Assignment 1, Subtask 3: COP290

Divyanka Chaudhari (2019CS50429)
Kshitiz Bansal (2019CS50438)

April 2, 2021

1 Metrics

1.1 Method 1:

- Utility – We have taken it as described in the problem description on the course web-page. The sum of absolute value of difference of queue density of variant program and baseline as the error. And utility as inverse of error. The only difference being that we took sum as the error instead of mean or median.

$$\text{error} = \sum |(\text{variant}_{\text{queue\_d}} - \text{baseline}_{\text{queue\_d}})|$$

$$\text{utility} = \frac{1}{\text{error}}$$

- Parameter – It is the number of frames on which we have sub-sampled the program.

1.2 Method 2:

- Utility – We have taken it as described in the problem description on the course web-page. The sum of absolute value of difference of queue density of variant program and baseline as the error. And utility as inverse of error. The only difference being that we took sum as the error instead of mean or median.

$$\text{error} = \sum |(\text{variant}_{\text{queue\_d}} - \text{baseline}_{\text{queue\_d}})|$$

$$\text{utility} = \frac{1}{\text{error}}$$

- Parameter – It is the factor by which the x and y dimensions of the frame are resized. If the parameters are fx, fy and dimensions are dim_x, dim_y, then the new dimensions are $fx * dim_x$,$fy * dim_y$ respectively. Keeping user convenience in mind, input factors are truncated to 2 decimal places. Eg: 0.2563 will be converted to 0.25 and then used (For easy analysis of the plot).

1.3 Method 3:

- Utility – As the work is still being done on all the frames, the utility is zero. We use run-time vs parameter graph to analyse.
- Parameter – It is the number of threads in which the program is split into with each thread working on a split section of the frame.

1.4 Method 4:

- Utility – As the work is still being done on all the frames, the utility is zero. We use run-time vs parameter graph to analyse.
- Parameter – It is the number of threads in which the program is split into with each thread working on different frames.

2 Methods

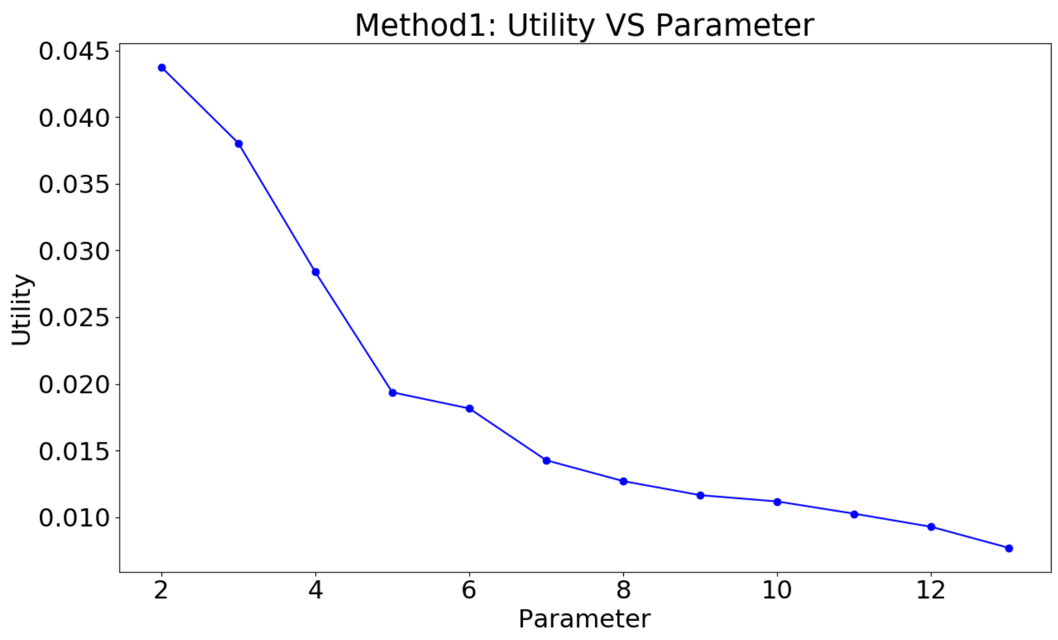
- **Method 1:** Successfully implemented sub-sampling. Program processes every n^{th} frame where n is the parameter.
- **Method 2:** Successfully implemented change of resolution. Program changes resolution of each frame before processing. Program asks for $x - factor$ and $y - factor$ as input, that are the factors by which the dimensions are changed.

