COP290: Assignment - 1(3)

Himanshu Gaurav Singh
2019CS10358

Abhinav Jain
2019CS10322

March 31, 2021

Trade-off analysis for software design choices

Abstract

- We analyse the runtime vs utility trade-off between various implementation approaches to the problem of determining queue and dynamic density on a road using *OpenCV* library functions.
- The baseline for the analysis is the output of the queue and dynamic density functions submitted in Assignment-1 Subtask 2.
- The parameters over which the analysis has been done are :
  - Number of frames skipped.
  - Resolution of the image.
  - Spatial multi-threading.
  - Temporal multi-threading.

Metrics

- The utility metric is the root-mean-squared-error from the baseline output(distance from baseline output in the L-2 norm). More the error, less the utility.
- The runtime metric is the time taken to run the whole program on my laptop(a quad-core MacOS machine).
- For the analysis of concurrent methods in Part-3 and Part-4, CPU utilisation is also used as a metric.

Method-1 : Sub-Sampling Frames

Methodology

The baseline code sub-samples one among every three frames. The skipping count (say x) was varied over 3,6,9,12,15,18,21 to get the outputs and then compared with the baseline results.

Results

The following plot describes the obtained results.

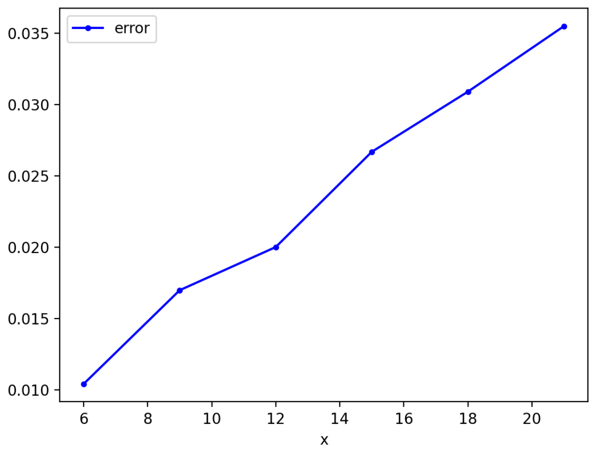

Figure 1: Error v/s x

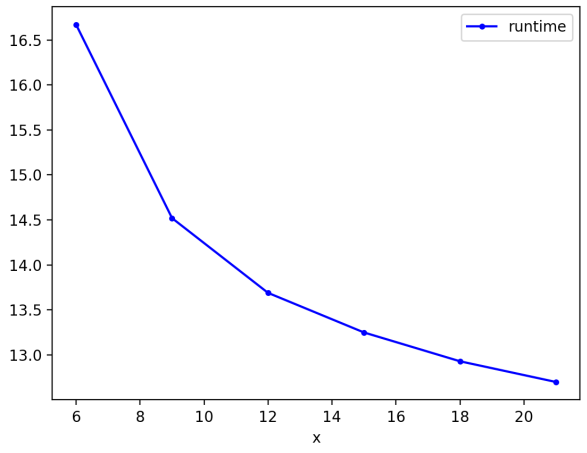

Figure 2: Runtime v/s x

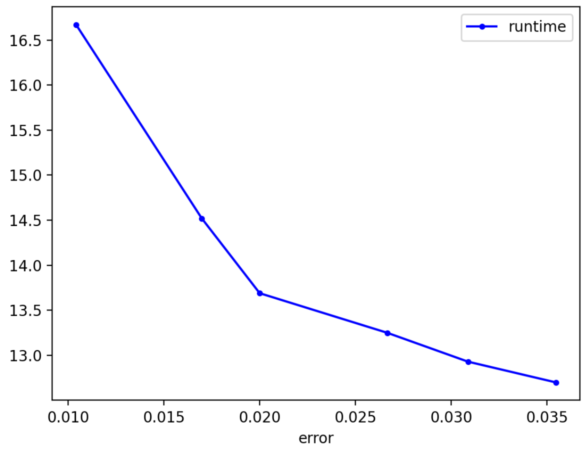

Figure 3: Error v/s Runtime

Observation & Explanation

An increase in the error and a decrease in runtime is observed on increasing x. The runtime decreases because less number of frames are processed as the skipping count increases. The error increases because we attempt to approximate the benchmark with a set of lesser number of frames as we increase x.

Method-2 : Modification of frame resolution

Methodology

The resolution of the frames are decreased as soon as they are captured before processing them. It is made sure that the row-column dimension ratio of the modified frame are same as that of the original frame.

