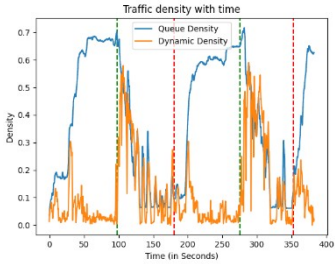
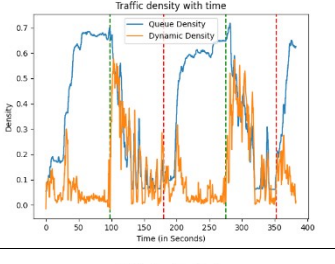
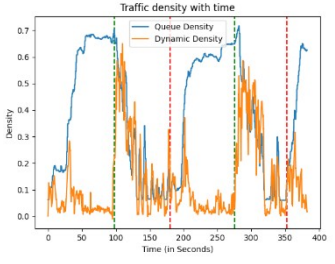
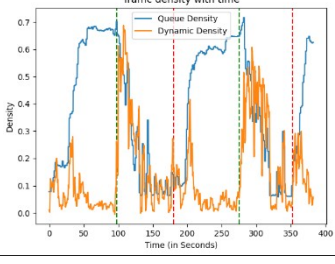
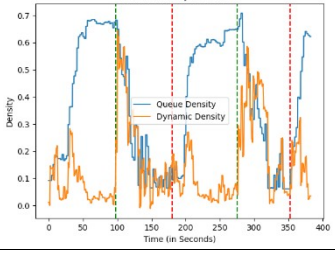
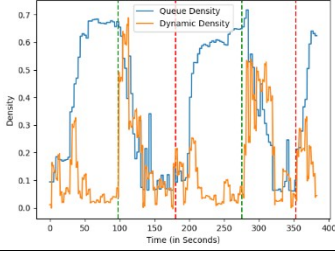
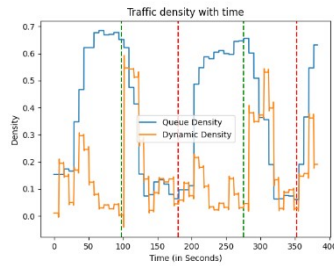


- Method 1

Parameter	Graph	Time
4		118.805
7		76.1824
10		59.0122
20		37.0357
30		30.1309
40		26.636

100

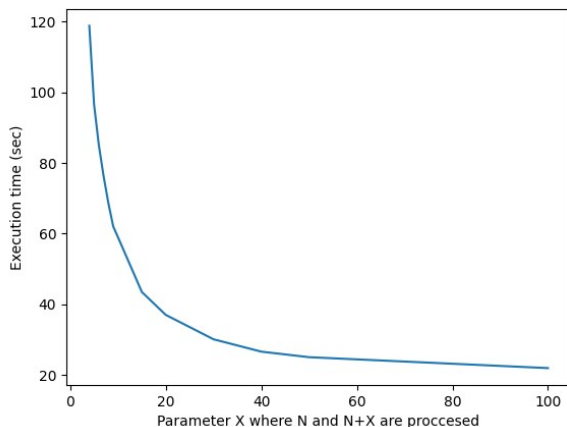


21.9822

Observations:

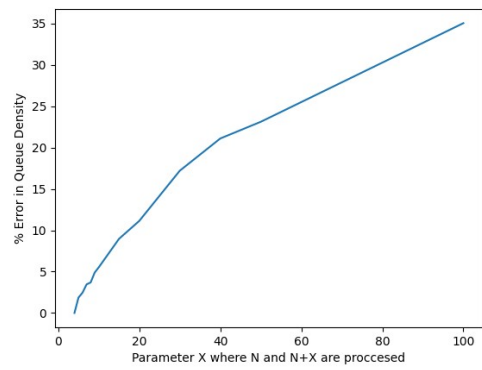
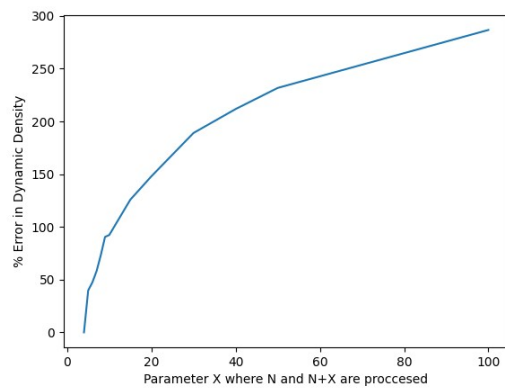
Parameter vs time taken:

Graph follows hyperbolic nature. The decrease in time is exponential and can be accounted to decrease in processing required as number of frames are skipped and previous values are utilised. There is sudden decrease as X parameter decreases by 1 unit. As 1 unit decrease in parameter causes most extensive part of the function to reduce by half. On further decrease in parameter the time taken still decreases but the decrease is less than previous decrease. Hence the graph saturates in the end. This can be attributed to time required for basic operations which are common in every case (ex. Traversing all the frames, cropping in aspect ratio) and will take time.