$$\text{new}_{\text{dim}} = \text{original}_{\text{dim}} * \text{factor}$$

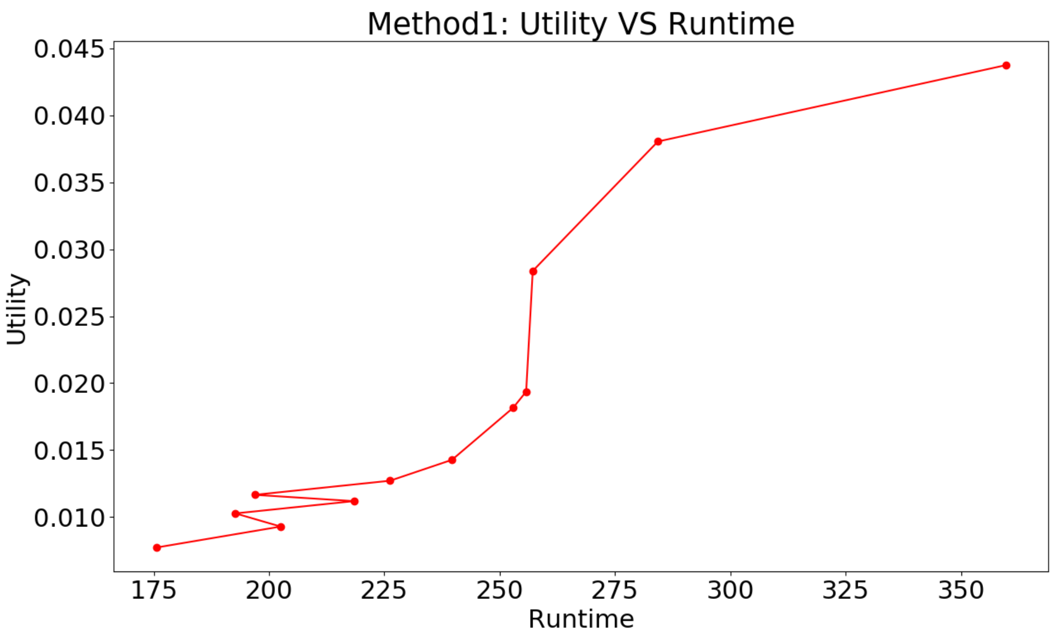
  where factors are fx and fy in code
- **Method 3:** Successfully implemented spatial threading. Program processes sub-strips of each frame in different threads based on the number of threads given as parameter.
- **Method 4:** Successfully implemented temporal threading. Program processes the frames in different threads based on the number of threads given as parameter.

3 Trade-off analysis

3.1 Method 1

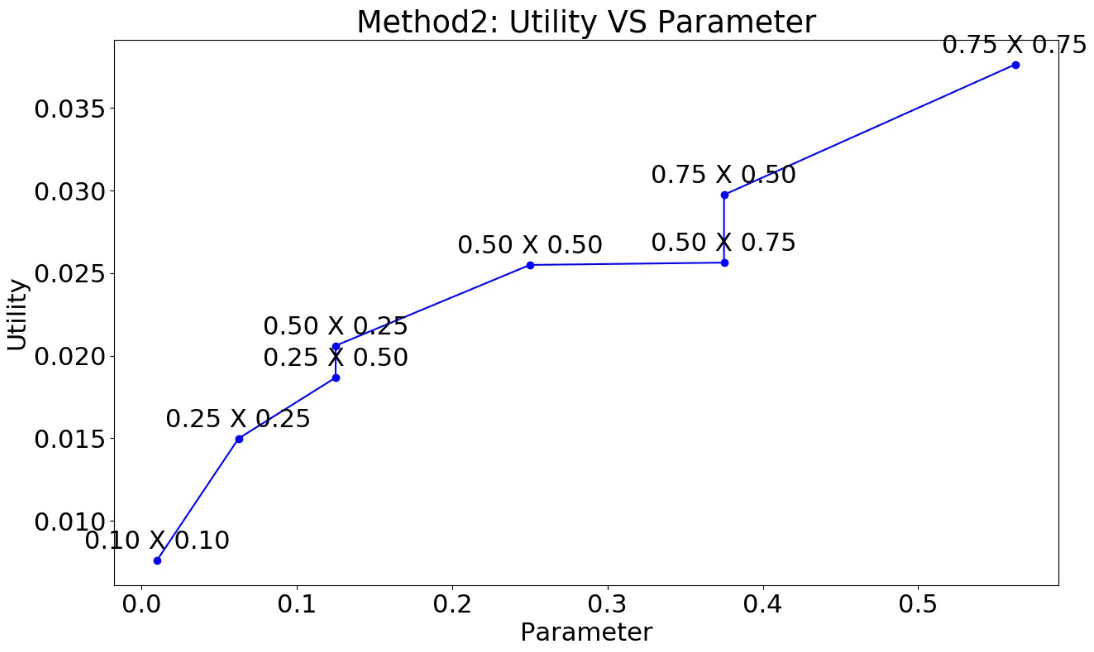


In this method, the parameter is the n, where every n^th frame is processed. From the graph we can clearly understand, that the utility of the program goes down as we increase the value of n. Also, this is quite intuitive because when we skip some frames in between, we assume that the density remains constant but in reality it does not. So the higher number of frames in which we assume it to take the value of its nearest processed frame, the higher is the error and thus lower is the utility.



