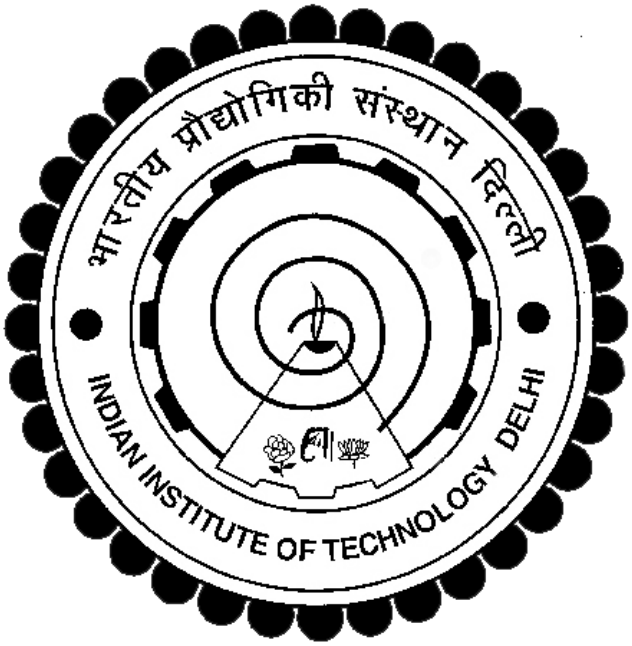


COP290 Design Practices

Task1 Analysis Report

Traffic density estimation using OpenCV functions

Understanding and analyzing trade offs
in Software design

Submitted by-

Pratyush Saini (2019CS10444)
Anirudha Kulkarni (2019CS50421)

Date - 31st March 2021

Prof. Rijurekha Sen
Department of Computer Science and Engineering
IIT Delhi

Metrics

1 Accuracy of the output

The most crucial metric to optimize as a part of any software design is the accuracy of the output. This has been considered in the subtasks 1 and 2. Though the output curves for both Queue density and Dynamic density do not exactly match with the absolute values of the desired output curves (due to difference in the parameters passed to the OpenCV functions), but the pattern of rise and declination of the curves corresponding to different signal changes show similar trend with the desired curve.

2 Latency

Another crucial metric for optimization is the latency metric. Though accuracy of the output is the main concern for us, yet the time required to obtain that output cannot be ignored. Modern software design should be efficient enough to produce accurate outputs within a short time interval, provided we should not compromise with the accuracy or hardware considerations of the systems. The final parameters chosen by us in our main density analysis functions are optimized depending on the time it takes for complete execution, provided the percentage error in the accuracy doesn't fall below a certain threshold.

3 Utility - Runtime time trade-off

For both queue and dynamic density analysis, we defined our utility metric as the percentage deviation from the baseline curve. The percentage deviation is defined as the mean of absolute value of deviation divided by the actual values in the baseline curve for all frames to generate the error percentage.

4 Running Time and CPU Usage

We used the in-built function high_resolution_clock at the start and end of our main function to compute the time taken for execution. Though this may depend on hardware specifications of different functions, yet the trend for Running time vs the parameters chosen is quite fairly analysed. We have used System Monitor available in Ubuntu to analyze CPU usage.

5 Some other crucial terms for trade-off analysis

5.1 Benchmark

Benchmark is the data set on which we analyse the trade-offs. The traffic video from Lajpat Nagar which we have been using from subtask 2 is set as the benchmark for analysing different trade-offs.

5.2 Base-line

Baseline is the method against which we compare other methods/parameters and infer from them whether they produce better or worse trade-off. The part of code we used for subtask2 is set as the baseline for subtask3.

Methods

The various methods used by us for trade-off analysis are as follows:

Method 1: Sub-Sampling Frames

In this method, we modified our density analysis function to process every x frame from the current frame. In other words, we process the frame N+x after processing the current frame N and for all the intermediate values, we just use the values obtained from Nth frame. In our base-line method, we have taken this parameter to be 5 by default. We perform our on analysis for the following values of x:

$$4, 5, 6, 7, 8, 9, 10, 15, 20, 30, 40, 50, 100, 700$$

Method 2: Reduce resolution for each frame

The resolution for a frame is defined by width and height of each frame used for density analysis. The parameters set for this method are X x Y (width and height). This method is used to analyse behaviour of the function after reducing resolution of each of the frames. We expect decrease in running time for the function and increase in the error generated with decreasing frame resolution. We perform our on analysis for the following values of pair X,Y:

$$(100, 100), (200, 200), (300, 300), (400, 400), (500, 500), (700, 700),$$

(1000,1000), (1250,1250), (1500,1500), (1920,1080)

Method 3: Split frames spatially across threads

In this method, we modified our function to split the input frames into multiple horizontal sections and pass each section of frame to a different threads for processing. All the sections of frame have similar dimensions. For each frame, the value for queue and dynamic density is the sum of the values obtained after processing all the sections. The time taken is defined as the time difference between execution of the main threads and after joining all the threads. The

parameter set for this method is the no. of splits or the number of threads used. We performed our analysis for the following parameter values:

1, 2, 3, 4, 5, 6

Method 4: Split work temporarily across threads

In this method, we split out our analysis function across different threads, giving each thread consecutive frames to process. The parameters set for this method are number of threads, similar to Method 3. For ex. if number of threads = 2, frame no. 1,3,5.. are processed by thread 1 and frame no. 2,4,6,8.... are processed by thread 2. We performed our analysis for the following parameter values:

0, 1, 2, 3, 5, 7, 9

Method 5: Estimation using Sparse vs Dense Optical flow

In our baseline density estimation function, we computed the Dynamic density using Dense optical flow, where we monitored each pixel in the current frame and perform our analysis based on movement of pixels in consecutive frames. In this method, we performed dynamic density analysis using Lucas-Kanade Optical flow (or Spare Optical flow) method, where instead of monitoring all the pixels, we monitor only certain pixels (known as special corners) obtained by opencv function function goodFeaturestoTrack. In this function, we give some points to track, we receive the optical flow vectors of those points. But again there are some problems. Until now, we were dealing with small motions. So it fails when there is large motion and a source for large error values.

In this method, we used 4 parameters, which are basically inputs to the function goodFeaturestotrack:

Parameter 1: MaxCorners

This parameter is an upper bound to the maximum number of corners to return. If there are more corners than are found, the strongest of them is returned. We performed our analysis for the following parameter values:

20, 40, 60, 80, 100, 120, 140, 160, 180, 200

Parameter 2: QualityLevel

This parameter characterizes the minimal accepted quality of image corners. The parameter value is multiplied by the best corner quality measure. The corners with the quality measure less than the product are rejected. For example, if the best corner has the quality measure = 500, and the quality Level=0.01 , then all the corners with the quality measure less than 5 are rejected. We performed our analysis for the following parameter values(scaled up 100 times):

2, 6, 10, 14, 18, 22, 26, 30, 34, 38, 42

Parameter 3: minDistance

This parameter characterizes the minimum possible Euclidean distance between the returned corners. We performed our analysis for the following parameter values:

3, 5, 7, 9, 11, 13, 15

Parameter 4: blockSize

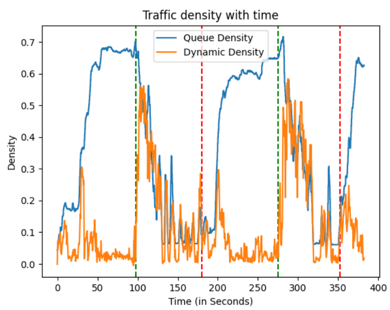
This parameter characterizes size of an average block for computing a derivative co-variation matrix over each pixel neighborhood. We performed our analysis for the following parameter values:

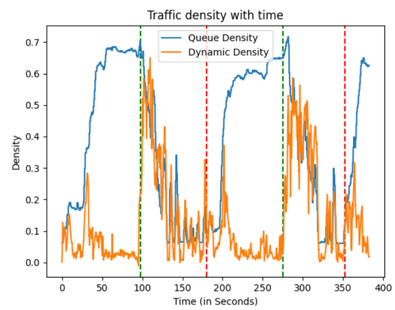
3, 5, 7, 9, 11, 13, 15

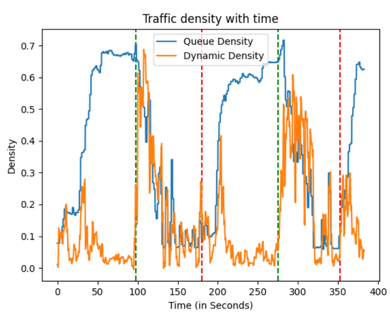
Trade-off Analysis

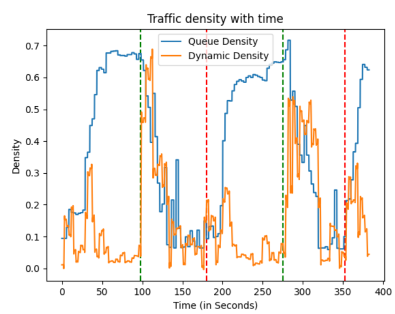
Method 1 : Sub-sampling frames

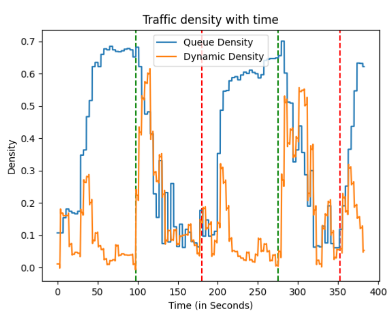
Parameter chosen : No. of Frames dropped