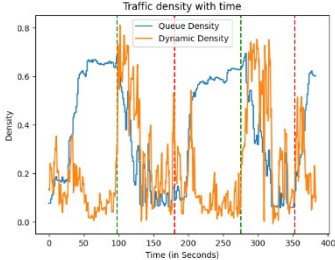
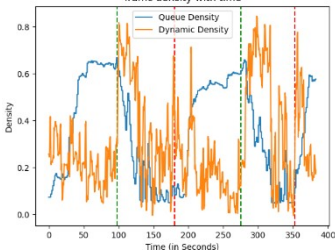
Parameter vs utility:

As X increases the error grows suddenly and utility drops. But after a while error start to saturate. This can explained as follows. In the video traffic remains constant for some duration of time and then there is sudden upsurge due to green signal. During that duration it doesn't matter if we skip 100 frames or 200 frames. Hence there is slight saturation. But error during transition period still builds up we still see growth. Hence error graph is superposition of exponential and linear.



- Method 2:

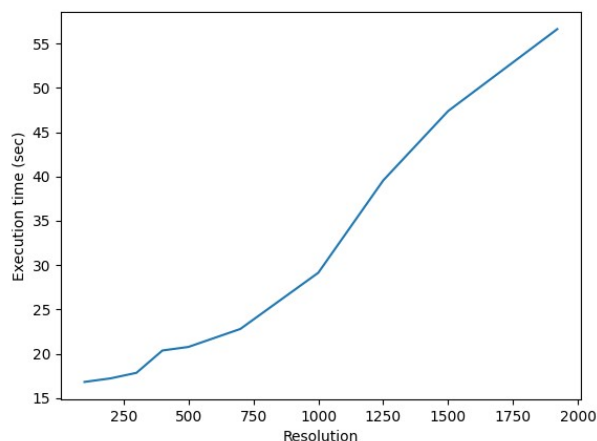
Resolution	Graph	Time
1920 x 1080		56.6293
1000 x 1000		29.1401
700 x 700		22.7988

400 x 400		20.3674
200 x 200		17.2196

Observations:

Parameter vs time taken:

As resolution increase we see increase in time taken. This can be explained as follows. As resolution increases we have higher pixels to process upon. The process frame function is the most costly and determining process. Hence as number of pixels increase the process frame function increases time complexity.



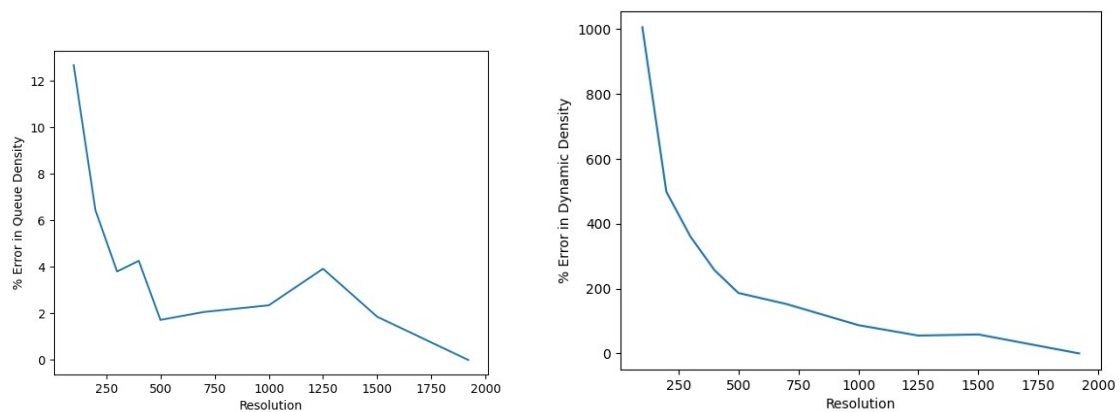
Parameter vs utility:

As resolution increases we see hyperbolic decrease in error percentages and corresponding increase in utility