From the plot, we understand that the utility increases as the processing time increases. But we know intuitively that processing time has to increase with decreasing n. (The less number of frames are not processed in between, higher the time)

3.2 Method 2



To be able to visualize, we have created a new parameter which is the product of the x-factor and the y-factor. X-factor (fx) and Y-factor (fy) are the factors by which the width and the height of all the frames is resized.

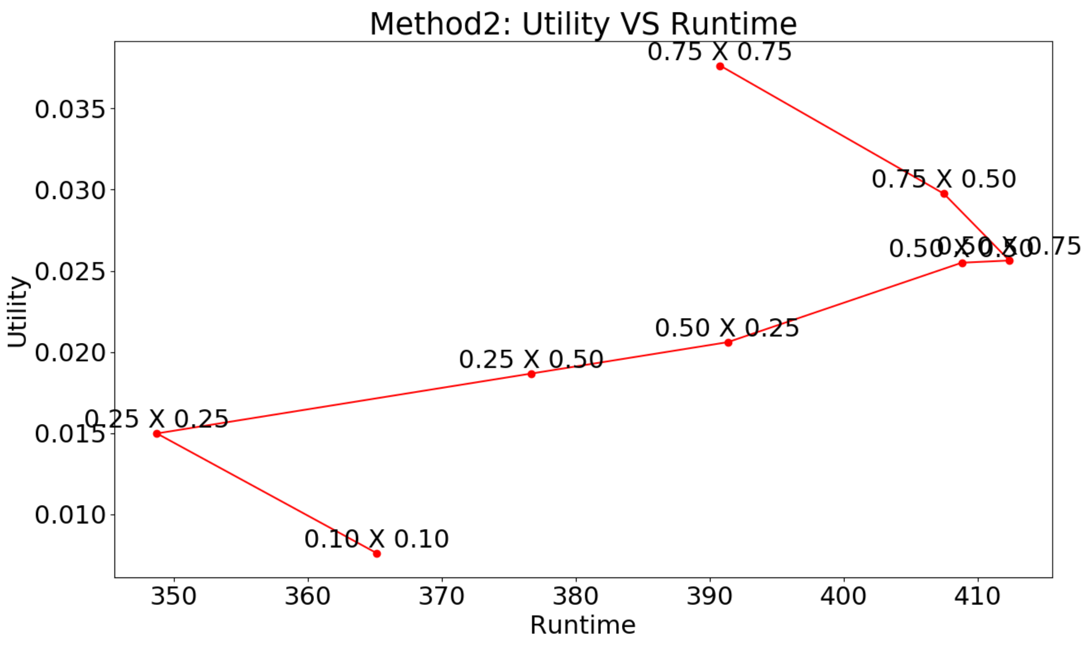
$$\text{new\_y}_{dim} = fy * \text{original\_y}_{dim}$$

$$\text{new\_x}_{dim} = fx * \text{original\_y}_{dim}$$

$$\text{parameter} = fx * fy$$

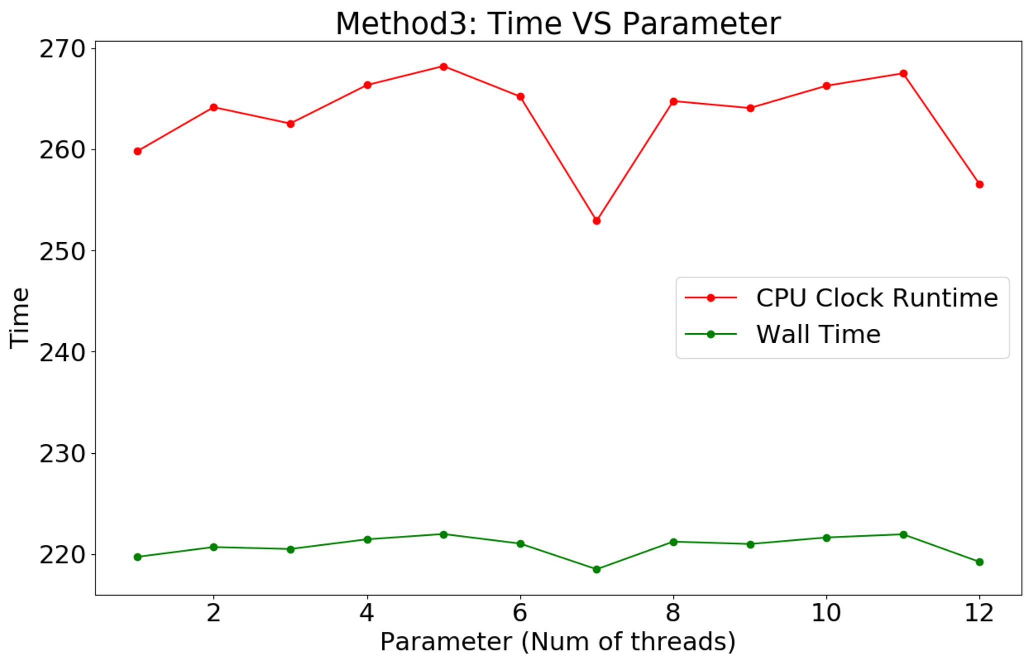
From the above graph, we can see that the utility increases as the parameter approaches 1. This is also quite intuitive as the baseline queue density is this new queue density with fx =1 and fy = 1.