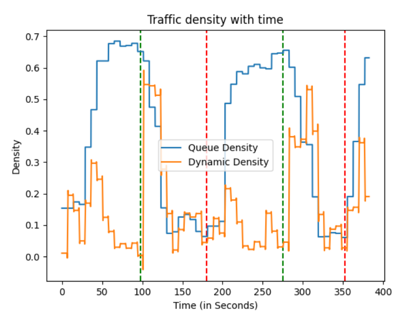






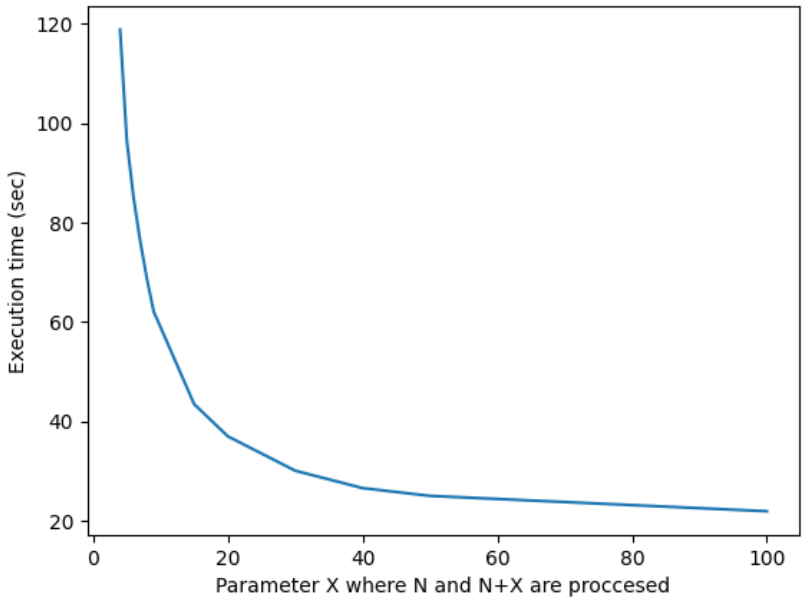






Observations

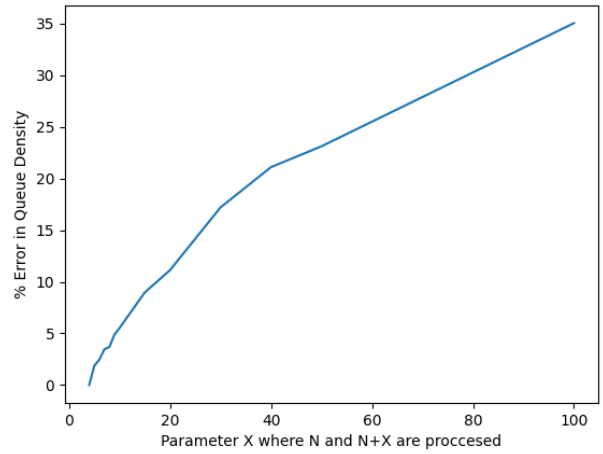
Parameter vs time taken:

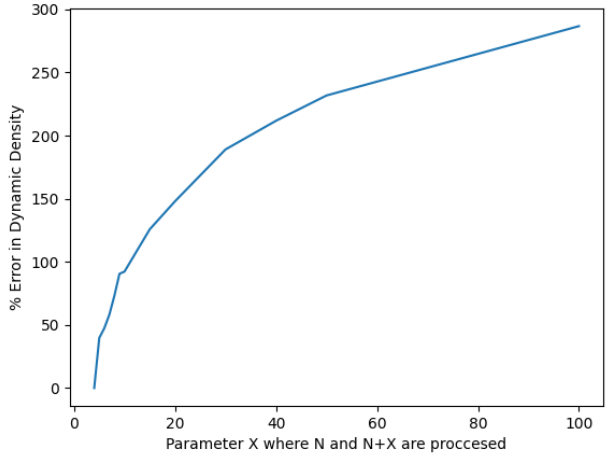


Graph follows hyperbolic nature. The decrease in time is exponential and can be accounted to decrease in processing required as number of frames are skipped and previous values are utilised. There is sudden decrease as X parameter decreases by 1 unit. As 1 unit decrease in parameter causes most extensive 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:



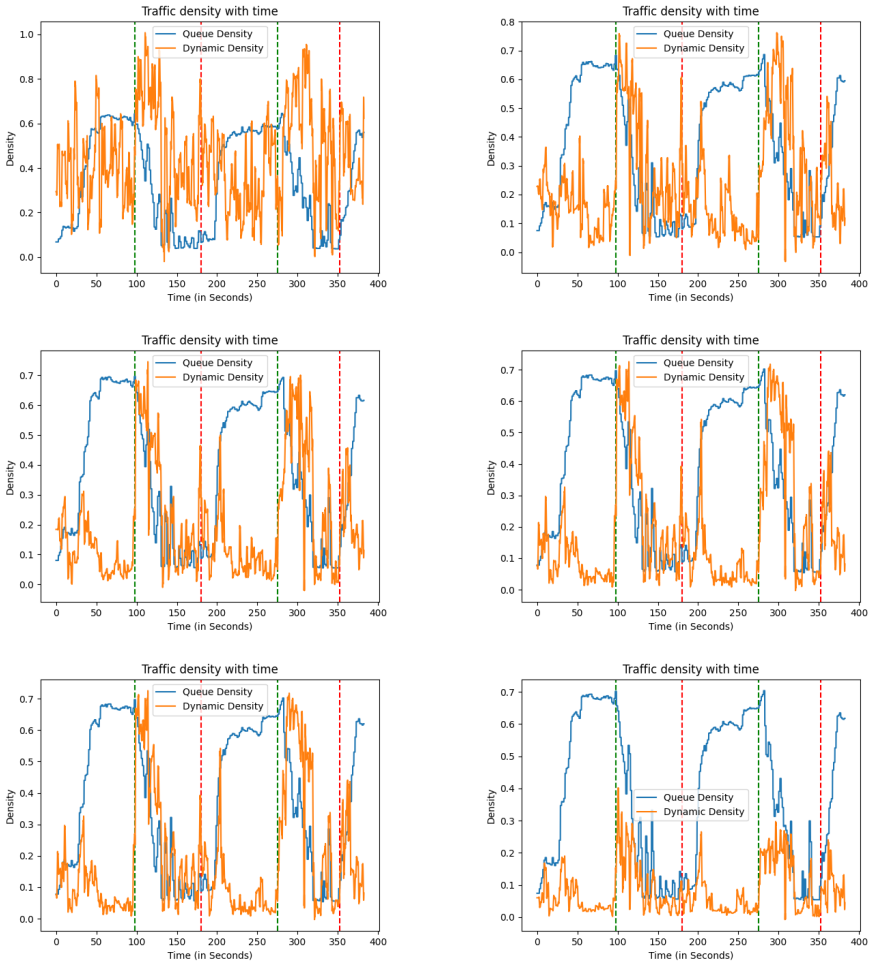


As X increases the error grows suddenly and utility drops. But after a while error start to saturate. This can explained as follows. In the video traffic remains constant for some duration of time and then there is sudden upsurge due to green signal. During that duration it doesn't matter if we skip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

Method 2 : Resolution reduction for each frame

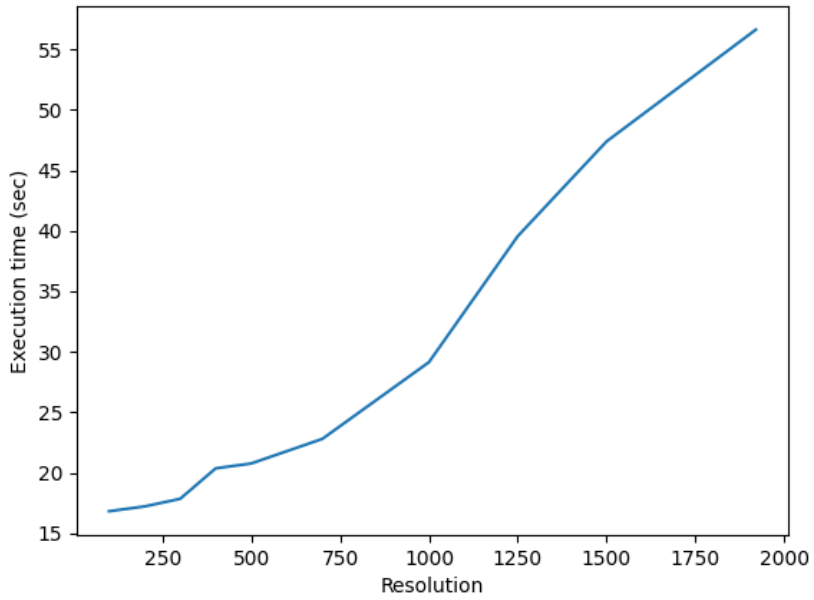
Parameter chosen : Resolution X x Y

Reduced resolution of each frame. The resolution X x Y are taken as parameters for this method.



Observations

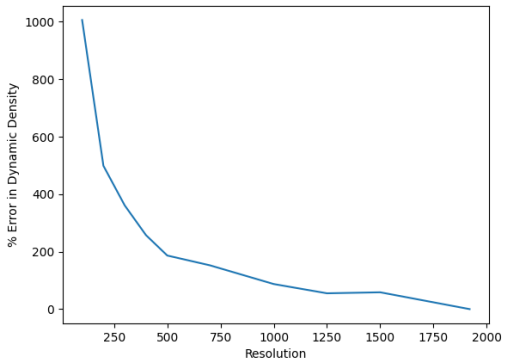
Parameter vs time taken:

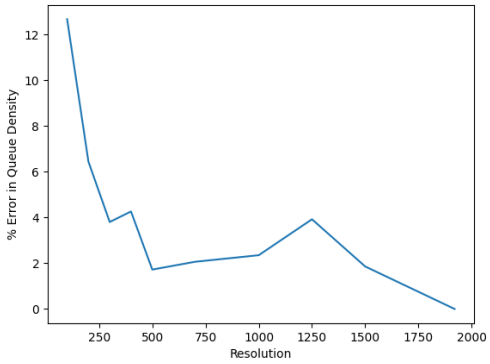


As resolution increase we see increase in time taken. This can be explained as follows. As resolution increases we have higher pixels to process upon.