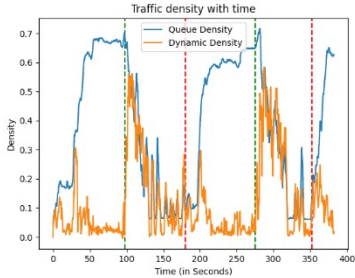
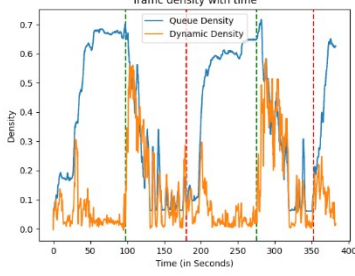
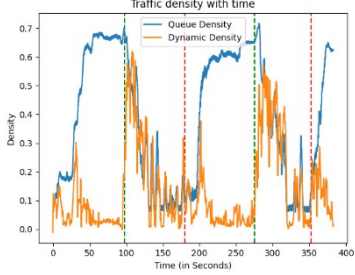
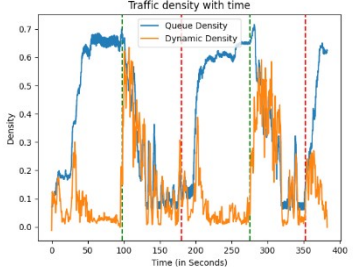
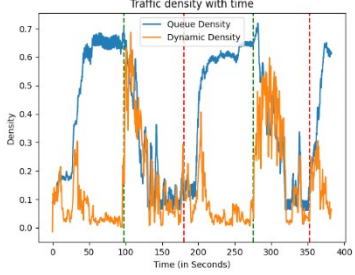
In case of queue density we see very small values of error and hence unexpected peaks in the graph at 1250 resolution. This can be explained as follows. Static density is dependent on removing empty frame from current frame and hence any reduction in both frames nullifies most of the error which still decreases as resolution increases.

In case of dynamic density we are using optical flow which requires to take into account the relative motion of each pixel which is considerably affected as many points are lost in resolution reduction process and gives noise in the process. Also shape of the graph is considerably affected the optical flow builds up any error introduced in the start and causes significant deviation till last frame.

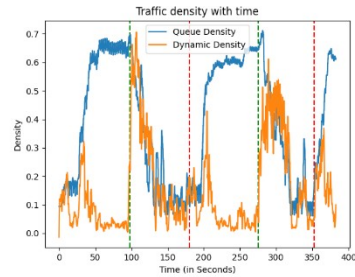
In general both error graphs follows hyperbolic shape which is again expected due to reduced noise as resolution increases. But after the resolution surpasses the original resolution the error is almost constant and zero as no new pixel is introduced



- Method 4:

Number of threads	Graph	Time
Without threads		96.5218
1		97.1115
2		71.1798
3		72.9062
5		102.761

7



134.66

Observations:

Parameter vs time taken:

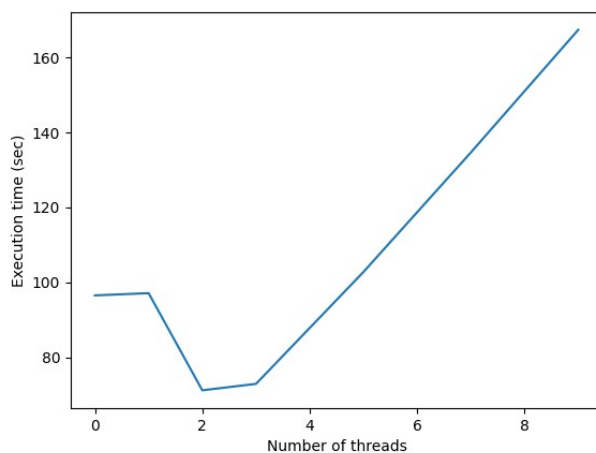
As number of threads increase we see initial decrease in the time and then again increase in the time. This can be explained as follows.

As number of threads increase we have two process going on. As thread number increase we have more workers and hence time taken to process decrease. But in parallel we need to divide the work among all the threads and again wait for synchronisation. This requires more processing.

Also we have individual object creation for each and every threads. As number of threads increase this also take more processing and hence time.

Till a point decrease due to increased work force dominates increase due to division and synchronisation. But after a point we see due to lot of threads, division and synchronisation takes considerable processing and hence increases time duration. After that point we see increase in time.

In this graph we see the time taken increases after thread number 3 and it surpasses the time without multithreading after 4 threads.



