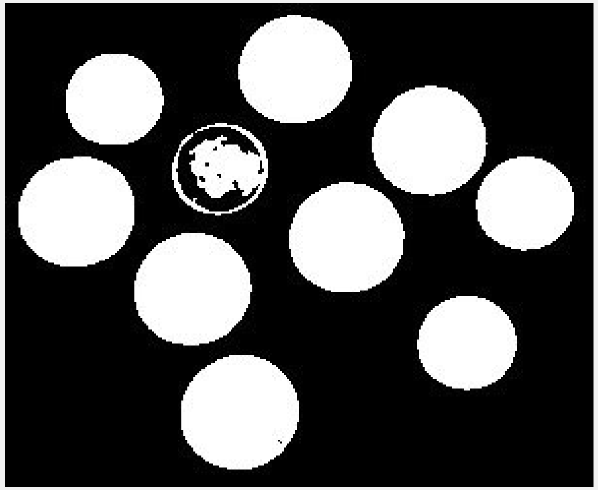
Otsu: Python Analysis



```
karan@karan-HP-15-Notebook-PC: ~/pc
File  Edit  View  Search  Terminal  Help
karan@karan-HP-15-Notebook-PC:~/pc$ python otsu.py
(2048, 3072)
Otsu's binarization process starts now.


threshold value = 80
Time: 0:00:33.585427
karan@karan-HP-15-Notebook-PC:~/pc$ python otsup.py
(2048, 3072)
Otsu's binarization process starts now.


threshold value = 80
Time: 0:00:19.443755
karan@karan-HP-15-Notebook-PC:~/pc$
```



```
karan@karan-HP-15-Notebook-PC: ~/pc
File  Edit  View  Search  Terminal  Help
aran@karan-HP-15-Notebook-PC:~/pc$ python otsu.py
512, 512)
tsu's binarization process starts now.


hreshold value = 114
ime: 0:00:01.446418
aran@karan-HP-15-Notebook-PC:~/pc$ python otsup.py
512, 512)
tsu's binarization process starts now.


hreshold value = 114
ime: 0:00:00.872016
aran@karan-HP-15-Notebook-PC:~/pc$
```



Otsu C Vs. Python

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.

Gaussian Blur

- De-noises the image
- Uses convolution process - add weighted values of neighboring pixels.

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

$$\left(\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} * \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \right)$$

Gaussian Blur using C

```c
load_image_data( ); /* input image1 */
clock_t begin = clock();

int i,j,l=x_size1,b=y_size1,x,y;
x_size2=l+2;
y_size2=b+2;
int **padded = (int **)malloc(sizeof(int *) * (l+2));
for(i=0;i<l+2;i++)
    padded[i] = (int *)malloc(sizeof(int) * (l+2) * (b+2));

int **res = (int **)malloc(sizeof(int *) * (l+2));
for(i=0;i<l+2;i++)
    res[i] = (int *)malloc(sizeof(int) * (l+2) * (b+2));

#pragma omp parallel for collapse(2) private(i,j) schedule(static)
for( i = 0; i<l ;i++)
    for (j = 0; j<b ; j++){
        padded[i+1][j+1]=image1[i][j];
    }

#pragma omp parallel for collapse(2) private(i,j) schedule(static)
for ( i = 1; i<l+1 ;i++)
    for (j = 1; j<b+1 ; j++)
        {
            res[i-1][j-1] = (convx[0][0]*padded[i-1][j-1] + convx[0][1]*padded[i-1][j]+convx[0][2]*padded[i-1][j+1]+
            convx[1][0]*padded[i][j-1]+convx[1][1]*padded[i][j] + convx[1][2]*padded[i][j+1]+
            convx[2][0]*padded[i+1][j-1]+ convx[2][1]*padded[i+1][j] +convx[2][2]*padded[i+1][j+1])%256;
            if(image2[i-1][j-1]>0)
                printf("%d\n",res[i-1][j-1]);
        }

#pragma omp parallel for collapse(2) private(i,j) schedule(static)
for (i = 0; i < l+2; i++)
      for (j = 0; j < b+2; j++)
          {
            image2[i][j] = res[i][j];
          }
   printf("%d\n",image2[l][b]);

clock_t end = clock();
double time_spent = (double)(end - begin) / CLOCKS_PER_SEC;
printf("TIME TAKEN: %f\n",time_spent);
save_image_data( ); /* output image2 */
```