Results

The following plot describes the obtained results.

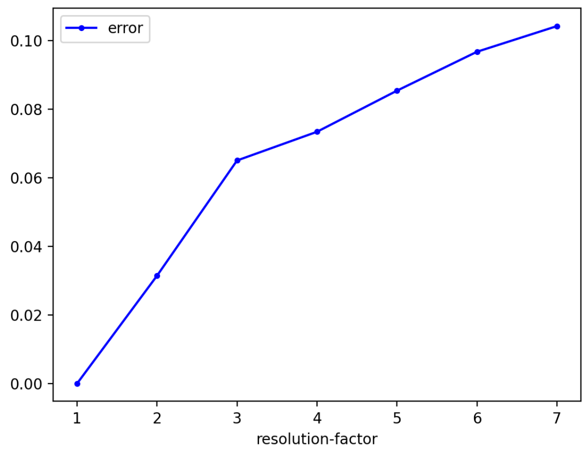

Figure 4: Error v/s Resolution-factor

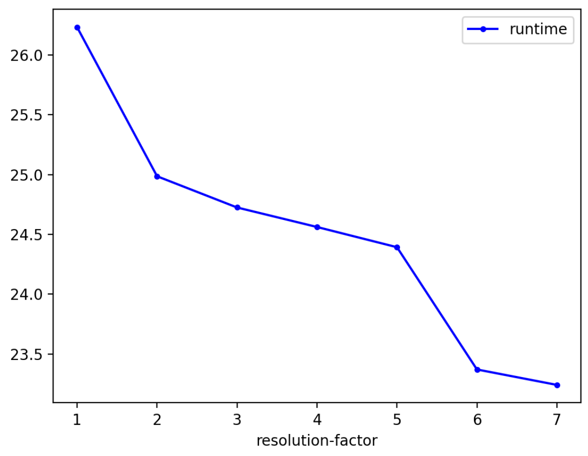

Figure 5: Runtime v/s Resolution-factor

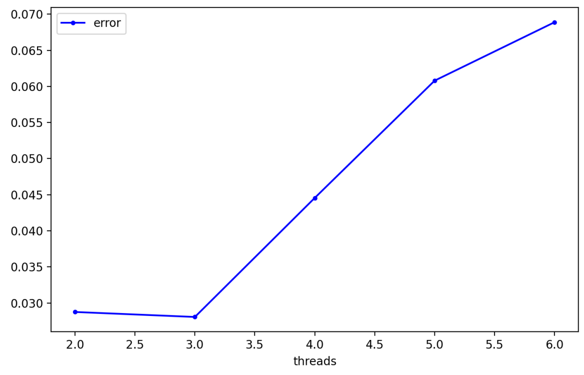

Figure 6: Error v/s Runtime

Observation & Explanation

An increase in the error and a decrease in runtime is observed on decreasing resolution. The runtime decreases because we process a smaller image matrix per frame as compared to the benchmark. The error increases because we attempt to approximate every frame with a smaller, averaged-out matrix as we decrease resolution.

Method-3 : Spatial Multi-threading

Methodology

- After capturing, each frame is divided into horizontal stripes same in number as the number of threads and then each stripe is passed-on to a separate thread.
- We tried to provide different divisions of the original frame depending upon the number of threads(dividing the original into four squares) but observed no significant advantage. Hence, we decided to stick to a generic implementation for the sake of uniformity.

Results

The following plot describes the obtained results.