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

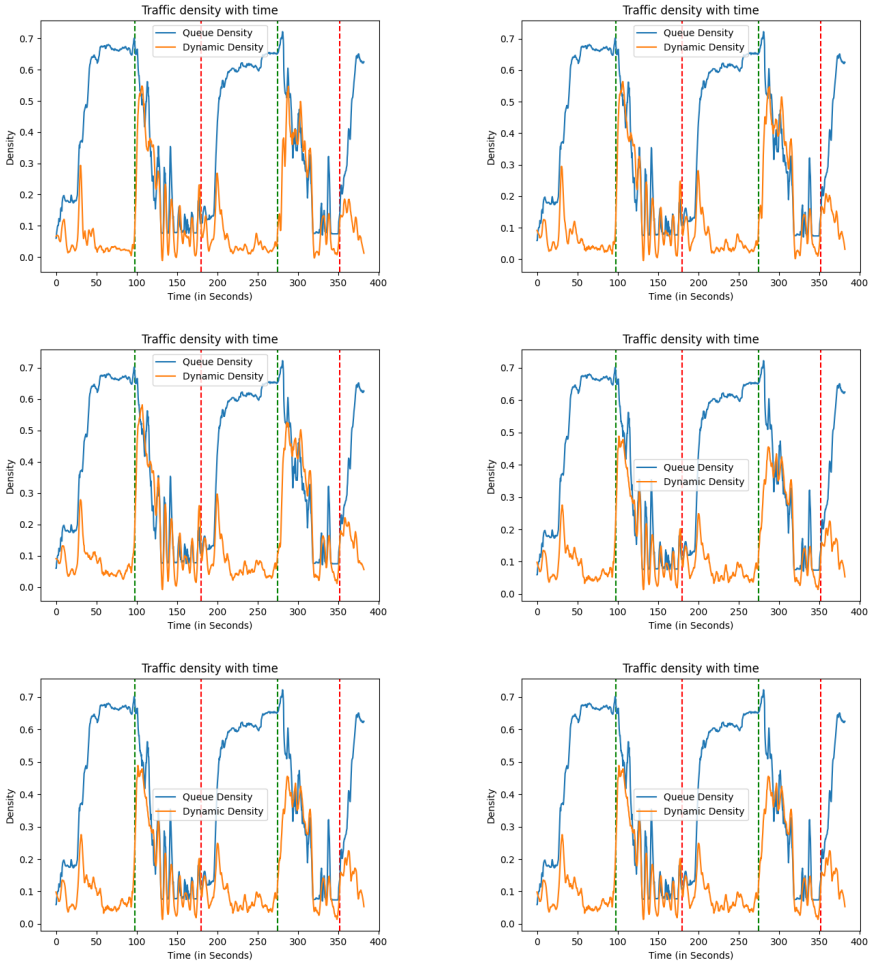
considerably affected the optical flow builds up any error introduced in the start and causes significant deviation till last frame.

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 3 : Frame Splitting

Parameter chosen : No of Splits of frames

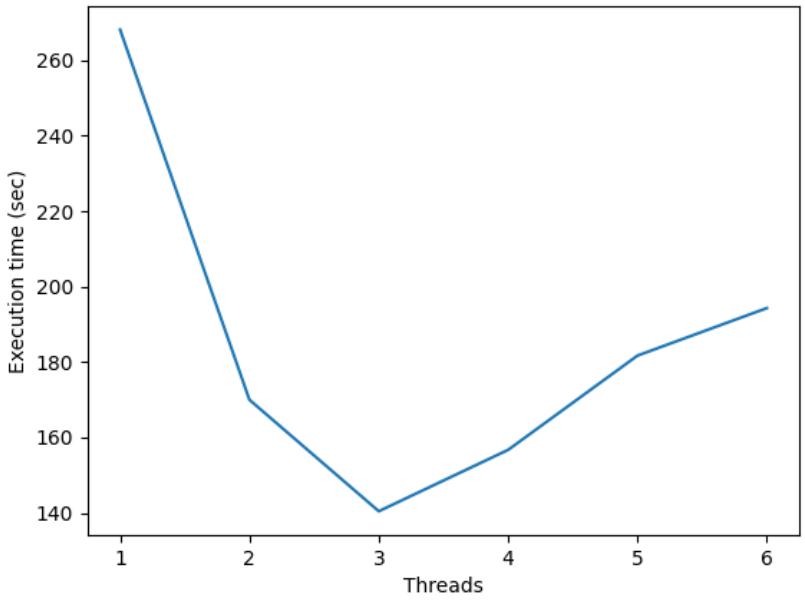
Processing of frames are done by various threads. The number of threads equals no. of splits and each thread is given a single split to process.



Observations

Parameters vs Time Taken

Time takes vs No. of threads

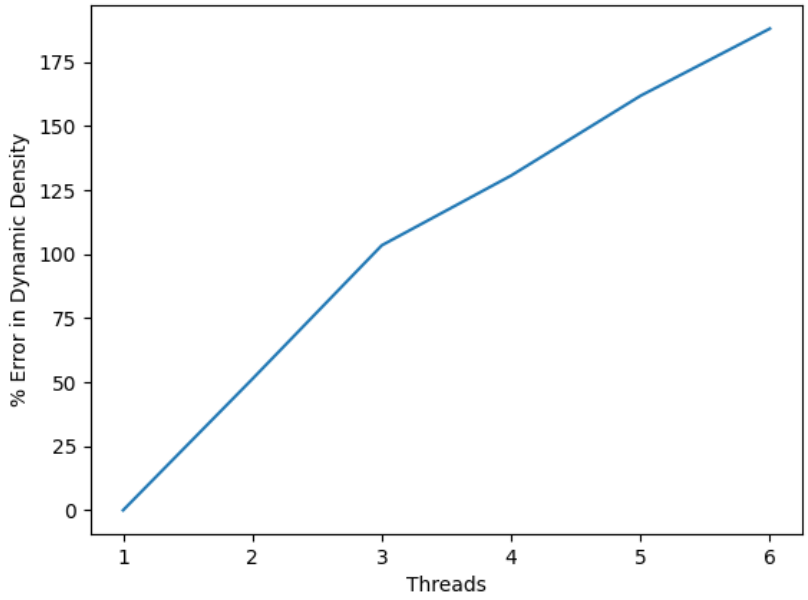


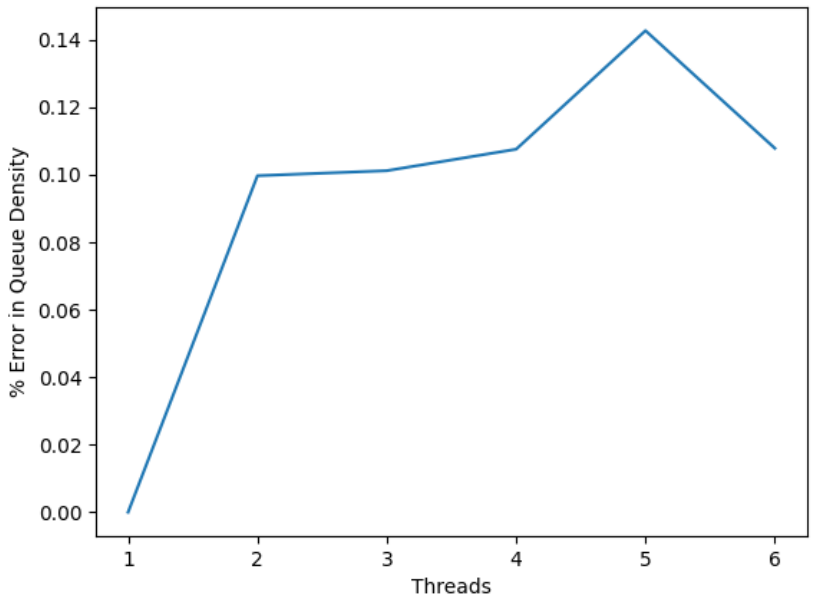
The graph approximately resembles to the curve of upward opening parabola, with minima achieved at No. of threads equals to 3. Starting from No. of threads = 1 (which is the parameter used in our baseline method), the execution time decreases upto 3. This can be explained as follows:

As we increase the number of threads, the section of frame each thread recieves reduces and therefore, the amount of work each thread performs reduces. This results in decrease in execution time up to a certain limit.

But after no. of threads surpasses 3, it is observed that execution time increases, in contrast to expected trend. This behavior is explained explicitly in the upcoming sections.

Parameters vs Utility





We observe an approximately linear increase in the percentage error as the number of threads increases.In general, the error obtained in dynamic density is much more higher than what we obtained in queue density. This can be explained as follows.

When the number of splits (i.e. no. of threads increases), the main cause of error is due to the border pixels which get neglected while processing different sections of frame.As the number of splits increases, the number of boundary pixels increases and thus results in more error.

