Parallelising image segmentation

The aim of this project is to perform a comparative analysis of serial and parallel implementation of Otsu thresholding and K- means clustering for image segmentation in Python (PyMP) and C(OpenMP) respectively.

Contributions:

- K Sudheeradh - 18D070016 - K-means analysis
- Sudarshan Gupta - 190040119 - Otsu thresholding
- Anubhaw Kuntal Xess - 150050100
- Yogesh Punia - 160100069

Otsu’s thresholding for image segmentation

Otsu's method 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 bimodal 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.

Otsu Serial Implementation

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 $\sigma_b^2(t)$

Parallel implementation of Otsu in Python using pymp

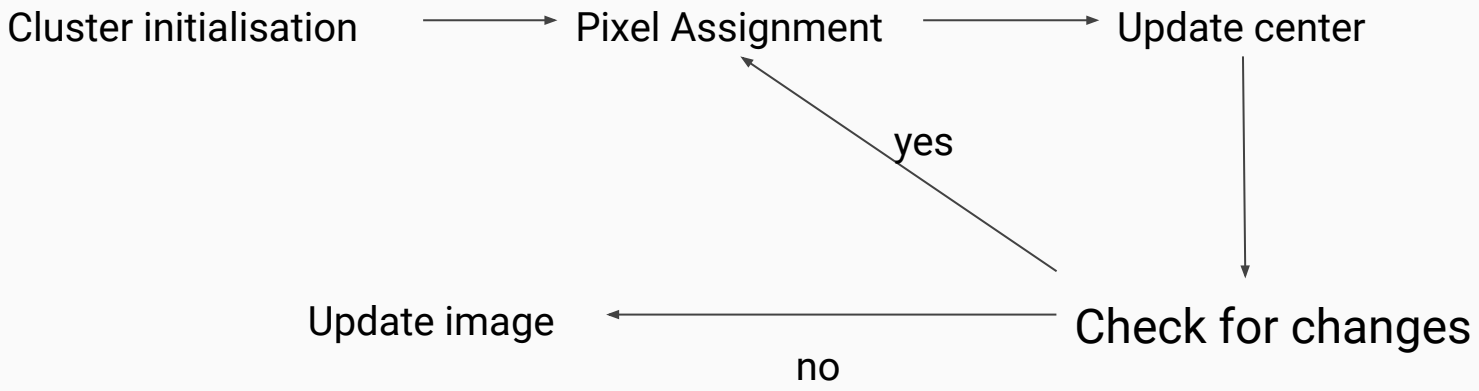
```
#/* binarization output into image2 */
x_size2 = x_size1
y_size2 = y_size1

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
```

Code Runtime

Image Size	747*749		3000*4000	
	Time	Speed up	Time	Speed up
Serial Code	04.349644	-	01:32.969159	-
Parallel (2 cores)	03.565550	1.219	01:17.713817	1.196
Parallel (4 cores)	03.711449	1.172	01:17.542181	1.199
Parallel (6 cores)	03.812567	1.141	01:17.224416	1.204

Parallelising K-Means clustering for Color-based Image Segmentation

Given an image and an integer number K, The k-means clustering algorithm can be used to partition the image pixels into K groups, in such a way that pixels in the same group are similar in terms of color