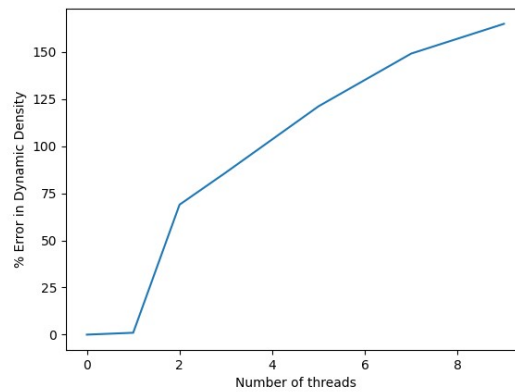
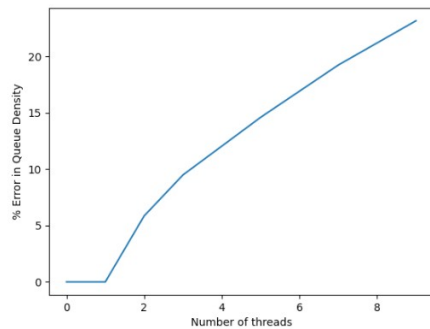
Parameter vs Utility:

We see negative exponential increase in error as number of threads increase. Also error in dynamic is considerably higher than queue. This can be explained as follows.

In this method we are dividing frames across threads. One option was to give certain fraction of video to a thread and certain to other. But the function which sets frame to current frame makes it to traverse through all frames till then and hence takes around 10 secs to just iterate through frames. Hence other option that is processing frames according to modulo value of frame number was considered. That is frame number 0, $0+N$, $0+2*N$... are processed by a thread then 1 , $1+N$, $1+2*N$... are processed by next where N is total number of threads.

In this case we are forced to calculate optical flow and queue density as per current frame and current frame - N th frame. Which induces error especially at the transition points. Hence even though we have very similar shape we get a lot of noise which is relatively small but is almost at every point which builds up the error.

We see expected increase in error with increase in threads.



- Utility of methods:

Utility of errors is taken as relative percentage error evaluated at each point and then taken as average for all points.

$$\text{Error} = 100 * |\text{measured value} - \text{original value}| / \text{original value}$$

High error is observed in each of the cases due to :

1. density values being very small in nature causing small errors to shoot up due to very small denominators
2. Noise in the methods being very high due to pedestrians and processing high number of frames.

Combined effect of noise rises the percentage error to very high levels.

Even though shape of the graph is almost same most of the error may rise upto 300 %.

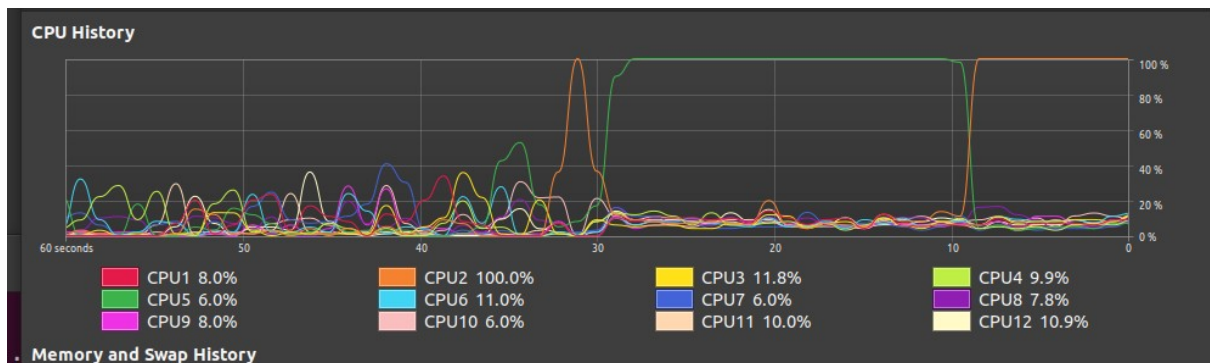
Hence utility can be better decided in relative terms by taking into account the shape of the graph rather than actual values.