Gaussian Blur using PyMP

```
face = misc.imread('111.pgm')
print (face.shape)
convx = array([[1/16, 2/16, 1/16],
          [2/16, 4/16, 2/16],
          [1/16, 2/16, 1/16]])
l=face.shape[0]
b=face.shape[1]
#padded = np.zeros((l+2,b+2))
padded = pymp.shared.array((l+2,b+2), dtype='uint8')
i=None
j=None
with pymp.Parallel(2) as p1:
        with pymp.Parallel(2) as p2:
                for i in p1.range(0,l):
                        for j in p2.range(0,b):
                                padded[i+1][j+1]=face[i][j]


res = pymp.shared.array((l,b), dtype='uint8')
i=None
j=None
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):
                                res[i-1][j-1] = (convx[0][0]*padded[i-1][j-1] + convx[0][1]*padded[i-1][j]+convx[0][2]*padded[i-1][j+1]+
                                                convx[1][0]*padded[i][j-1]+convx[1][1]*padded[i][j] + convx[1][2]*padded[i][j+1]+
                                                convx[2][0]*padded[i+1][j-1]+ convx[2][1]*padded[i+1][j] +convx[2][2]*padded[i+1][j+1])


img = Image.fromarray(res)
img.save('my.pgm')
img.show()
b = datetime.datetime.now()
print(b-a)
```

Results

Speed up Comparison

For C Code

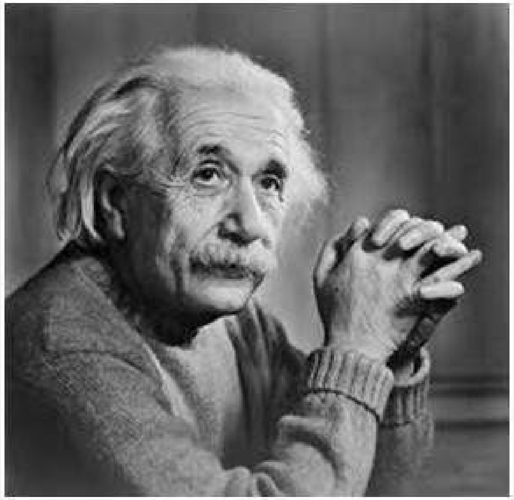
- Time in serial, ts : 0.0053s
- Time in parallel, tp : 0.01152s
- Speed up (ts/tp) : 0.504

For Python Code

- Time in serial, ts : 0.8796s
- Time in parallel , tp: 0.5658s
- Speed up (ts/tp) : 1.554

