Analysis

CS5190426, CS1190350

March 2021

1 Introduction

This task's aim is to reduce the runtime for the program used in subtask 2. The task describes about utility, runtime trade off using four methods as discussed in the following sections.

2 Metrics

We defined our utility metric as utility percentage when compared to baseline.
So utility percentage is 100 – error percentage .
Where n is no of frames loaded in baseline i.e total frame count/5
Runtime is time for which our code run.
error percentage= square root of (summation of ((qd for parameter-qd of baseline/qd of baseline)^2)/n).

3 Methods

3.1 Method 1 (Sub Sampling frames)

Basically in this method we drop x frames ,if we drop frames from N to N+X when dropping these frames all intermediate frames queue density will be same as Nth frame.
Since in our baseline we dropped 5 frames and calculated queue density.
So in my error calculation I will calculate error at every interval of 5 frames and get RMS value of error for entire video.

Parameter X: I varied my X from 4 to 15 i.e i calculated utility and runtime for dropping 4 frames to dropping 15 frames.

3.2 Method 2 (Reducing resolution)

Basically in this method we decrease resolution for each frame of the video and calculate queue density.

Our baseline queue density resolution is 400 X 800.
So in error calculation we will calculate error for this reduced frame parameter XxY with baseline.

Parameters XxY: 200x400 , 250 x 500 , 300 x 600 , 350x700 , 400x800 , 450x900 , 500 x1000

3.3 Method 3 (Frame split)

In this method we split the whole frame into pieces. We have considered spliting it vertically only, to make the code simple. Each of these nearly equal sized frames are operated by individual threads to calculate the densities. The integer values of densities are passed to the main thread to add, write into the file. The fact that all threads have completed before writing is ensured by the conditional statement using global boolean variables.

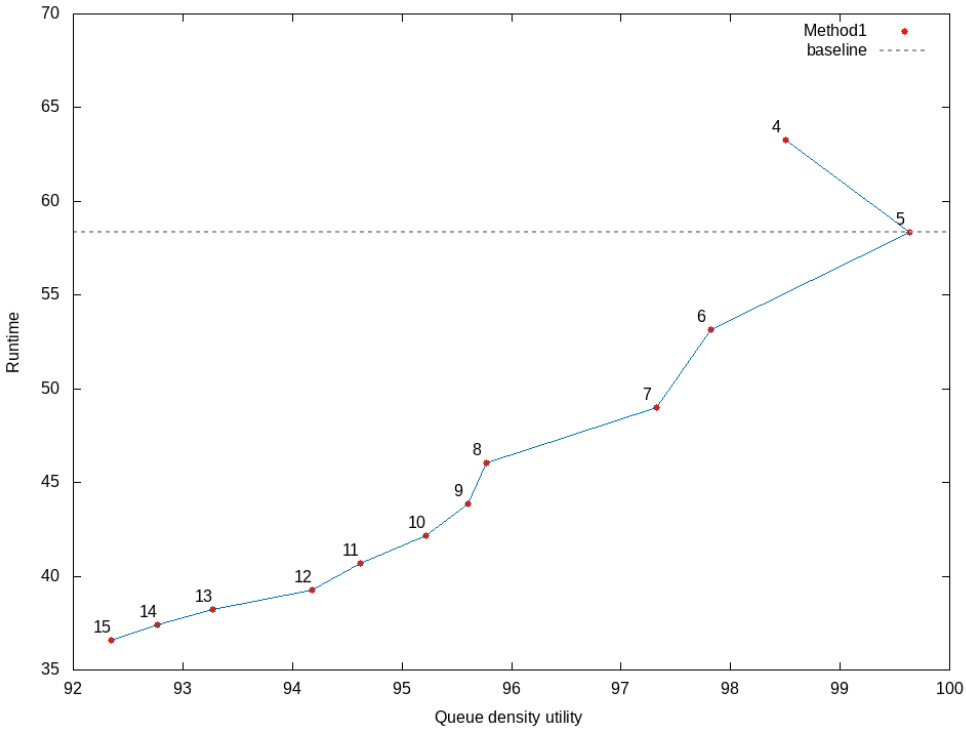
3.4 Method 4 (Consecutive Frame assignment)

In this method we operate on consecutive frames in different threads i.e., first frame is given to one thread, next is given to another thread and a cycle of assignment is formed according to the number of threads available. The assignment is done in main thread to global variables from which the threads can access the frame corresponding to their IDs. The densities are passed to main thread using global variables for writing into file. We write the densities generated by the thread, just before next assignment of frame to the thread, assuming its complete.