Hence we shall focus more on the trend analysis by nature of graph than actual values

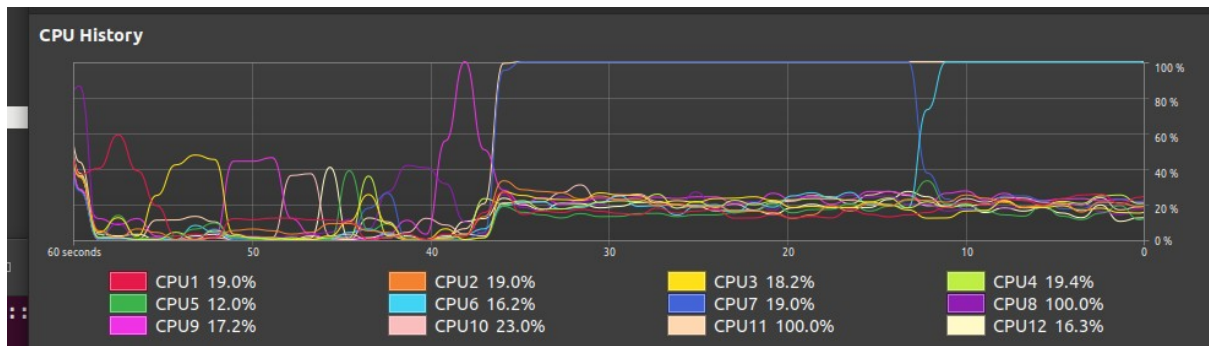
CPU utilisation with threads:

Method 3:

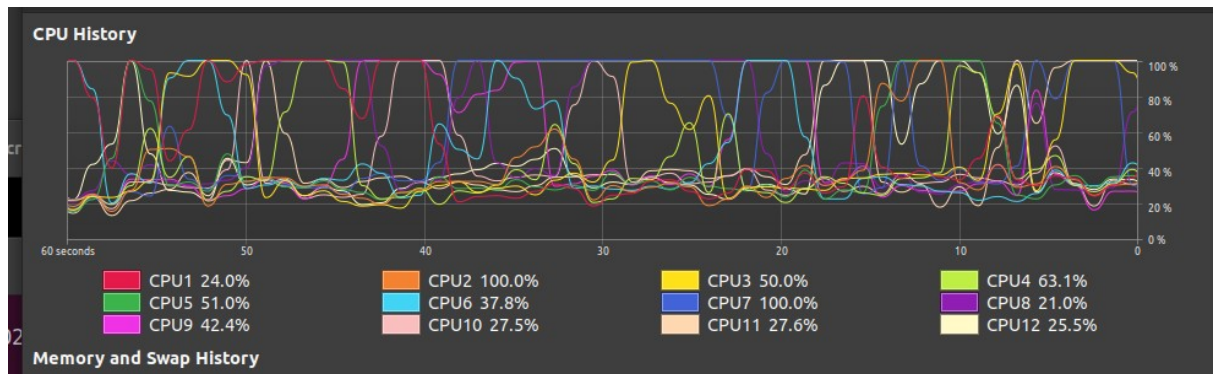
1



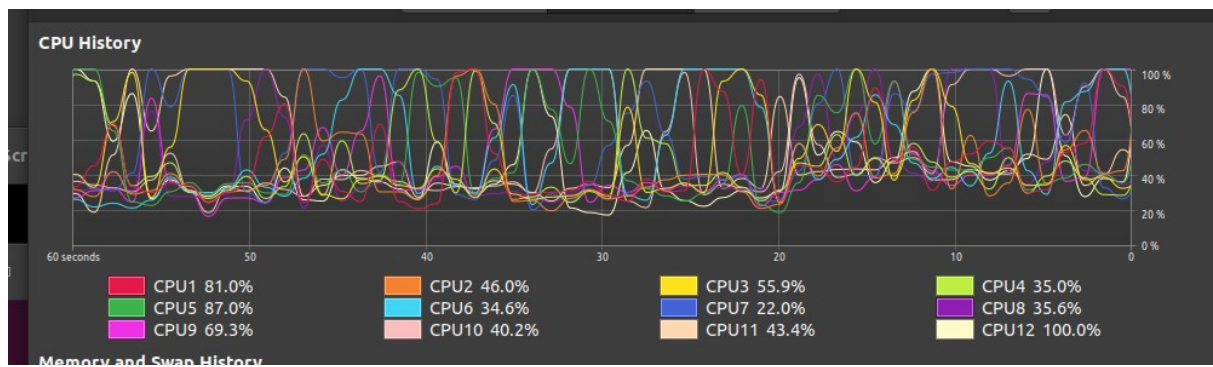
2



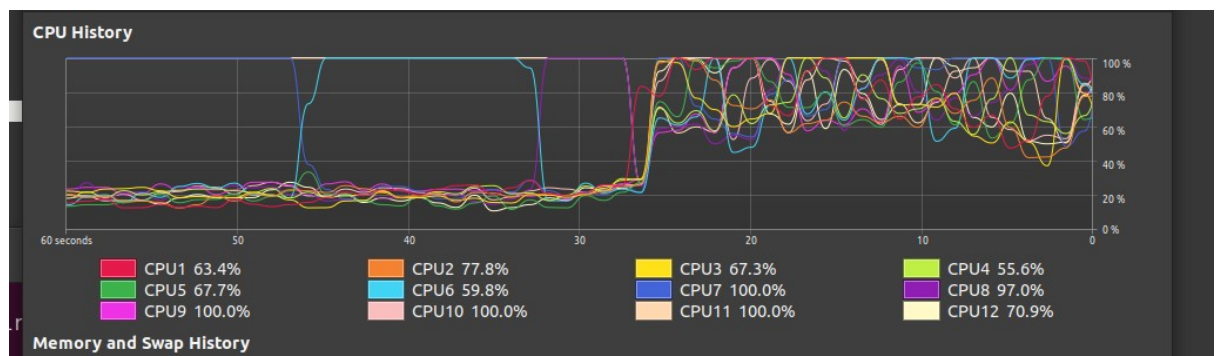
3



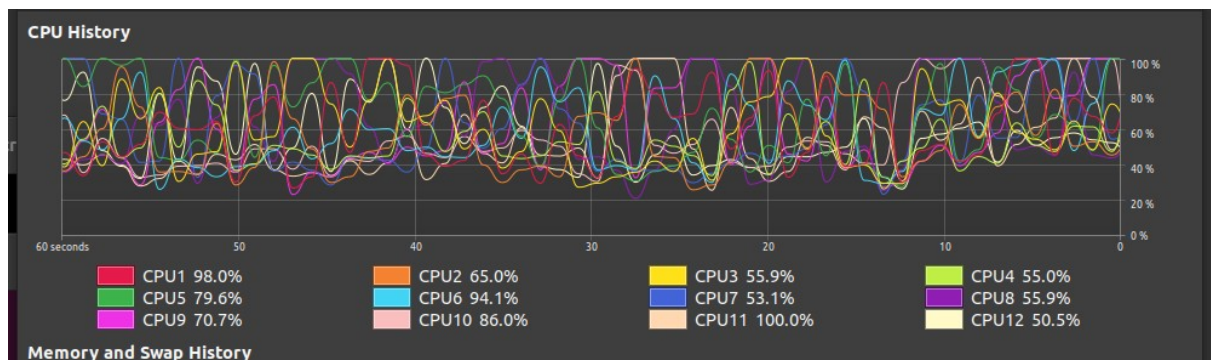