K = 2

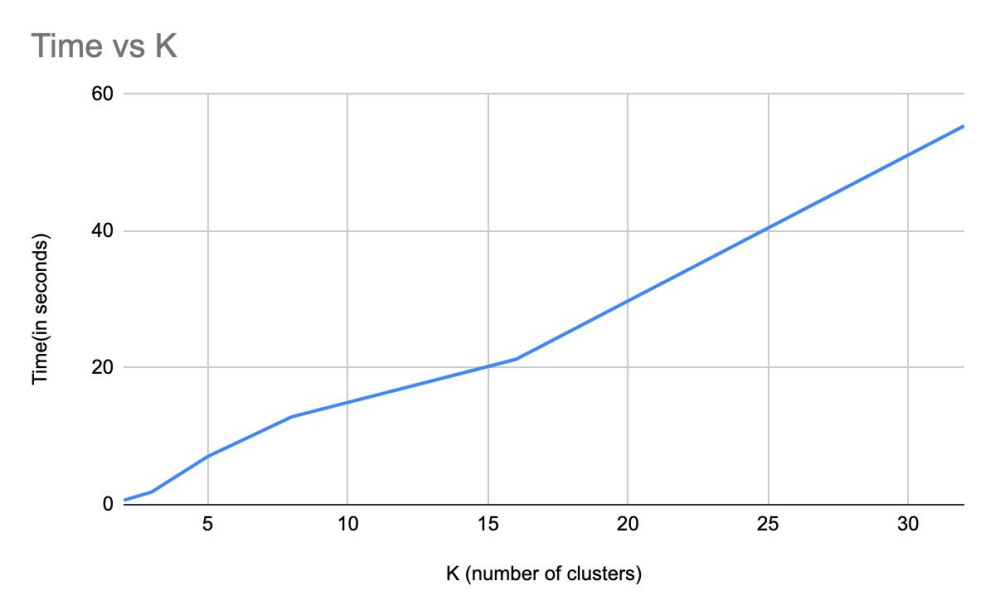

K = 3


K = 5


K = 16

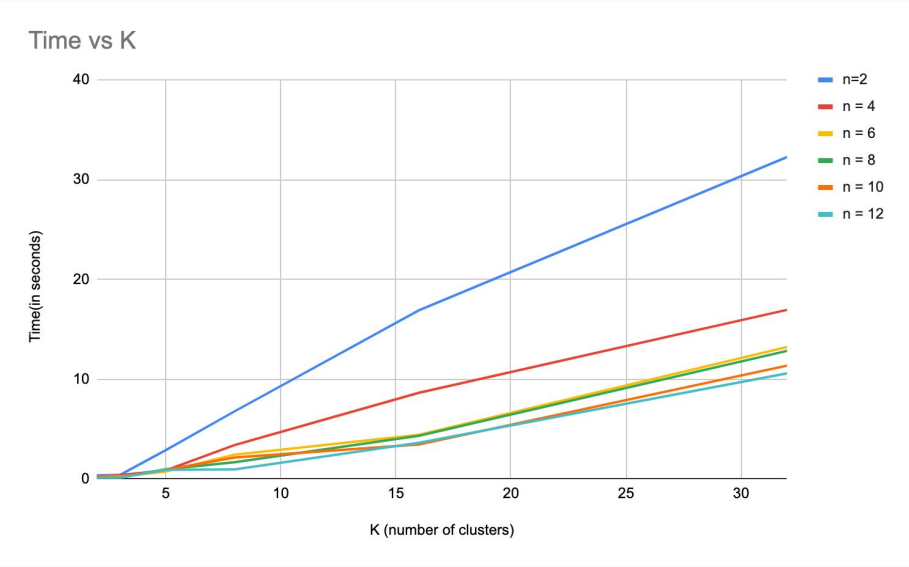
Sequential code runtime

K (number of clusters)	Time(in seconds)
2	0.642773
3	1.857564
5	7.035693
8	12.844987
16	21.2569
32	55.388927



Parallel code runtime

K (number of clusters)	Time(in seconds) n - number of threads					
	n=2	n = 4	n = 6	n = 8	n = 10	n = 12
2	0.372275	0.254146	0.188513	0.172551	0.154262	0.125635
3	0.400883	0.379079	0.240873	0.142251	0.15388	0.13442
5	2.89449	0.875069	0.745475	0.978225	0.927522	0.91979
8	6.797256	3.408044	2.44726	1.687233	2.168198	0.960925
16	16.91064	8.642086	4.425368	4.331817	3.463592	3.634403
32	32.27271	16.950818	13.233873	12.845837	11.363671	10.59308



K (number of clusters)	Speedup n - number of threads					
	n=2	n = 4	n = 6	n = 8	n = 10	n = 12
2	1.726608018	2.5291486	3.409701188	3.72511895	4.166761743	5.116193736
3	4.633681149	4.90020286	7.711798334	13.0583546	12.07151027	13.8191043
5	2.430719401	8.040157976	9.43786579	7.192305451	7.585472905	7.649238413
8	1.889731239	3.769020294	5.248721836	7.613048702	5.924268448	13.36731483
16	1.257013336	2.459695495	4.803419738	4.907155589	6.137241338	5.848801027
32	1.71627753	3.26762561	4.18539055	4.31181923	4.874210719	5.22878398

Machine specifications:
(for k-means analysis)

Machine: MacBook Pro (15-inch, 2019)
Processor: 2.6 GHz 6-Core Intel Core i7
Memory: 16 GB 2400 MHz DDR4

Due to intel hyperthreading,
12(6*2) cores are available