4 Trade-off analysis

4.1 Method 1

4.1.1 Runtime vs Utility graph for method 1



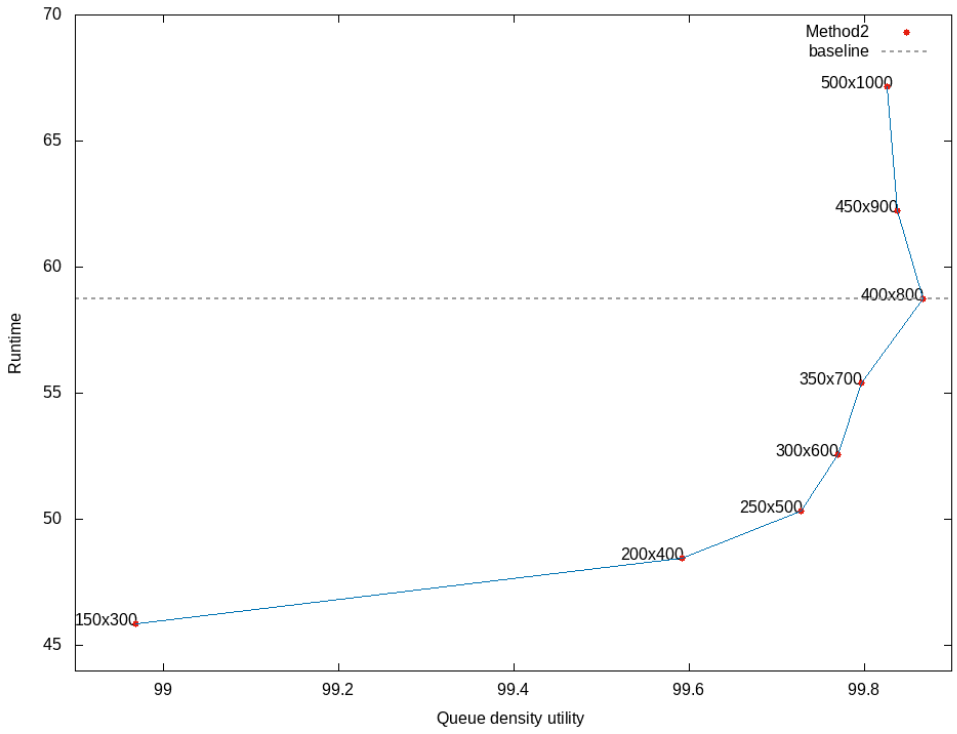
4.1.2 Inferences specific to method 1

Points to be noted:
1.Since we are comparing utility w.r.t baseline, Our baseline will have highest utility.
2.If we load less frames per second it should take more time to run (since it should run more frames to complete execution)
3.If we load more frames per second it should take less time to run (since it will run less frames to complete execution)
From graph we can observe that our above points are strictly followed.
From graph,if we observe utility,by dropping more frames utility is also decreasing its because if we drop frames we will get different queue density values w.r.t for suppose if we drop 7 frames our 2nd addition in error calculation uses value of queue density of frame 8 - queue density for base line of frame 11 and if we drop 11 frames our 2nd addition in error calculation uses value of queue density

of frame 1 - queue density for base line of frame 11,So we are calculating error wrt more long back frame our error will be more for later.