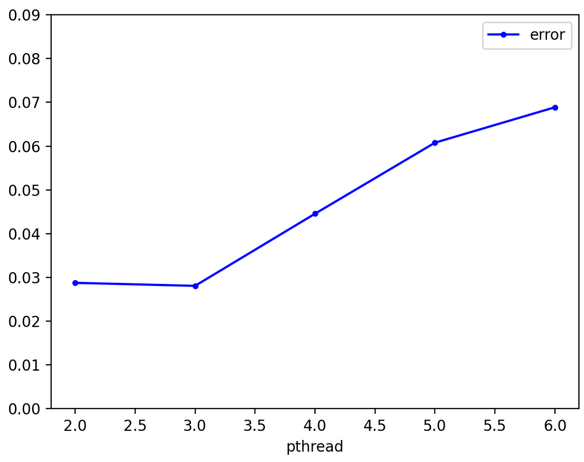

Figure 7: Error v/s Number of threads

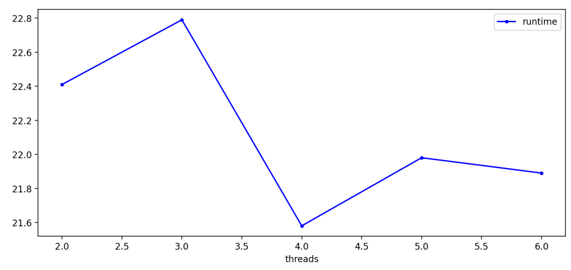

Figure 8: Runtime v/s Number of threads

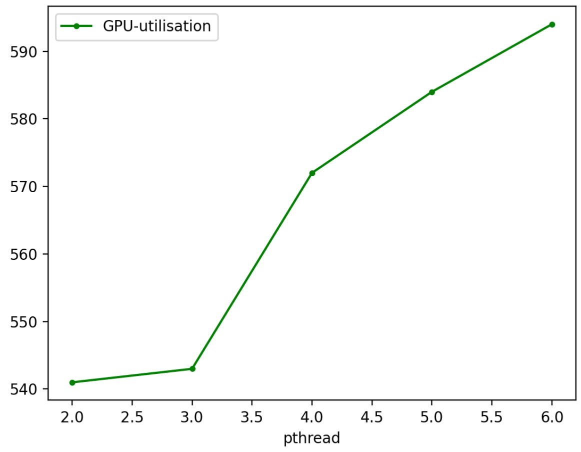

Figure 9: CPU usage v/s Number of threads

Observation & Explanation

- An increase in the error is observed with increase in number of threads. By dividing the original frame into several frames, we create several "edges/boundaries" in the image. In the image processing involved in dense optical flow, the edge pixels are not processed as properly as others. Hence the error increases as the number of threads in spatially concurrent sparse optical flow increases.
- The plot shows no observable trend between runtime as the number of threads increases.
- A likely explanation that we came up was that the frame processing required for queue density is not large enough to result in a noticeable runtime gain on parallelization.**In the case of queue density estimation, the bottleneck for runtime is the time needed for capturing the frames and not the subsequent image processing**. Any minuscule gains by parallelizing are nullified by the overhead required for creating threads. No observable indications show up on CPU utilisation either.
- To validate our hypothesis, we ran code that just went through all the frames with the same speed as the baseline code and indeed, it's runtime was sufficiently close to the baseline runtime, hence, strengthening our hypothesis.
- As the number of threads increases, the average CPU usage increases since more of the resources get allocated to the running program.

'

Method-4 : Temporal Multi-threading

Methodology

- After capturing, every i-th frame is alloted to the i-th pthread. The return values from threads are sorted on the basis of timestamps attached to the frame.

Results

The following plot describes the obtained results.

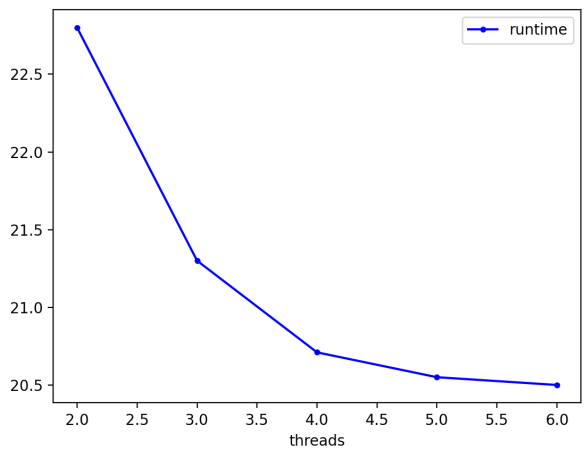

Figure 10: Runtime v/s Number of threads

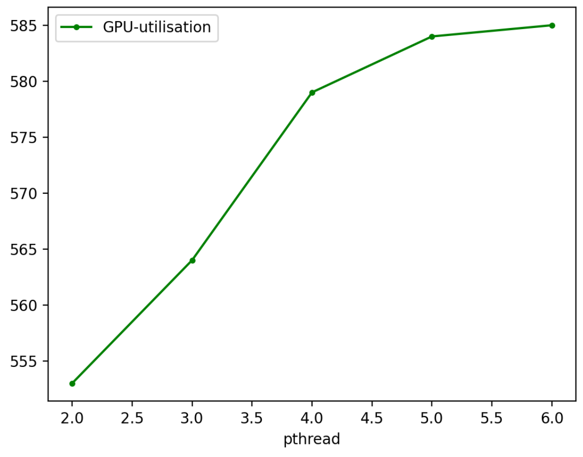

Figure 11: CPU usage v/s Number of threads

Observation & Explanation

- The error in temporal concurrency is negligible in comparison to that in spatial concurrency. This is reasonable since the frames are processed exactly similar to the way in the non-concurrent setting.
- A decrease in runtime is observed with increase in number of threads since more number of frames are being processed in parallel.
- As the number of threads increases, the average CPU usage increases since more of the resources get allocated to the running program.