Sobel edge detection

- Gradient based approach – uses two directions
- Uses convolution technique



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

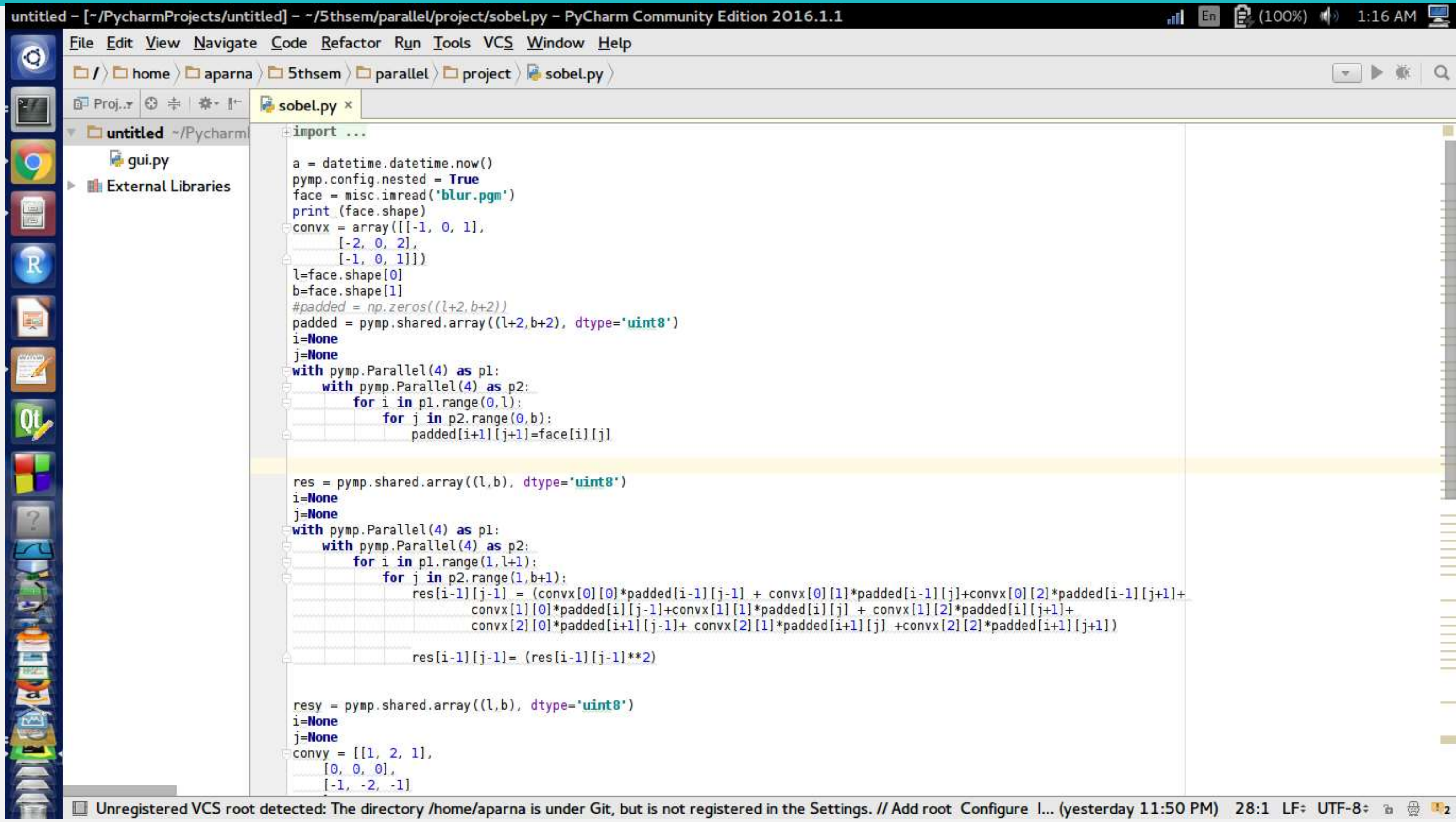
$$\mathbf{G} = \sqrt{{\mathbf{G}_x}^2 + {\mathbf{G}_y}^2}$$