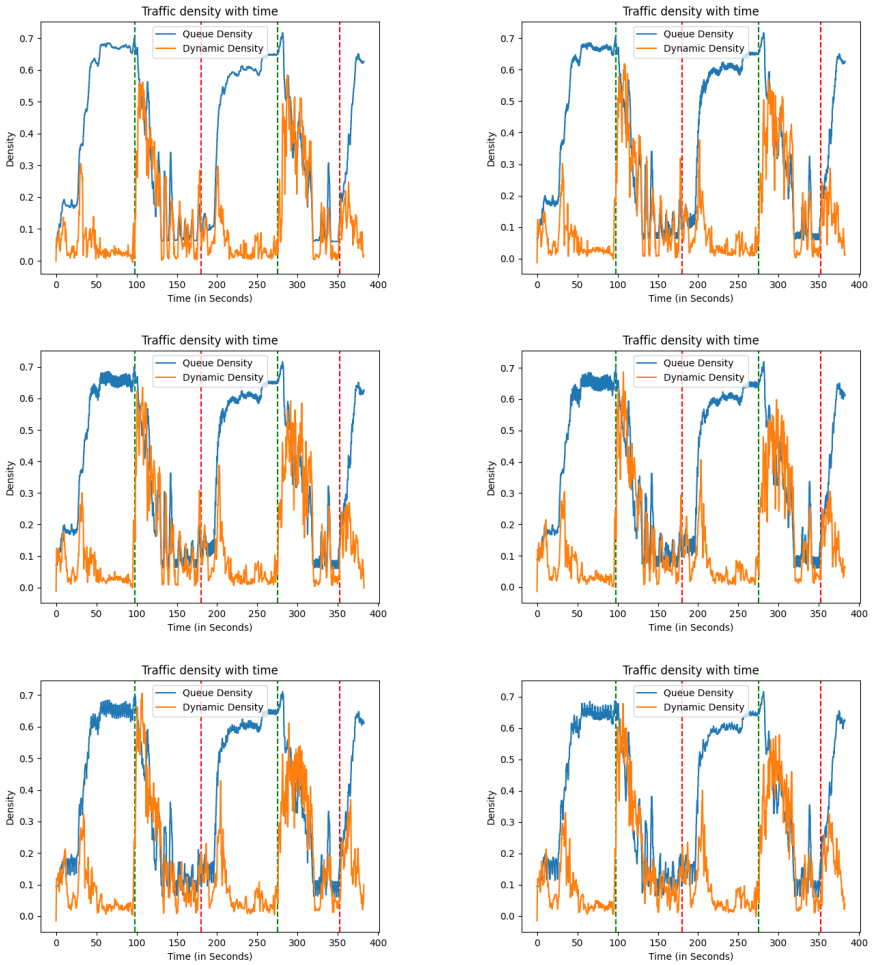
The error obtained in queue density is quite less due to the fact that queue density is based on background subtraction model, so it does not take into account the movement of pixels across different frame sections.

But this is not the case with dynamic density, where we monitor the movement of pixels across consecutive frames (Using Optical flow).So, this causes higher error percentages in dynamic density estimation.

Method 4 : Processing Consecutive frames by different threads

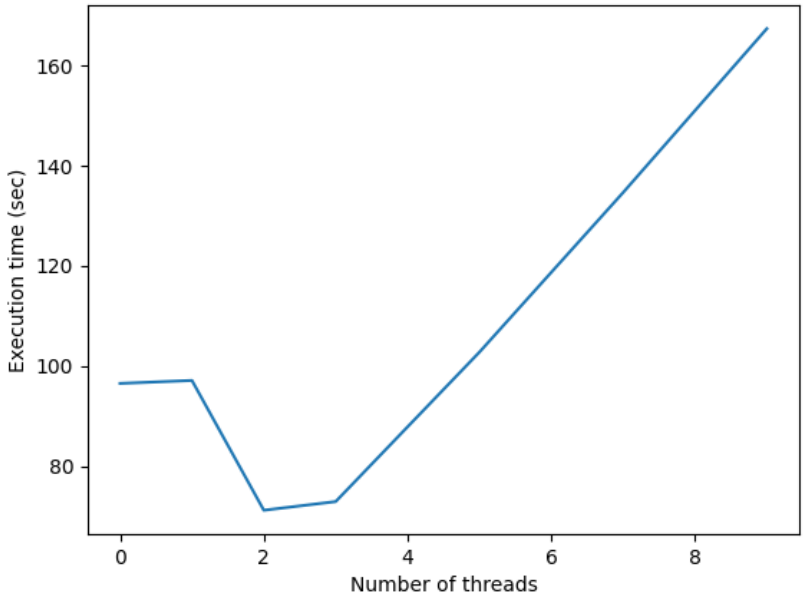
Parameter chosen : Number of threads

Each thread is given consecutive frames for processing.



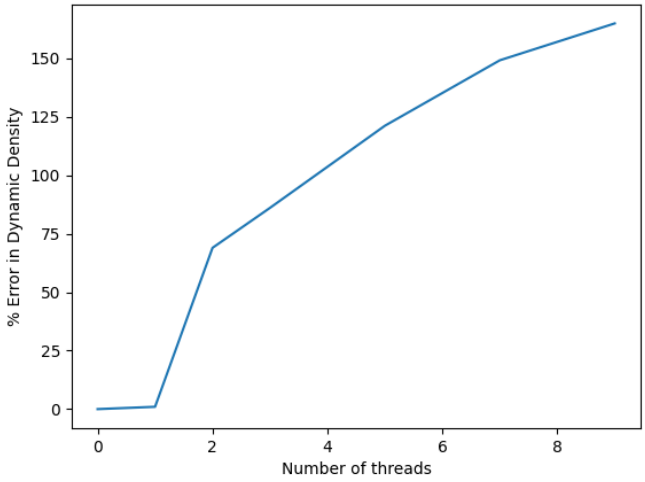
Observations

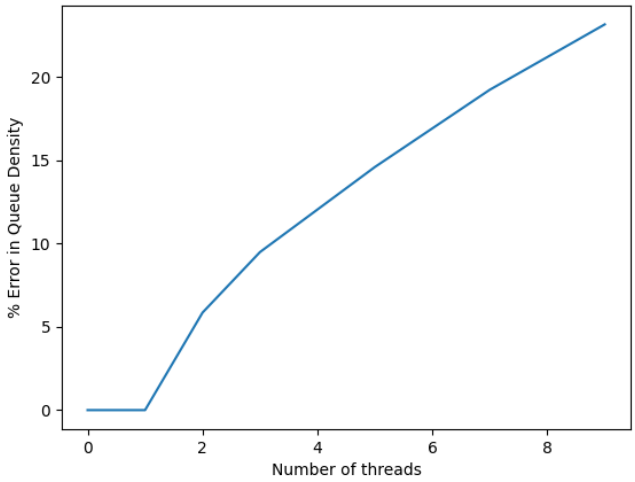
Parameter vs time taken:



As number of threads increase we see initial decrease in the time and then again increase in the time. This trend is similar to what we obtained in method 3 and is explained in the upcoming sections.

Parameter vs Utility:





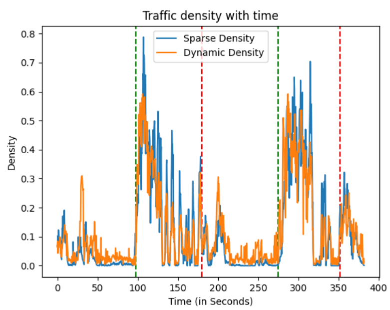
We see negative exponential increase in error as number of threads increase. Also error in dynamic is considerably higher than queue. This can be explained as follows. In this method we are dividing frames across threads. One option was to give certain fraction of video to a thread and certain to other. 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.

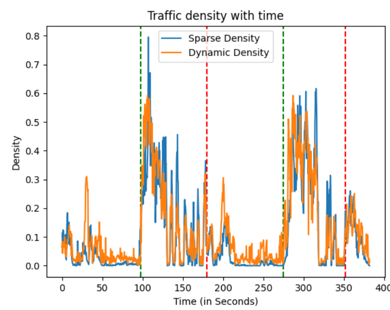
Hence other option that is processing frames according to modulo value of frame number was considered. That is frame number 0, 0+N,0+2*N ... are processed by a thread then 1,1+N,1+2*N... are processed by next where N is total number of threads.

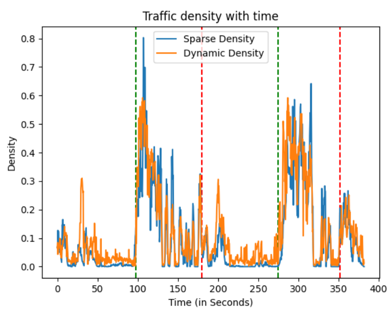
In this case we are forced to calculate optical flow and queue density as per current frame and current frame – N th frame. Which induces error especially at the transition points. Hence even though we have very similar shape we get a lot of noise which is relatively small but is almost at every point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

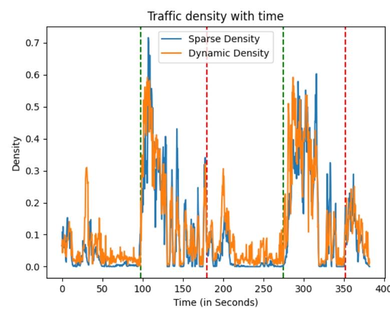
Method 5 : Sparse vs Dynamic Optical flow