4



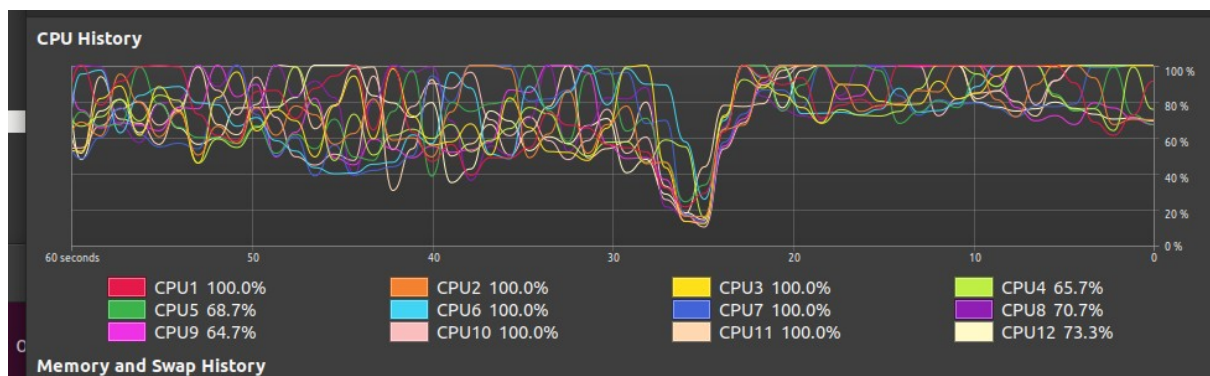
5



6

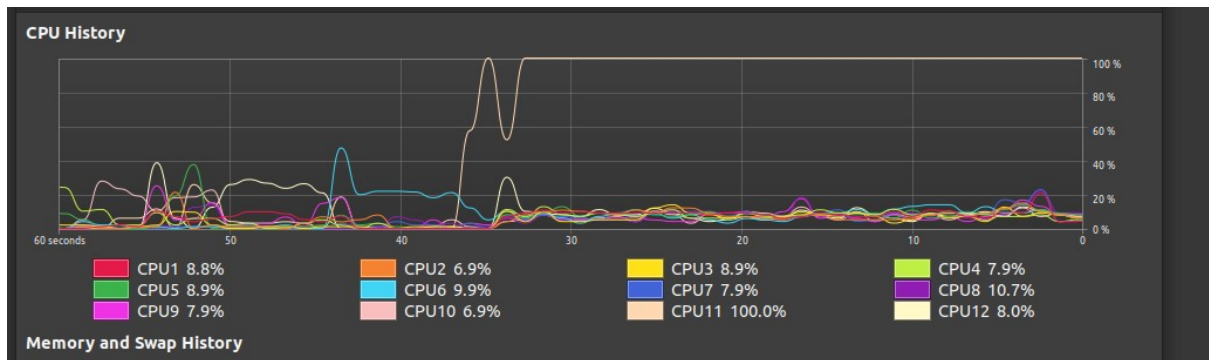


7

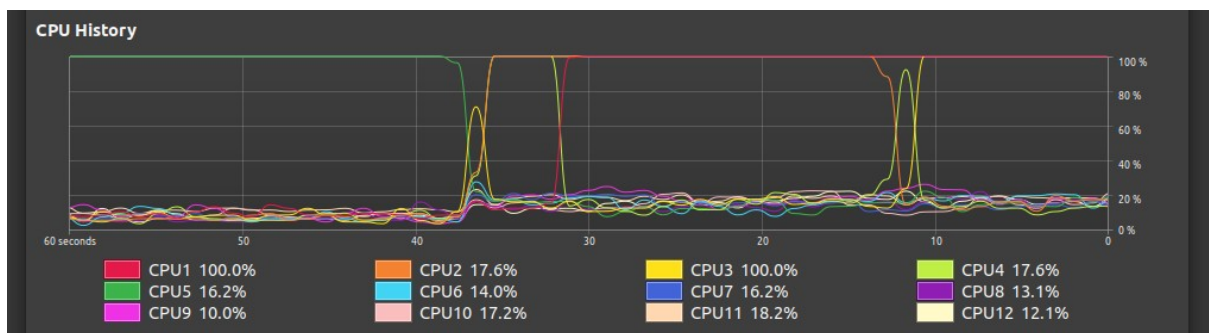


Method 4:

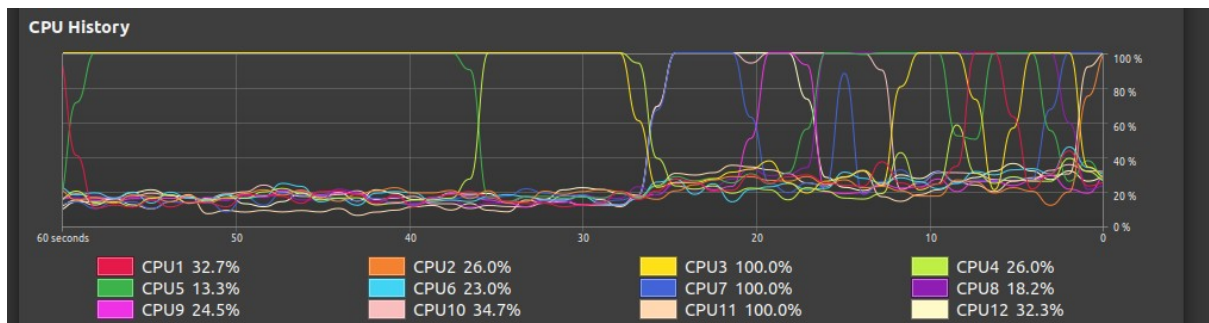
1



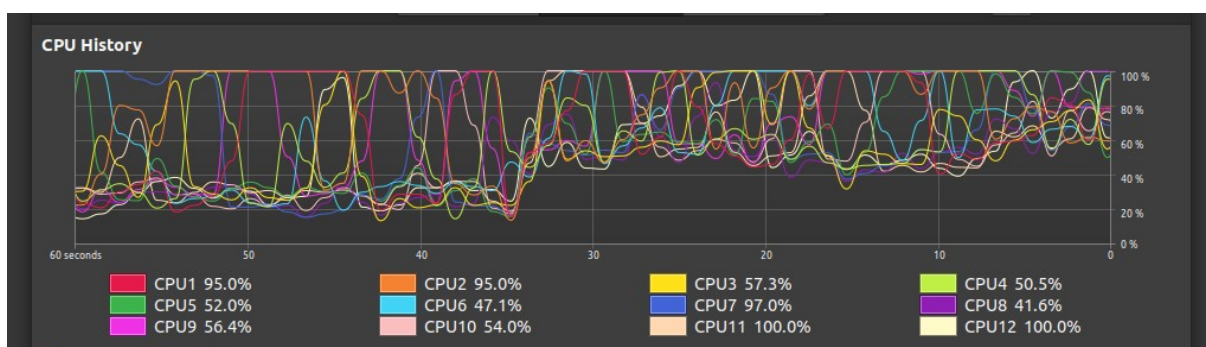
2



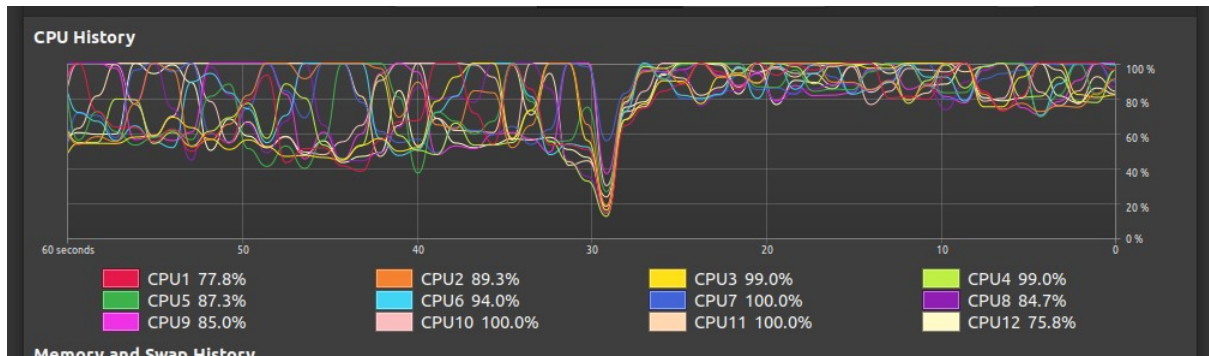
3



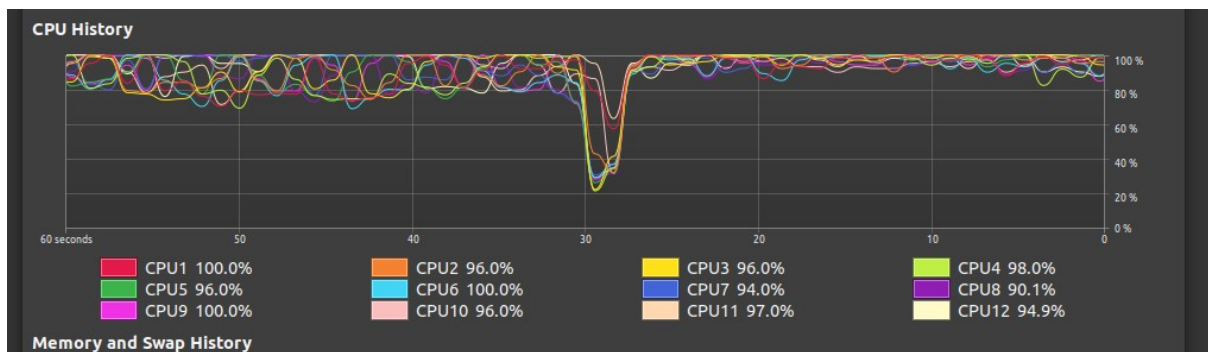
5



7



9



Analysis:

In both the cases we see steady increase as number of threads increase. The CPU however shifts load of cpu from one core to another during execution of a program to avoid overheating. This gives rise to intermediate shifting in core usage from average to 100 and then again falling back to normal

We also see increase in average usage of cores as number of threads increase. This can be attributed to the fact that division of work accross cores needs cpu usage and so is synchronization. As number of threads increase this increase is significant and hence average utilisation of other cores also increases.

In general we see at a given point number of threads used is approximately equal to number of threads close to 100 percent utilization. Also average of not so busy cores is proportional to number of threads used