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

Sobel edge detection implementation

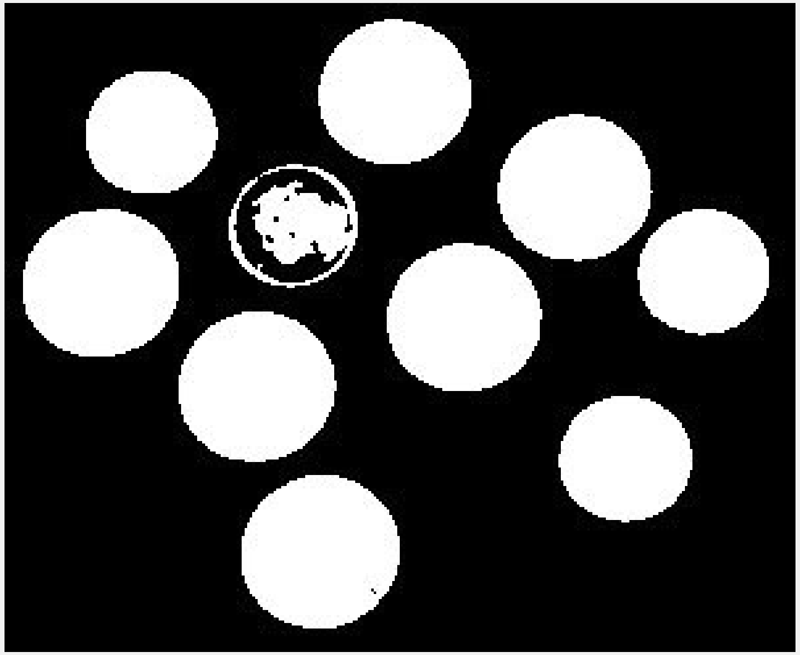
- Implemented serial algorithm in Python
- Used PIL,NumPy, SciPy for image and matrix manipulation
- Used PyMP for parallelizing
- Thread based parallelism
- Used sections for computing gradients in X and Y

Sobel edge detection using PyMP

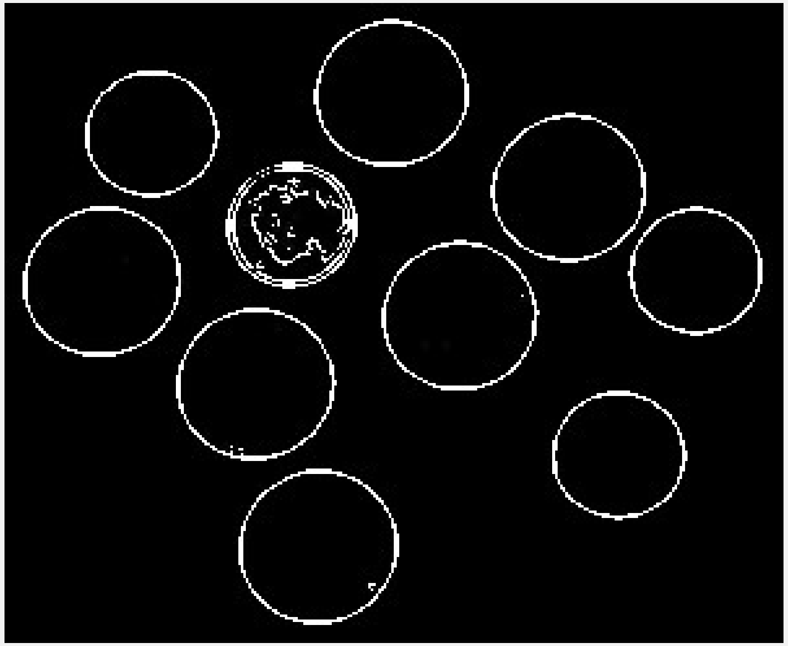


Screenshot of code for Sobel edge detection algorithm in Python

Sobel edge detection using PyMP



Input to edge detection



Output of Sobel edge detection

Edge Detection : Python Analysis

- For an input image of size 246X300
- Execution time in serial: ts = 6. 22s
- Execution time in parallel: tp = 3.33s
- Speedup: ts/tp = 1.86

Individual Contribution

- Otsu, Segmentation Algorithm using OpenMP and pymp:

    Neha,Karan.

- Sobel and Denoising Convolution Algorithm using pymp and OpenMP :

    Aparna, Prerana.

Conclusion

- For Python, using PyMP, speedup > 1
- For C, using OpenMP, speedup < 1

References

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

[2] https://en.wikipedia.org/wiki/OpenMP

Thank You