CSC367 Assignment 2 report

Xuan Du, Tianchen Xu

October 2021

Part 1

Run the sequential algorithm as well as the parallel methods for a given image. For each run, vary the number of threads from 1 to the total number of cores by doubling the number of threads. Consider checking the L1 cache misses and explaining why and how these factor into your results.

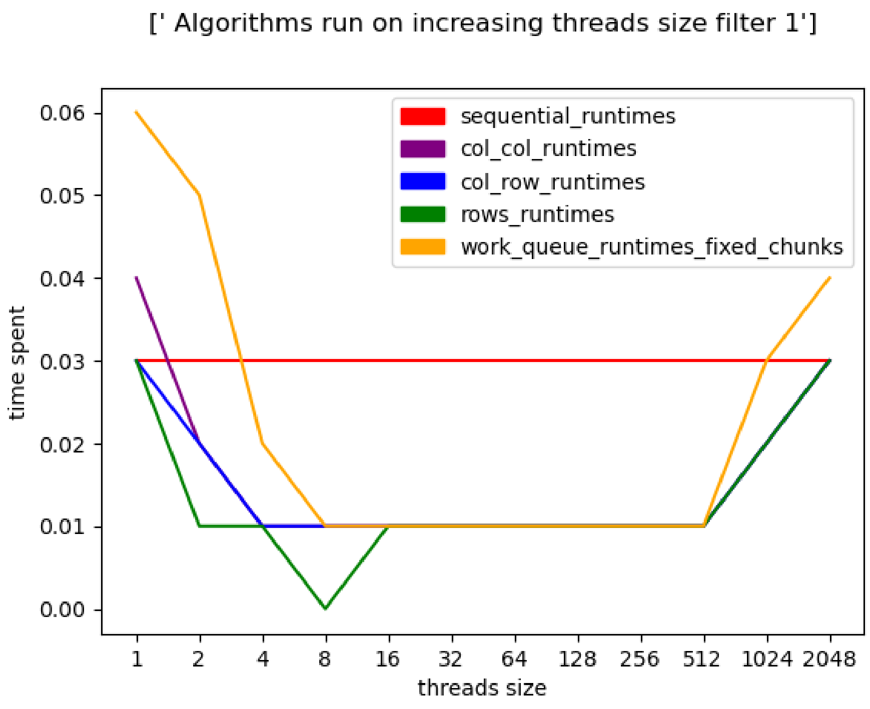
For the work queue implementation, the length of the tile is set to be equal to the number of threads used. That is, the chunk will be N x N, where N is the number of threads. Remember that your implementation uses synchronization regardless of N. In fact, observing the locking overhead when N=1 is an interesting aspect to consider.

1. Sequential vs Parallel

   For sequential method, we can clearly observe the time taken for built-in big image is 0.03ms, the parallel method starts with about 0.03ms, when threads increase, the time taken goes down to 0.01ms. Interestingly, the time taken for sequential method goes back up when threads grows to 1024. The reason here is the size of built-in big image is 1024 x 1024, if the number of threads are above 1024, a lot of wasted threads would produced and working thread are not well-divided. Also when threads size are large, the L1 will continue to hit misses. The col col major implementation is slower then col row major at thread = 1 because spatial locality.

2. Work Queue Implementation

   For work queue implementation, we can observe that when chunk size is 1, the time taken to complete image processing is 0.06ms, two time the time of sequential method. Let us take a look on the code, we used mutex lock when threads are taking a tile to process, when chunk size is 1, such operations would take too much time to do. If thread size goes up, work queue implementation behave similarly with parallel method.