Sparse and Dense Optical Flow

Methodology

- In dense optical flow, we have used the contours(set of convex hulls) of white pixels to estimate the fraction of the road covered by moving vehicles.

- In sparse optical flow, we have estimated the area by proportionately relating it to the number of *Shi-Tomashi Points* that have a sufficient displacement in two consecutive frames.

Results

	Sparse Optical Flow	Dense Optical Flow
Runtime(secs)	30.51	210
Error	0.0853835	0(baseline)

Observation & Explanation

- A relative error of around 15% is observed in sparse optical flow.
- This is reasonable since sparse optical flow estimates the motion of object by tracking a determined set of points whereas dense optical flow tracks all the pixels. Hence, for the calculation of area, the dense version is more suitable as we intend to distinguish between large and small vehicles.
- Since sparse flow deals with a smaller set of points, it take lesser time than dense flow.

Temporal and Spatial Multi-threading on Dense optical flow

Methodology

- In the spatial case, we split up the dynamic density estimation procedure to different threads concurrently by dividing the frames into vertical stripes and passing them on to threads.
- In the temporal case, we gave every i-th frame to the i-th thread.

Results

The following plot describes the obtained results.

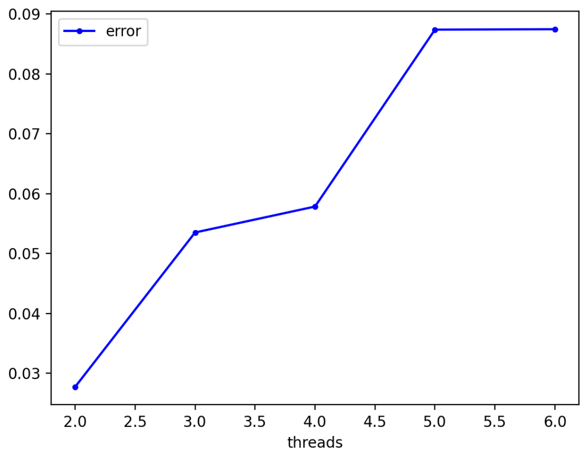

Figure 12: Error v/s Number of spatial threads

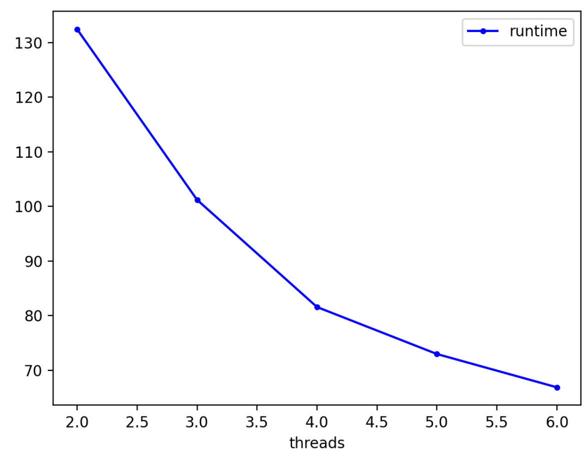

Figure 13: Runtime v/s Number of spatial threads

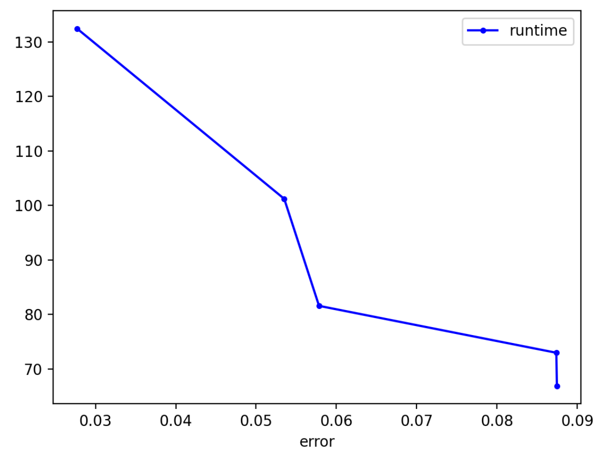

Figure 14: Error v/s Runtime (in case of spatial threads)

