COL 380 : Lab 1

Sagar Goyal 2015CS10253

Quick Hull

Algorithm Used :-

The algorithm that is implemented by me was implemented in an iterative manner, opposite from the most common recursive versions of the algo available on the internet.
A struct called region was defined separately that took 2 points which represented the line joining them and a vector of points lying in a particular region of space on a side of the line.

Two other functions were defined ‘findside’ and ‘lineDist’ which told us the side of line on which a point lies of the line and the distance of the point from a line.

The entire area was first divided into two areas using two points with the maximum and the minimum point respectively. These two areas were made in the form of the struct ‘region’. For each of these regions the point P at the maximum distance from the line was found out and was used to divide a region further and was added in the list of answers as this will be a part of the convex hull.

P was also used to make new regions with points A & P and P&B with the points that were not a part of the triangle APB where A&B were the points of the region to start with.

This process was followed till no more regions were left in the vector and no points were left to classify i.e. the points in any region were zero and after a while no new regions were formed.

Parallel: To do this algorithm in parallel, the process of removing the regions (in a critical section) from the vector and performing the calculation was done parallely on all the regions and new regions were added in the vector in a critical section.

Finally, the ans array we have formed will contain the convex hull.
Every new point that we doscover as the maximum in any region is added to the Result vector in a critical section and when the algorithm stops this contains all the points of the convex hull.

Now to implement this algorithm in parallel, what I did was that I made all the working in one single iteration as parallel. That is, if there are say ‘n’ regions in the array and ‘p’ threads were available each thread will take ‘n/p’ regions from the array, work on them and then create new regions and the iterative process will start again with a different size of the array of divisions this time.

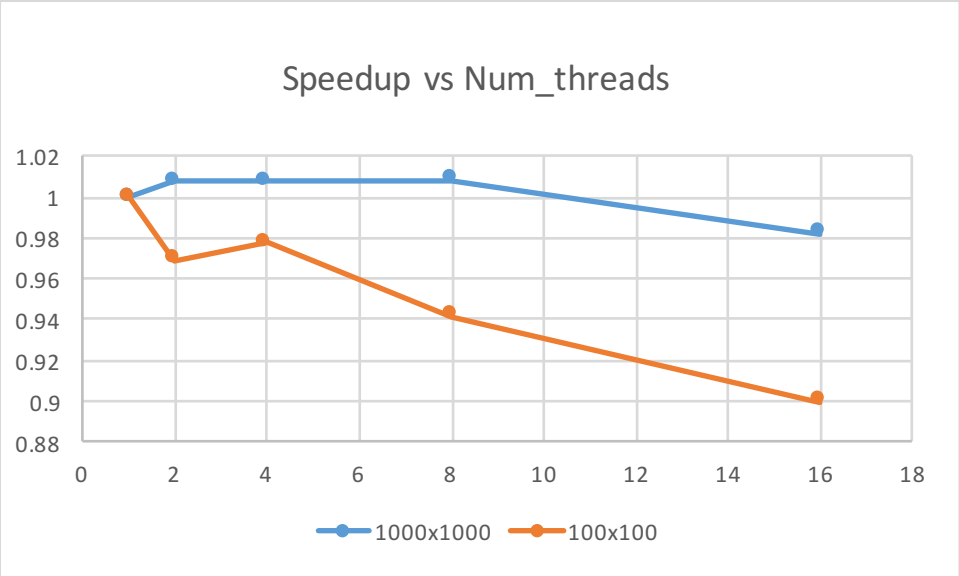
Speedup and Efficiency plots :-

Specifications used-

An image of size 1000*1000 filled with randomly generated 0s and 1s as inputs. The same matrix was used for all the computations.

Average time taken for 20 runs for every number of threads.

Speedup vs Number of Threads :-

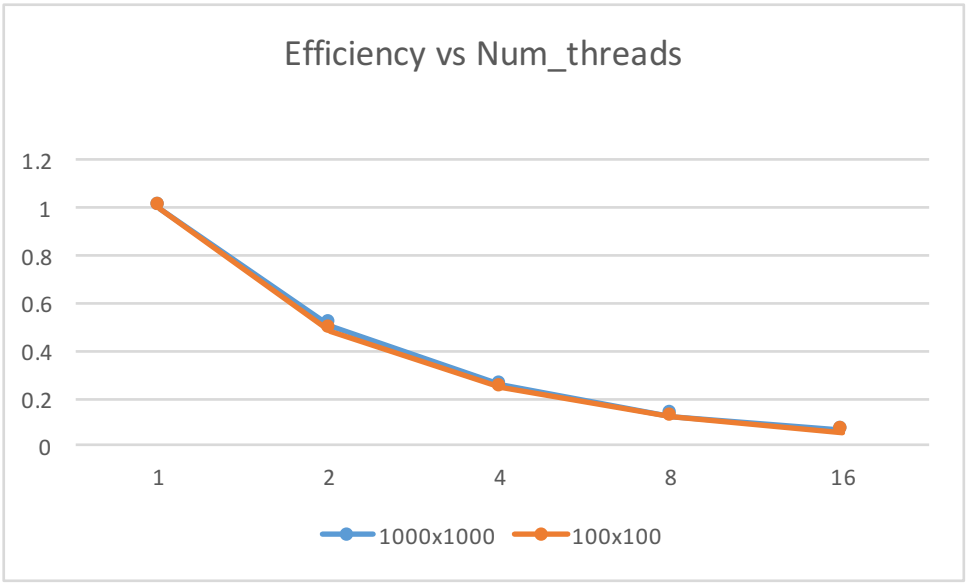


Ideally the speedup should have been equal to 'p' for each case but parallel programming involves a lot of overheads in inter-process communication and hence the speedups obtained are considerably less.
The number of threads that the computer can run at a time is limited by the number of cores on the pc and even if we keep increasing the number of threads in the code it doesn't make much difference and rather increases the overhead without any improvement in the time of execution and hence reducing the speedups on increasing the threads beyond 4.

For smaller amount of data the overhead of parallel processing becomes way larger than the time saved in parallel execution because the total time that a program takes in linear execution is already less due to the small amount of data that is being processed and hence the speed up continuously decreases on increasing the number of threads when the data size is small.

For smaller amount of data the overhead of parallel processing becomes way larger than the time saved in parallel execution because the total time that a program takes in linear execution is already less due to the small amount of data that is being processed and hence the speed up continuously decreases on increasing the number of threads when the data size is small.

Efficiency vs Number of Threads :-



We can clearly observe that the efficiency decreases with the increase in number of threads as the speedup doesn't increase at the same pace as the number of threads. However we do observe that the decline in efficiency is similar for both the problem sizes and the problem size has no effect on the efficiency and the number of threads relation.