# Raghav Pandya

# Net id: rp835

# Assignment 2

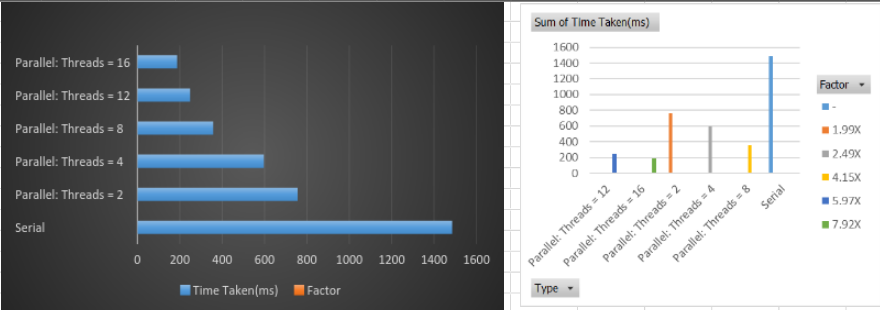
**Problem 2:**

1) Please find attached the code file with option to give any number of threads below 16.

This is done by diving the the image matrix by Total rows/ Number of Threads, so that we have independent datasets

2) Parallelism is achieved by diving the image generation work by separating the image by Total Rows / Number of threads, to get independent datasets which are worked by threads in parallel





The speedup is not linear as we can see from the strong scaling analysis in the next part.

3) We see a variation between the time of different threads proving the hypothesis, but overall when we increase the number of threads, we get a smaller load that is it takes smaller time to get the image codes for the pixels it contains

4) Since different pixel may take different time to compute its color code, we can do a better load balancing by splitting the image into smaller pieces such as individual row so that each worker thread can pick and compute next available bit as it is finished with the previous row. This way we ensure that waiting time is much lower for threads

**Problem 3:**

1) Please find the program attached as a file.



From the results above we see that both OpemMP and pthreads give better performance, with OpenMP better by few ms

2) There are 5 scheduling types:

* Static: Upon entering the loop, each thread independently decides which chunk of the loop they will process.
* Dynamic: No particular order in which loop items are assigned to different threads
* guided
* auto
* runtime

Yes choosing different scheduling type may affect the runtime for example,

Dynamic schedule will improve the runtime as we know different pixel can take different number of iterations to get the color code, and when one thread finsishes, it can be assigned to next iteration that has not been executed yet, but it has some overhead to switch, but overall it works best

3) Strong scaling = t1/ tn \* 100%, Calculating we have, for

n = 2, Strong scaling = 1486/756 ~ 2

n = 4, Strong scaling = 1486/597 = 2.48

n = 8, Strong scaling = 1486/357 ~4

So the speedup is not linear

Weak scaling performed by reducing the problem size with thread = 4,

we have Weak scaling = t1/tn \* 100% = 37/597 = 6%

4) Consider a problem of video processing from camera. This a domain decomposition problem as different workers can work on different frames.

It is not load balanced as some pixel may take longer computation times depending upon their color.

Processing images in a worker thread is asynchronous and it means that accessing the resources by the worker thread and the main thread has to be coordinated. Otherwise, both threads could access the same resource at the same time, what would lead to unpredictable data corruption.

The worker thread of the vision application has a loop. In this loop it grabs images from a camera and does some kind of processing. Images are stored in memory of a buffer as *avl::Image* data. The main thread (UI thread) presents the results of the processing and/or images from the camera. It has to be ensured that the images are not read by the UI thread and processed by the worker thread at the same time.