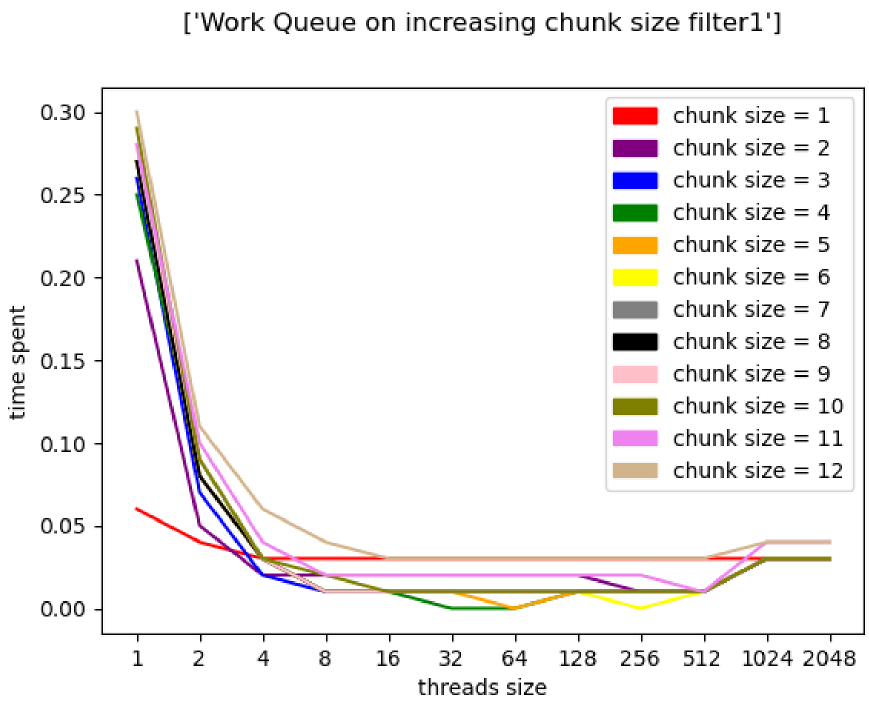
```
// 1) Apply the filter on the image
while (chunk_in_filter < chunks_count) {
    chunk serve;

    pthread_mutex_lock(&mutex);
    if (chunk_in_filter >= chunks_count) {
      pthread_mutex_unlock(&mutex);
      break;
    }
    // filter next waiting chunk in queue
    serve = chunks_queue[chunk_in_filter];
    chunk_in_filter += 1;
    pthread_mutex_unlock(&mutex);

    apply_filter2d_by_range(id, cw->f, cw->original_image,
    cw->output_image, cw->width, cw->height, serve.row_start,
    serve.row_end, serve.col_start, serve.col_end, false);
}
```

Part 2

Test the work pool method for a variety of chunk sizes. For example, using N threads, you could measure the time depending on chunk size, by changing the chunk size: 1x1, 2x2, 4x4, 8x8, 16x16, 32x32, etc. Plot a separate line for each N between 1 and the number of cores on the same graph.



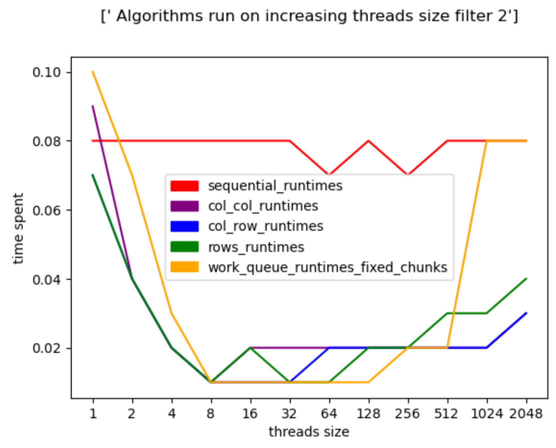
1. Thread = 1, how chunk size affect efficiency

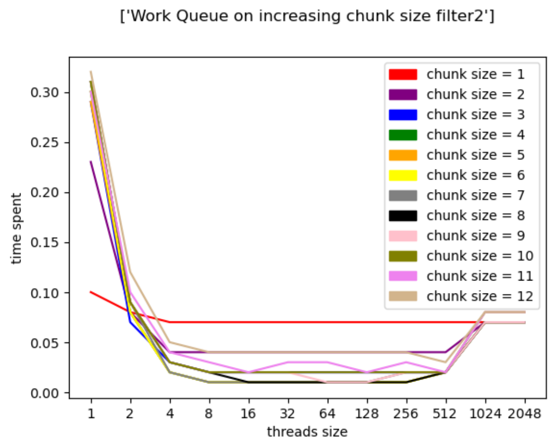
   When we create one thread to process the image by work queue implementation, we can observe that larger chunk size cause less time to compute. It is because the mutex lock implementation that we discussed in part1. So larger chunk size, less time compute.