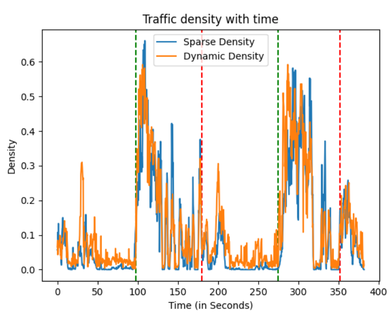
Parameter : BlockSize

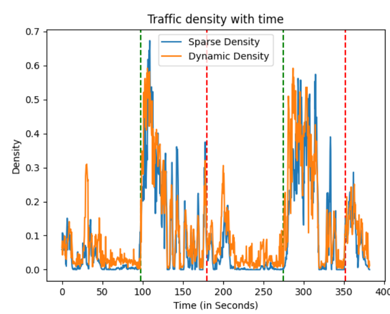






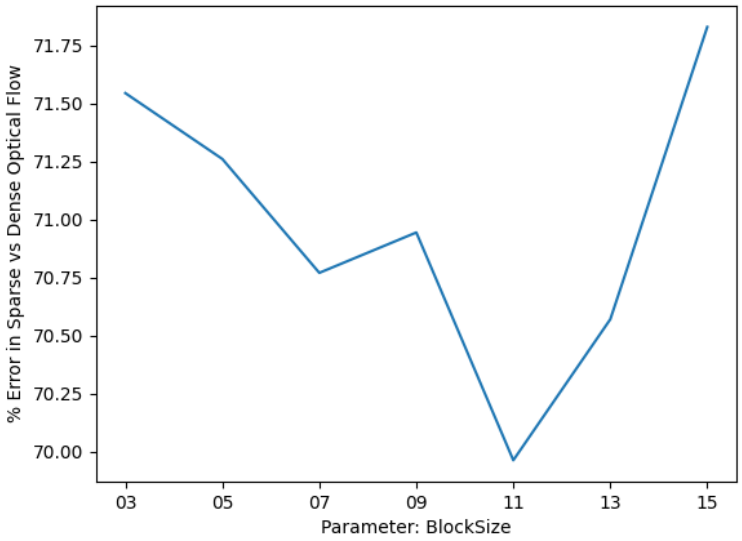




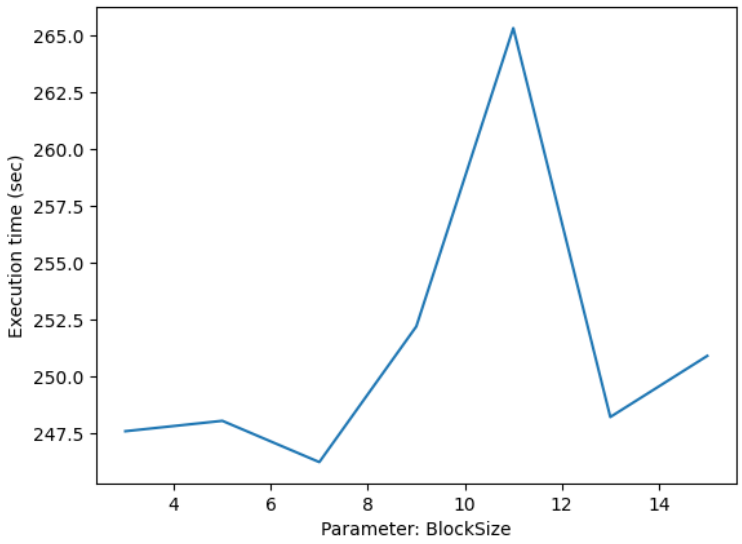


Observations

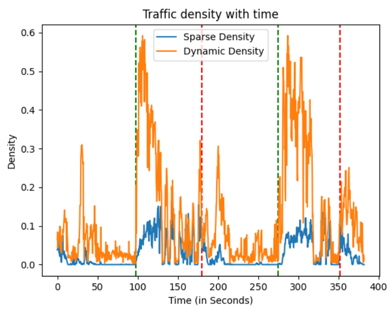
Error in dynamic and Spare OF vs BlockSize

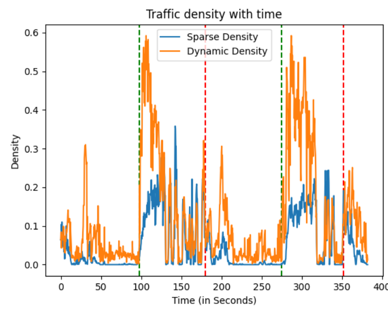


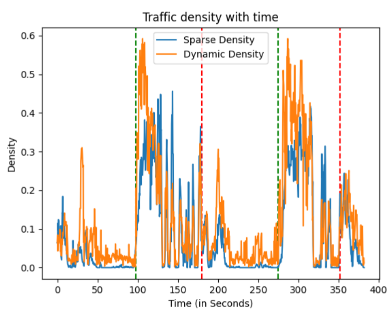
Execution time vs BlockSize

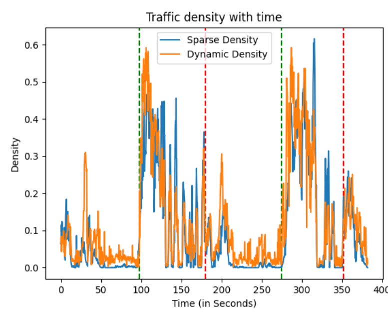


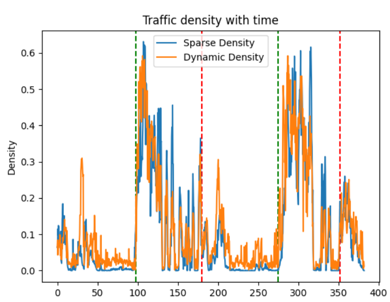
0.1 Parameter : maxCorners

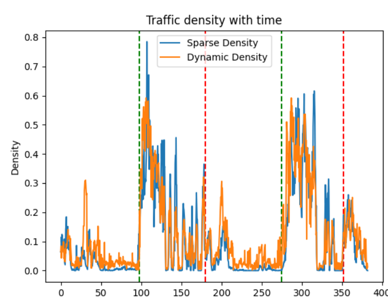






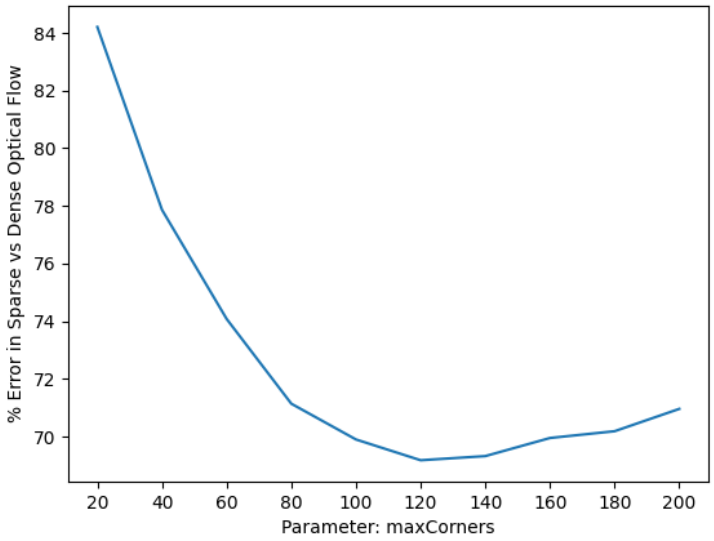




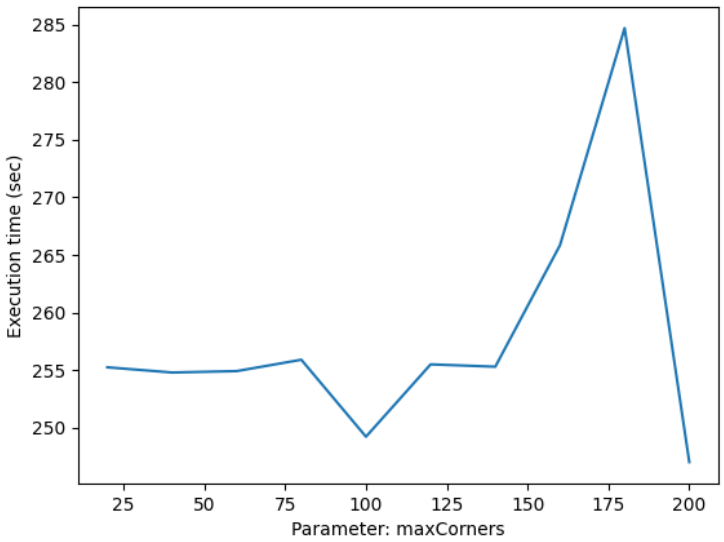


Observations

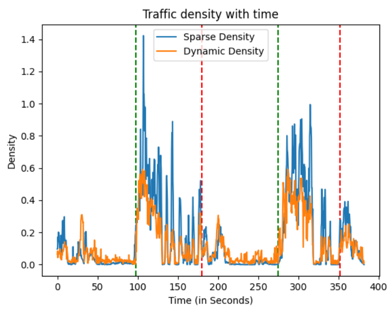
Error in dynamic and Spare OF vs maxCorners