One very interesting takeaway from the graph is that as the original image was rectangular (not square), there is a difference in utilities when fx and fy are interchanged. It would be interesting to test this on a square shaped image.



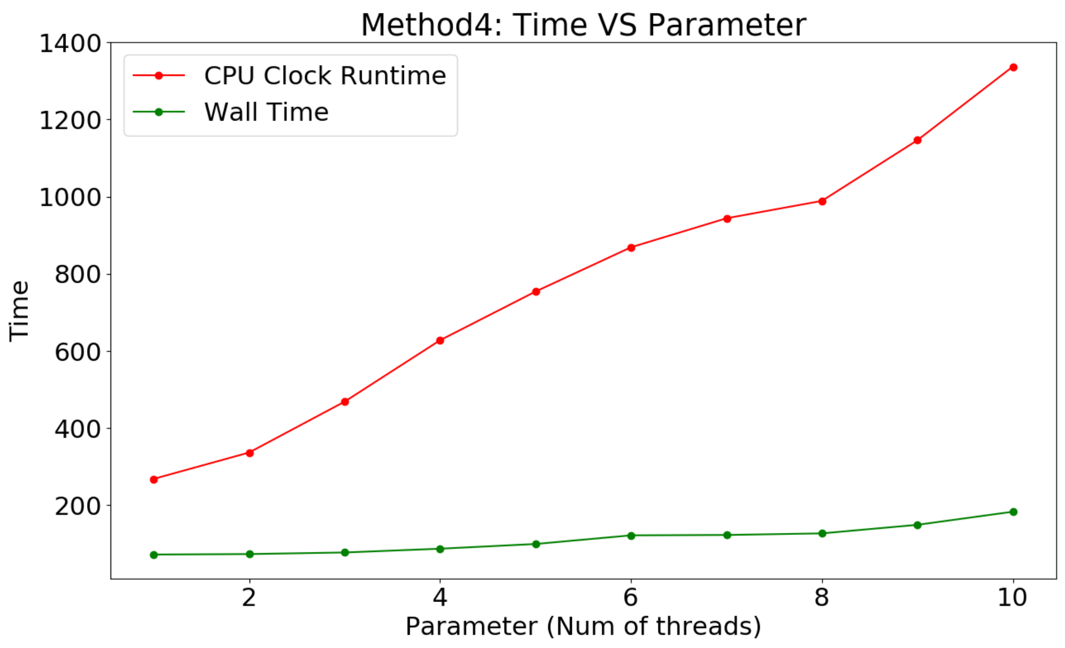
This is one of the most interesting graph of the ones we have seen till now. The nature of the graph is intuitive with respect to change in utility with product of fx and fy. (higher parameter $\implies$ higher utility as there is less loss of information while resizing) But the nature of the utility vs run-time curve does not match with our intuition. In order to try to reason such a curve, there is a possibility that the resizing images part is becoming the bottle neck or the main contributor to run-time and is thus changing very weirdly.

3.3 Method 3



We measured both the CPU time and the wall time to analyse with change in parameter. From the graph, we observe that the times achieve a minima around 7 threads. The run times are decreasing till 7 and then start increasing slowly. It is quite intuitive that it reaches a minima. Initially as the number of threads increase, the time decreases as the processing is divided among them, but with a higher number of threads, the time of creation and joining of threads overcomes its benefits, and thus it starts increasing.

3.4 Method 4



Here, again, we measure CPU clock time and Wall clock time both. Since CPU clock time also measures total time taken by thread, it increases as number of threads increases. This proves that the program is multi-threaded. The wall clock time remains almost constant throughout until 5 threads. But, later with a higher number of threads, the time of creation and joining of threads overcomes its benefits, and thus it starts increasing.