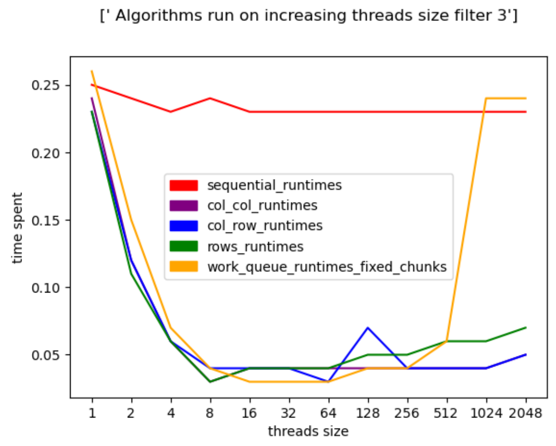
2. General Behaviour Similar to parallel methods, time taken decrease when thread number increases. When threads are more than 1024, the time goes back up again.

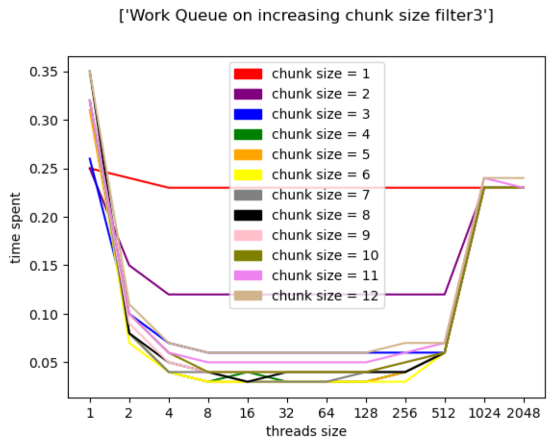
Part 3

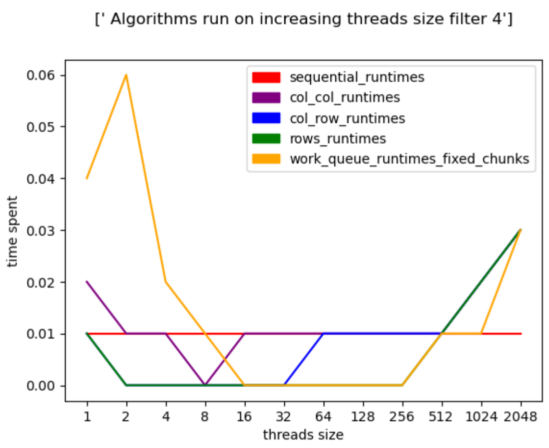
Now vary the filter size for each of the methods (including the sequential one). Keep the number of threads constant to an N = the number of physical cores, and (for work pool) use a chunk size of N x N. You may additionally vary these if you wish, or if you plan to gain further insights.

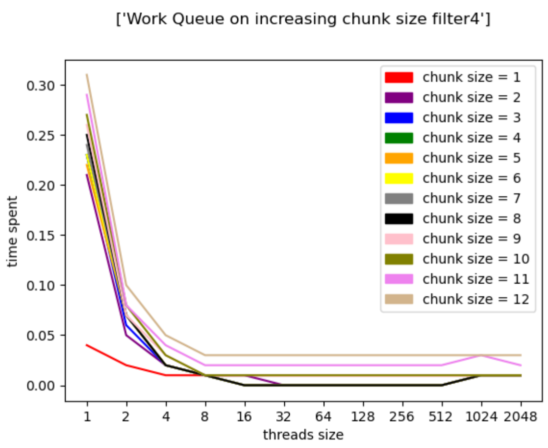












1. Different patterns with different filter

   In general, we observe that the bigger the filter is, the longer the time taken to compute image. It is obvious since more computation required. When we use 9 x 9 filter, we can see the time taken is 0.25ms, which is 4 times the 3 x 3 filter time. Larger filter also meaning more room to improve the efficiency, for example 9 x 9 filter when thread number can well-divide the tasks, the compute time can be improved from 0.25ms to 0.04ms (500% improvement).

2. Identity filter does not improve efficiency of work queue implementation
   We can observe that for work queue implementation, the change of filter does not largely affect the performance of image processing. Reversely, we can assume that work queue implementation can take advantage on big filter that will not loss performance.