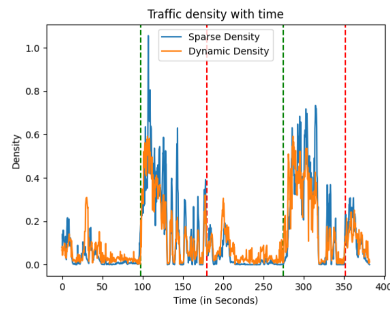


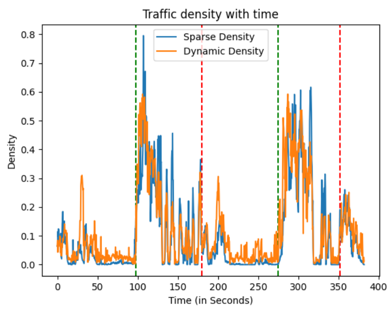
Execution time vs maxCorners

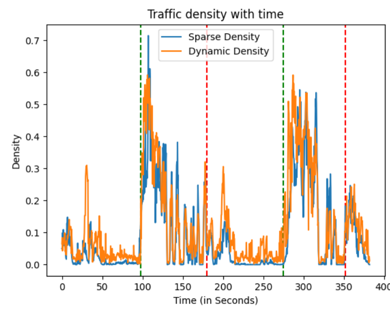


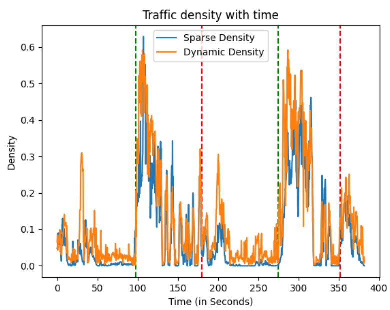
Parameter : minDistance

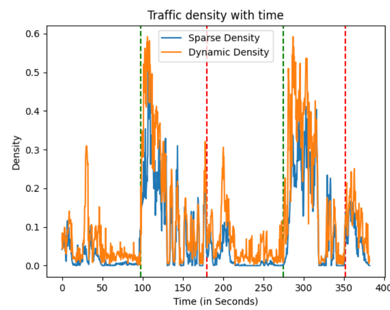






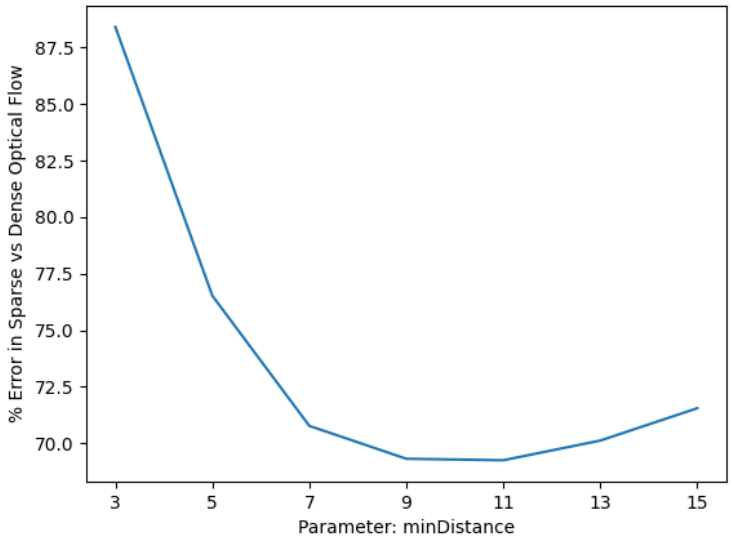




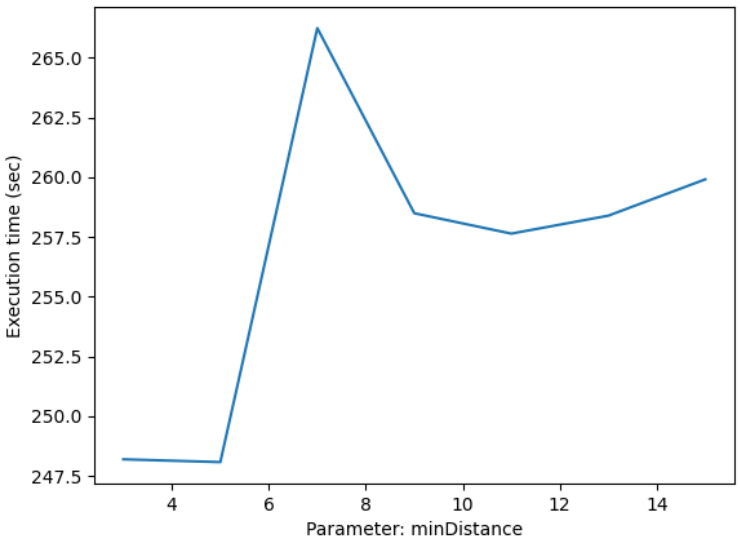


***Observations**

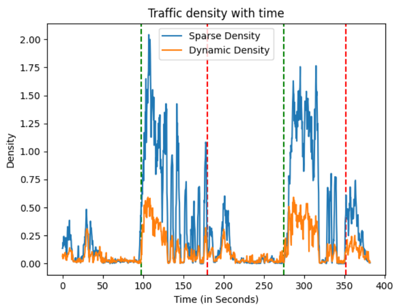
Error in dynamic and Spare OF vs minDistance

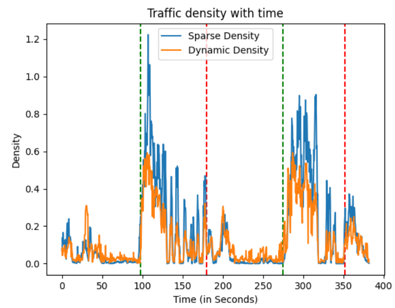


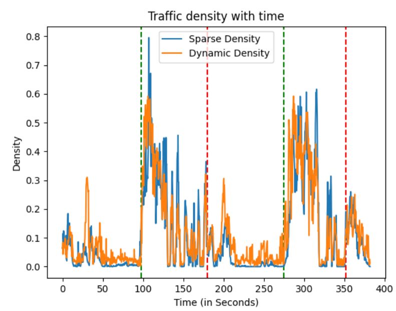
Execution time vs minDistace

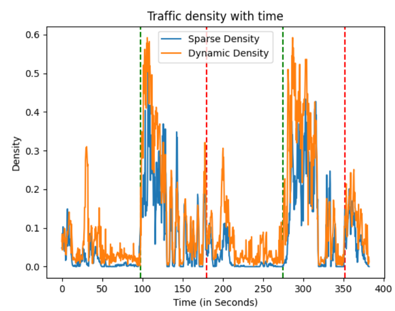


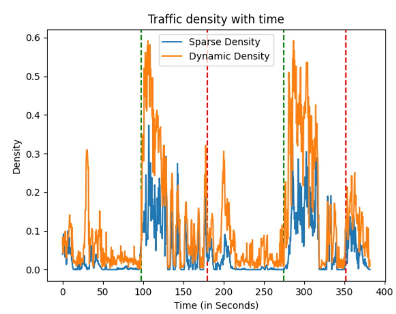
0.2 Parameter : QualityLevel

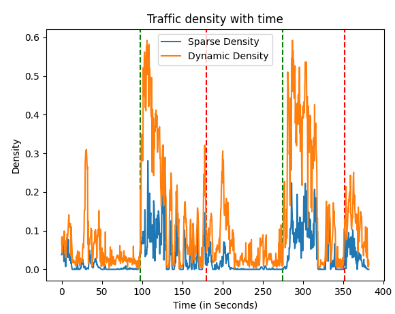






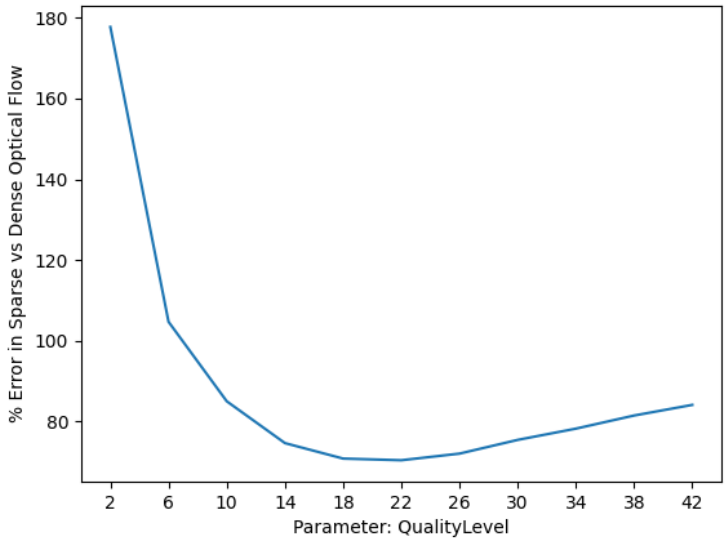




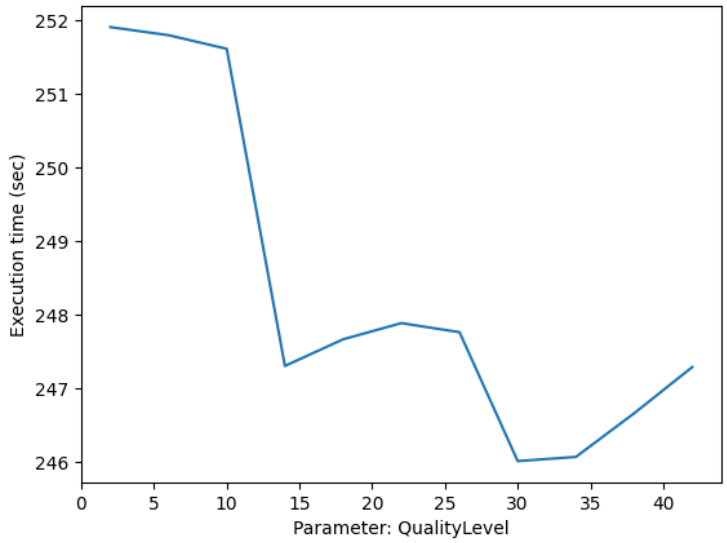


***Observations**

Error in dynamic and Spare OF vs QualityLevel



Execution time vs QualityLevel



Method 5 Trade-off analysis

Parameter vs Time taken
We observe that the trend in Execution time vs Parameters is not regular. For the first three parameters, we observe that the time taken first increases non linearly till and then starts declining from a certain value. While in case of quality level, it roughly decreases across it's ends.

The algorithms selects the points which are less than quality level times the corners of the best corners. As we increase the quality level, more and more points get selected and due to decrease in algorithm complexity, the execution time decreases roughly.

As minimum distance increases, the algorithm selects points which lie at a distance more than the parameter value. The algorithm thus need to analyze less number of points and hence execution time roughly decreases.

Parameter vs Utility

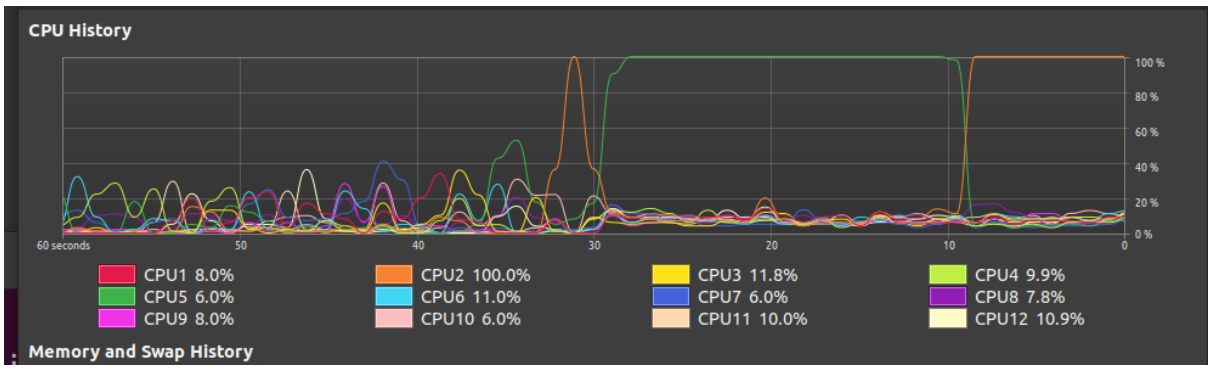
For all the three parameters MinDistance, Qualitylevel, maxCorners, the utility run-time curves resembles hyperbolic shape.The explanation for this trend is beyond the scope of our current theoretical knowledge.