4.2 Method 2

4.2.1 Runtime vs Utility graph for method 2

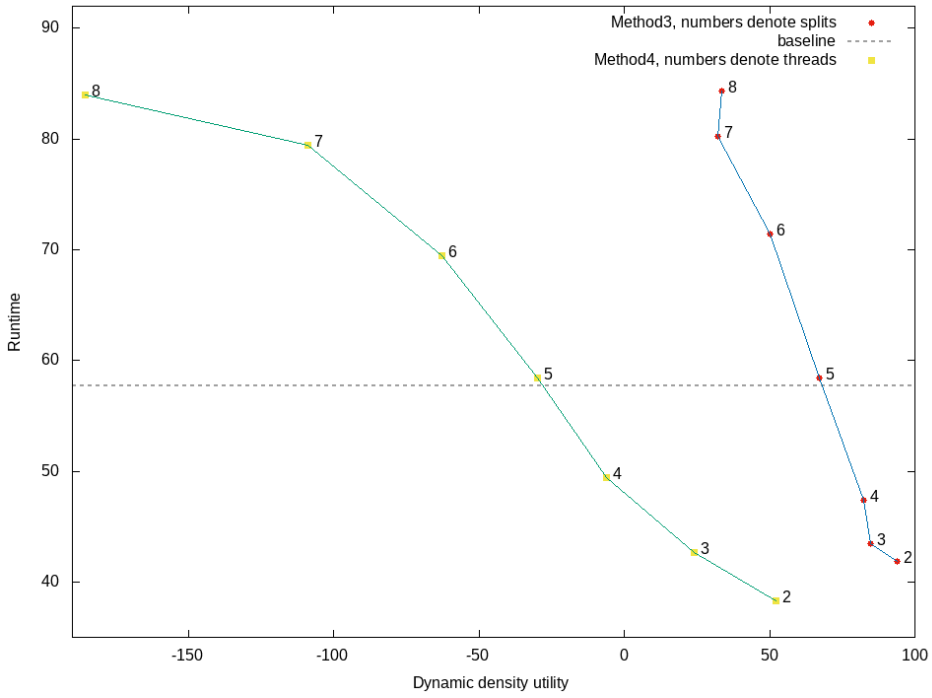


4.2.2 Inferences specific to method 2

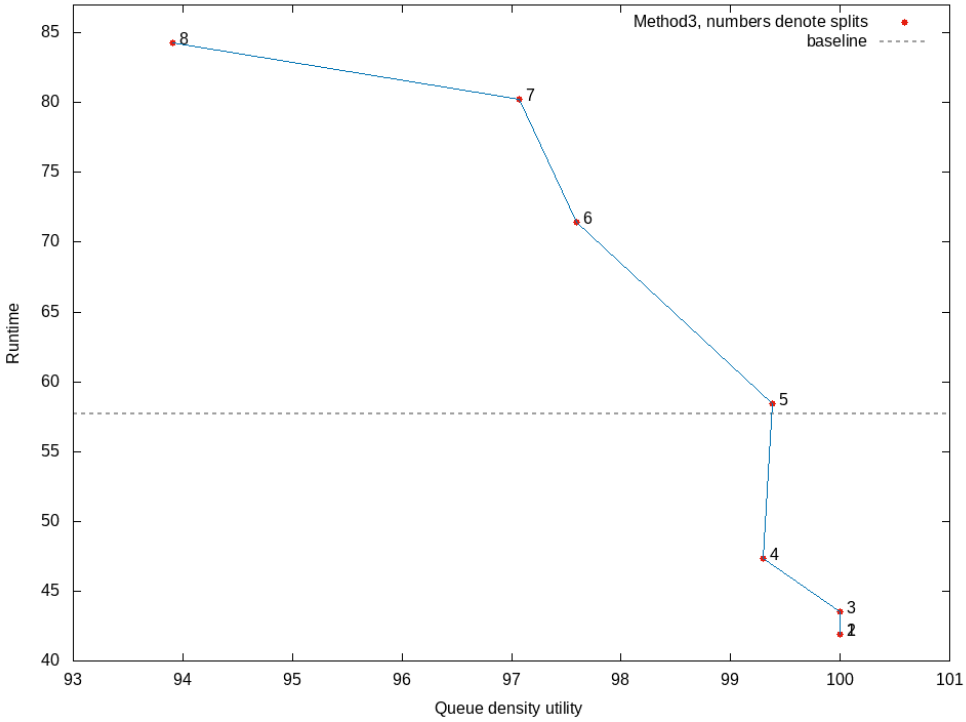
Points to be noted:
1.Since we are comparing utility w.r.t baseline, Our baseline will have highest utility.
2.Since if we decrease resolution it takes less time to traverse each pixel in each frame.
3.Since if I increase resolution it takes more time to traverse each pixel in each frame
From graph we can observe that our above points are strictly followed.
From graph,if we observe utility,by reducing resolution utility is also decreasing this is because accuracy of reading a white pixel for our queue density at that point will also decrease so ultimately error increases and then utility decreases.