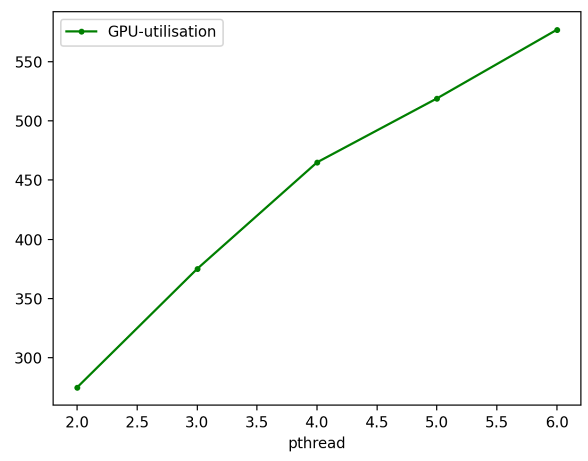

Figure 15: CPU usage v/s Number of spatial threads

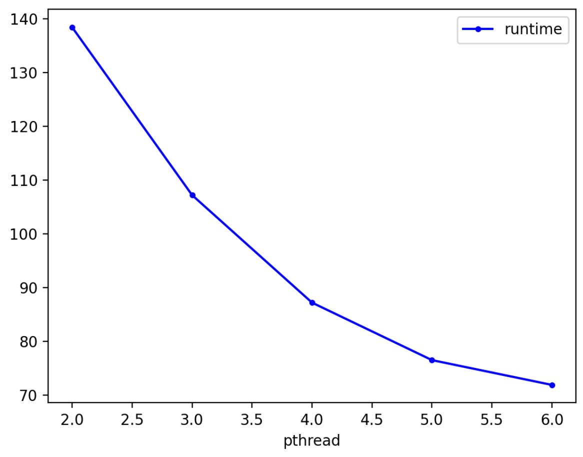

Figure 16: Runtime v/s Number of temporal threads

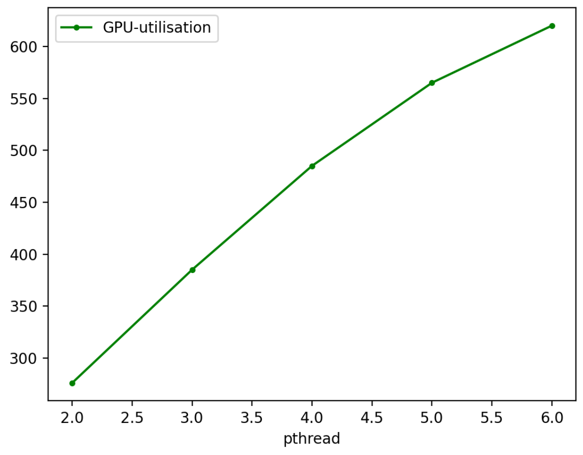

Figure 17: CPU usage v/s Number of temporal threads

Observation & Explanation

- A noticeable runtime v/s utility trade-off is observed in this case. The runtime decreases and the error increases as the number of threads increases.
- This is reasonable since dynamic density estimation is expensive with respect to runtime and dividing its work into threads is likely to reduce runtime substantially.
- By Dividing the original frame into several frames, we create several "edges/boundaries" in the image. In the image processing involved in dense optical flow, the edge pixels are not processed as properly as others. Hence the error increases as the number of threads in spatially concurrent dense optical flow increase.
- Also, the error in temporal concurrency is negligible in comparison to that in the spatial case which is reasonable since there are no boundary issues in temporal threads, the whole original frame is given to one or the other thread. This shows that temporal concurrency is a better option than the spatial counterpart, given the same number of threads.
- The CPU usage increases with increase in the number of threads since more of the resources get allocated to the running program.

Conclusion

- Method-1,2 , Sparse v/s Dense optical and concurrency in dense optical flow show an obtrusive runtime-utility trade-off.
- However, concurrency seems to gives no advantage in the background subtraction procedure, which is explainable. Concurrency would have led to substantial runtime gains, had the bottleneck procedure for runtime was also parallelized which is not the case with us. Implementing dense optical flow concurrently would have shown the real gains of concurrency.