But what we can infer from the curve is that there is a certain value for each parameter which gives the least error. We accommodated these parameters in out final estimation function and observed that the accuracy shoots up significantly.

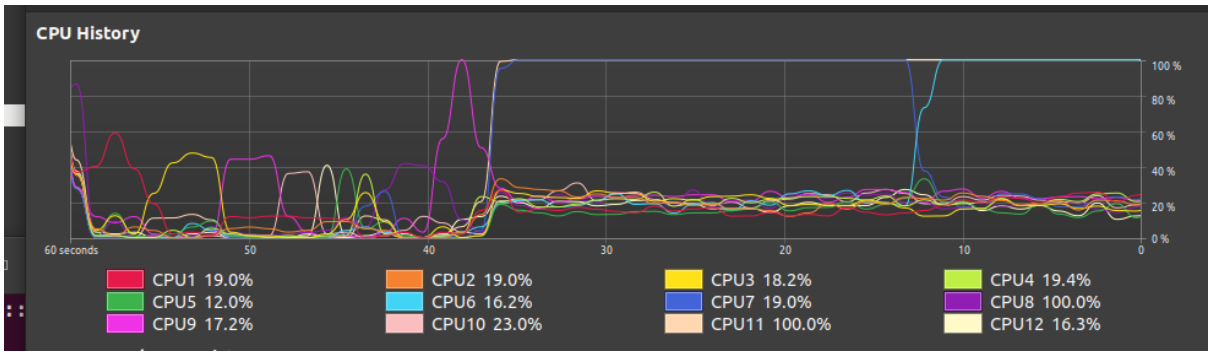
CPU Usage and Multi-threading performance analysis

The curves for CPU utilization with number of threads are as follows:

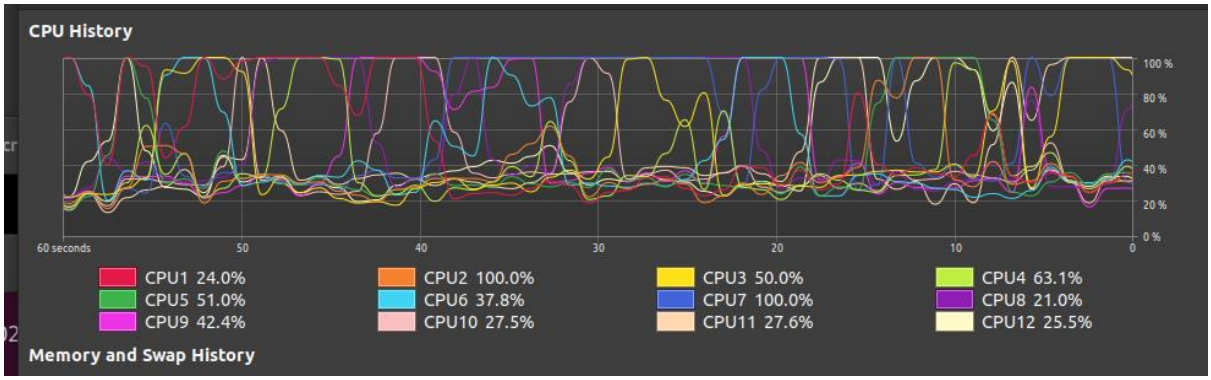
Method-3



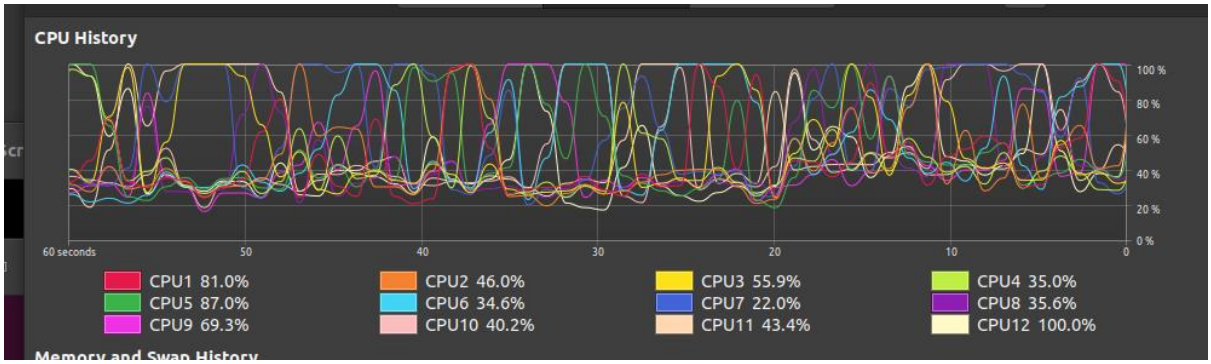
#Threads = 1



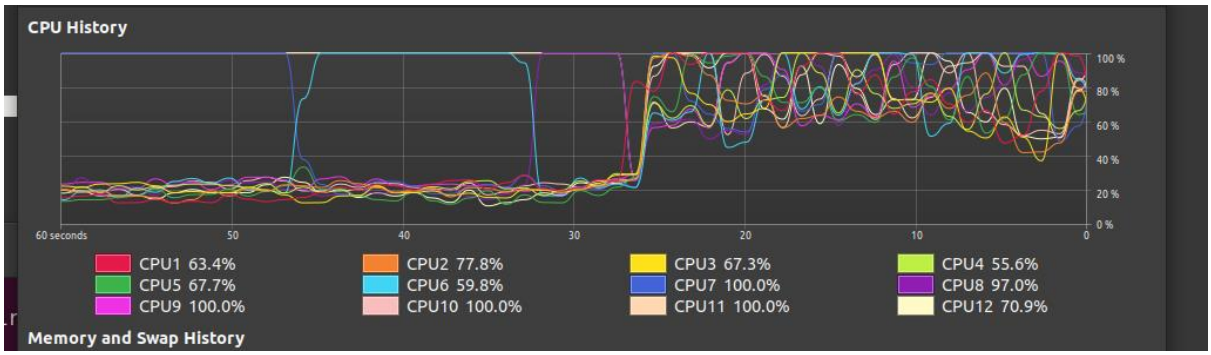
#Threads = 2



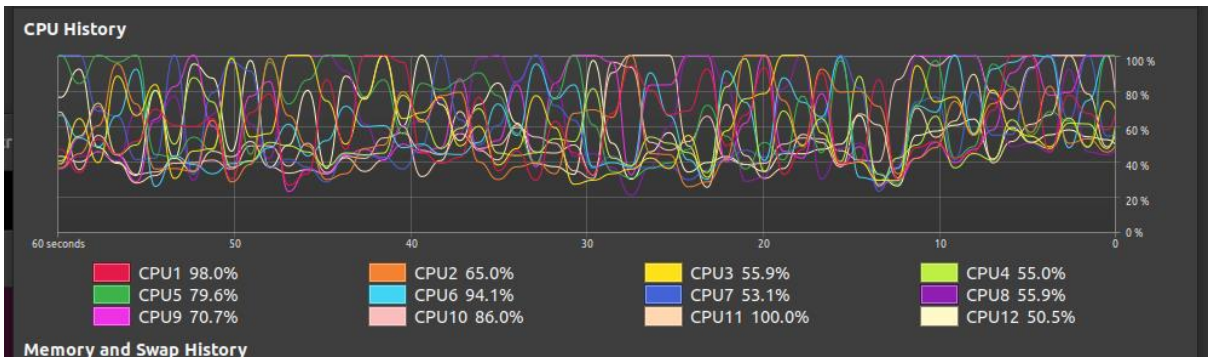
#Threads = 3



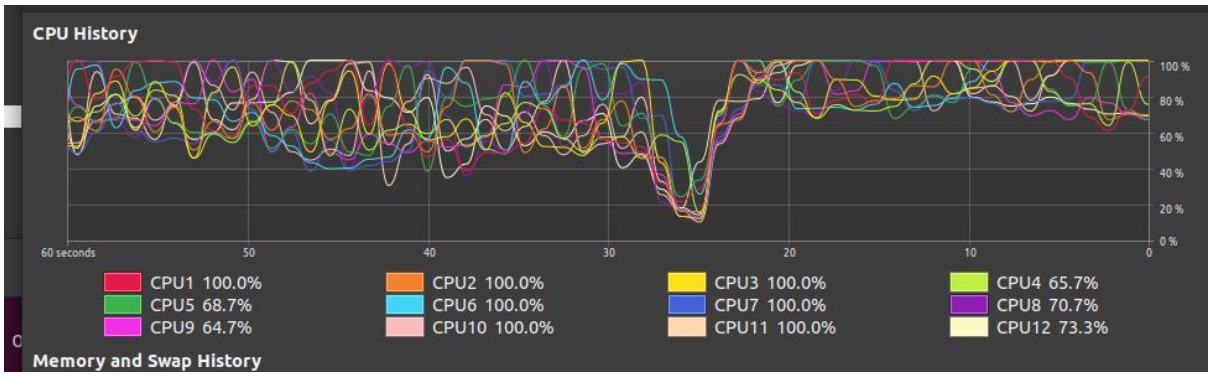
#Threads = 4



#Threads = 5

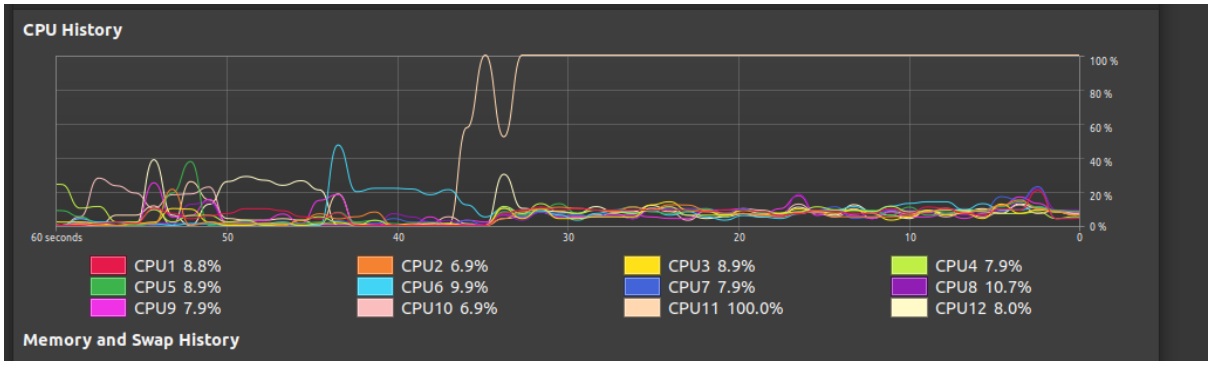


#Threads = 6

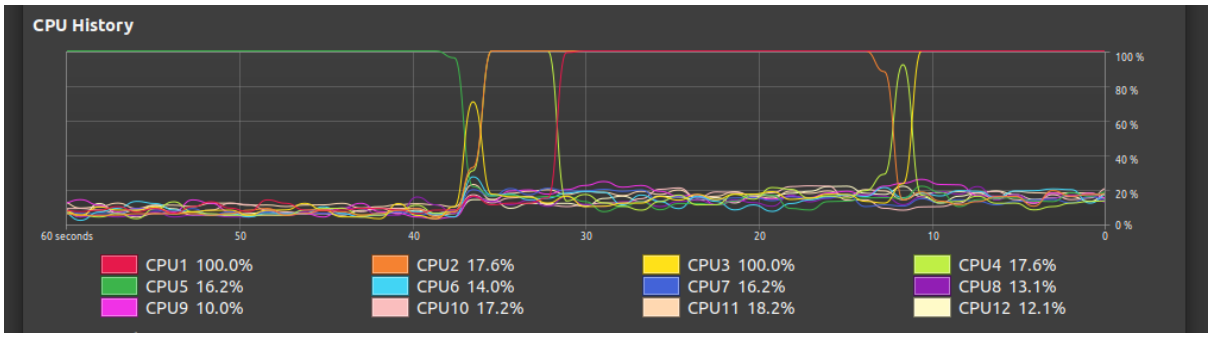


#Threads = 7

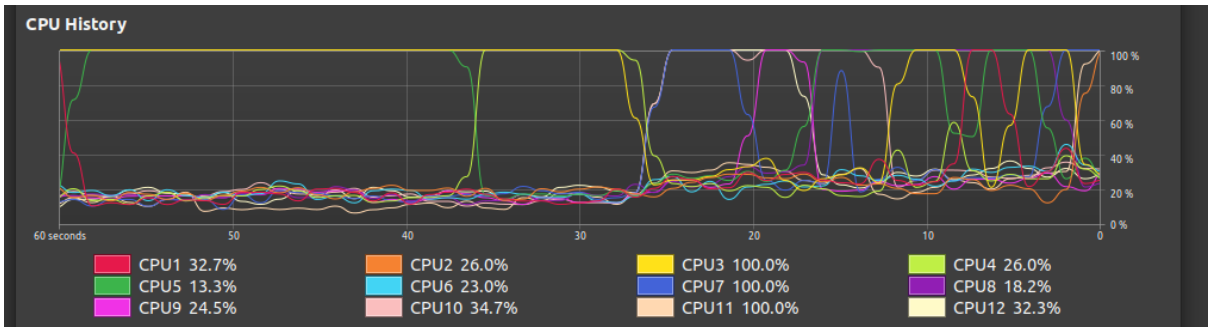
Method-4



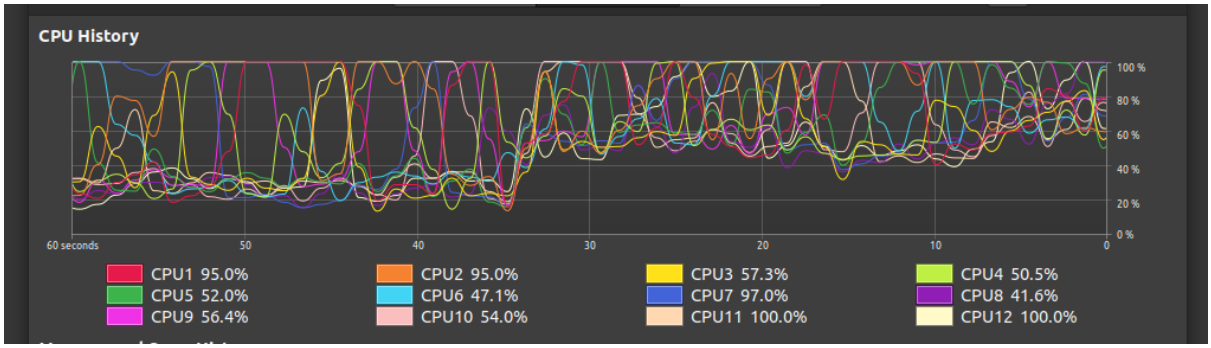
#Threads = 1



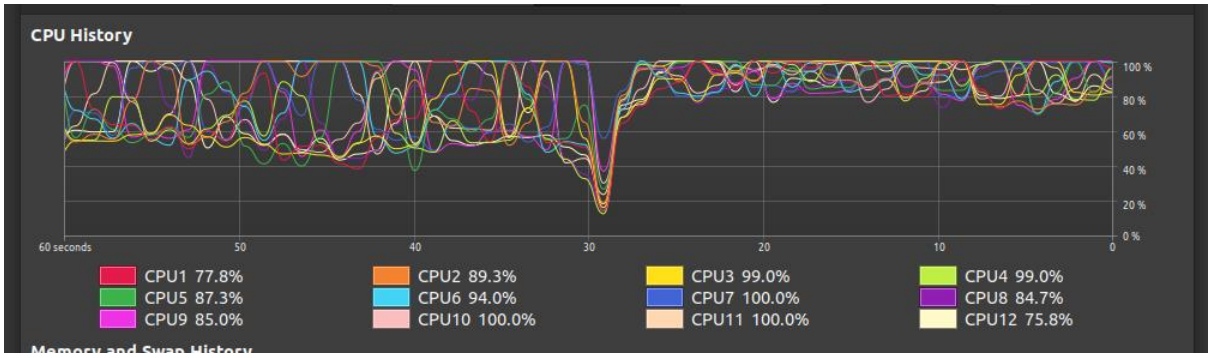
#Threads = 2



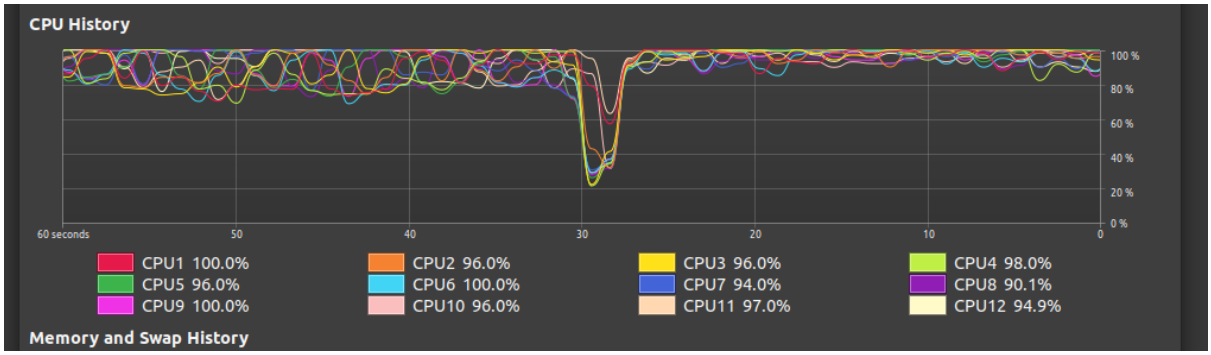
#Threads = 3



#Threads = 5



#Threads = 7



#Threads = 9

Analysis:

Multithreaded code has large number of pitfalls. They tend to reduce the effective size of cache, since they are saving recently used data and code for multiple threads.

CPU usage vs Runtime trade-off

For both methods 3 and 4 we see steady increase in CPU usage 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 across 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.

Parameter vs Execution time analysis for methods 3 and 4

As number of threads increases, we see initial decrease in the execution time and then again increase in the time after number of threads surpasses 3. This can be explained as follows.

Threads usually slows down the execution of programs when a sequential algorithm is refactored into a series of threads. If the threads execute on same core, then the slow down is primarily caused by context switching between the threads.

As number of threads increases, we have two process going on. As thread number increases, 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.

With increase in no. of threads, the overhead of initiating and terminating threads itself slows down the CPU and disguises the actual work, and another overhead which costs the CPU is from the way the multiple threads share the finite hardware resources.

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.

Conclusion:

After the trade-off analysis done by all the above methods, we infer that splitting frames spatially and consecutively across various threads (Methods 3 and 4) does not give much of a difference in the utility metrics, but show a significant optimization in the latency Metric.

While rest of the methods do not optimize the execution time, yet they fit better when it comes to optimizing the utility metric. The utility metric remains more or less constant for methods 3 and 4, while changes significantly for the rest of the methods, giving most optimized performance at the points of minima.