4.3 Inferences for multi-threading

Uniform distribution of tasks among different threads is vital for obtaining reasonable improvement in runtime and accuracy.
The runtime decreases drastically on switching from single thread (main of subtask 2) to three threads (two threads calculating densities, main reading, writing) in both methods.
The runtime increases, utility decreases with increase in number of threads in both methods.
The runtime is less than original when using 2 to 4 extra threads then it increases very much for more threads. Also error increases very much for more than 5 threads. Indicating that the computer on which the code was run has 4 cores.
Using 5 extra threads is closest to baseline and utility is also more for that with respect neighbouring points.
Dynamic density utility is very good for method 3 when compared to method 4.
For 2 extra threads, dynamic density utility of method 3 is 93.7 and of method 4 is 52.2. Since, we are skipping frames in method 4, dynamic density is calculated on consecutive frames.
The values in the graphs are obtained by taking average over 4-6 runs for each parameter. We have also plotted the point denoting only one extra thread in both graphs.



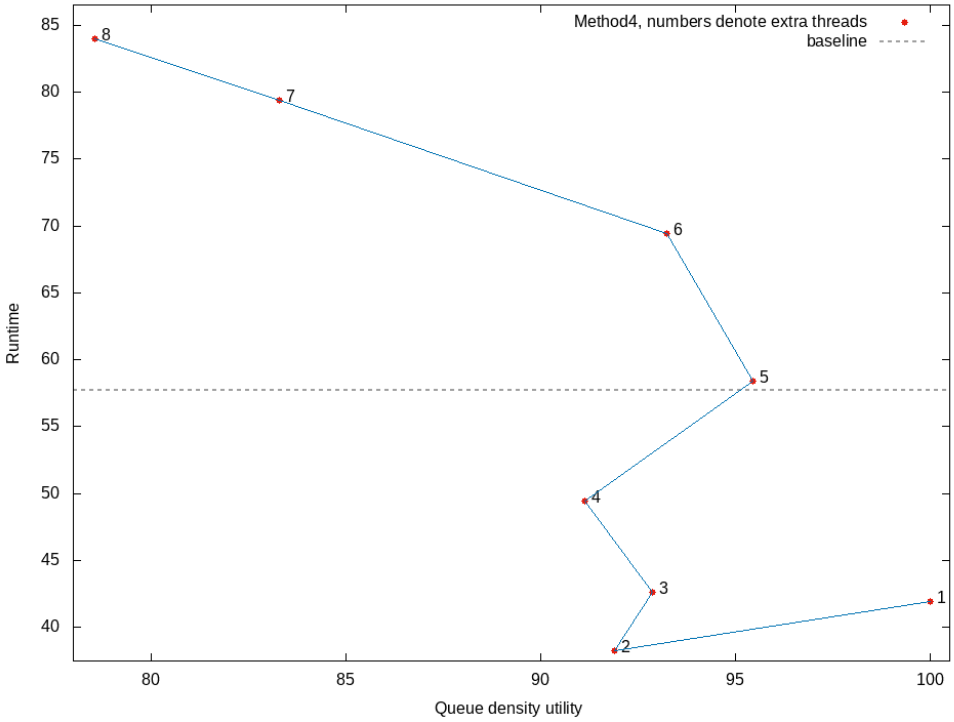
4.3.1 Runtime vs utility graph for method3



4.3.2 Inference specific to method 3

The queue density utility is 100 for 2, 3 splits, for more it steadily decreases.
The dynamic density utility is good for 2-5 splits, for more it decreases.
This method has better utility than method4 but it takes slightly more time for equal number of threads.

4.3.3 Runtime vs utility graph for method4



4.3.4 Inference specific to method 4

This method has the fastest runtime for 2 extra threads. The utility decreases drastically on increasing number of threads. Accuracy does not seem to show a strict bound for this method indicating the frames assigned are not exactly corresponding to frames assigned in the case of baseline. This leads to accumulation of errors.

4.4 Problems faced in multithreading

Unable to imshow.
Hard to know the source of error.
Managing the shared resources is difficult.