CONVEX HULL REPORT

Name: Udayin Biswas
Entry Number: 2014CS50296
Machine Specifications:2.5 GHz I5 core(quad core), 4 gb ram=> 4 THREADS assuming max

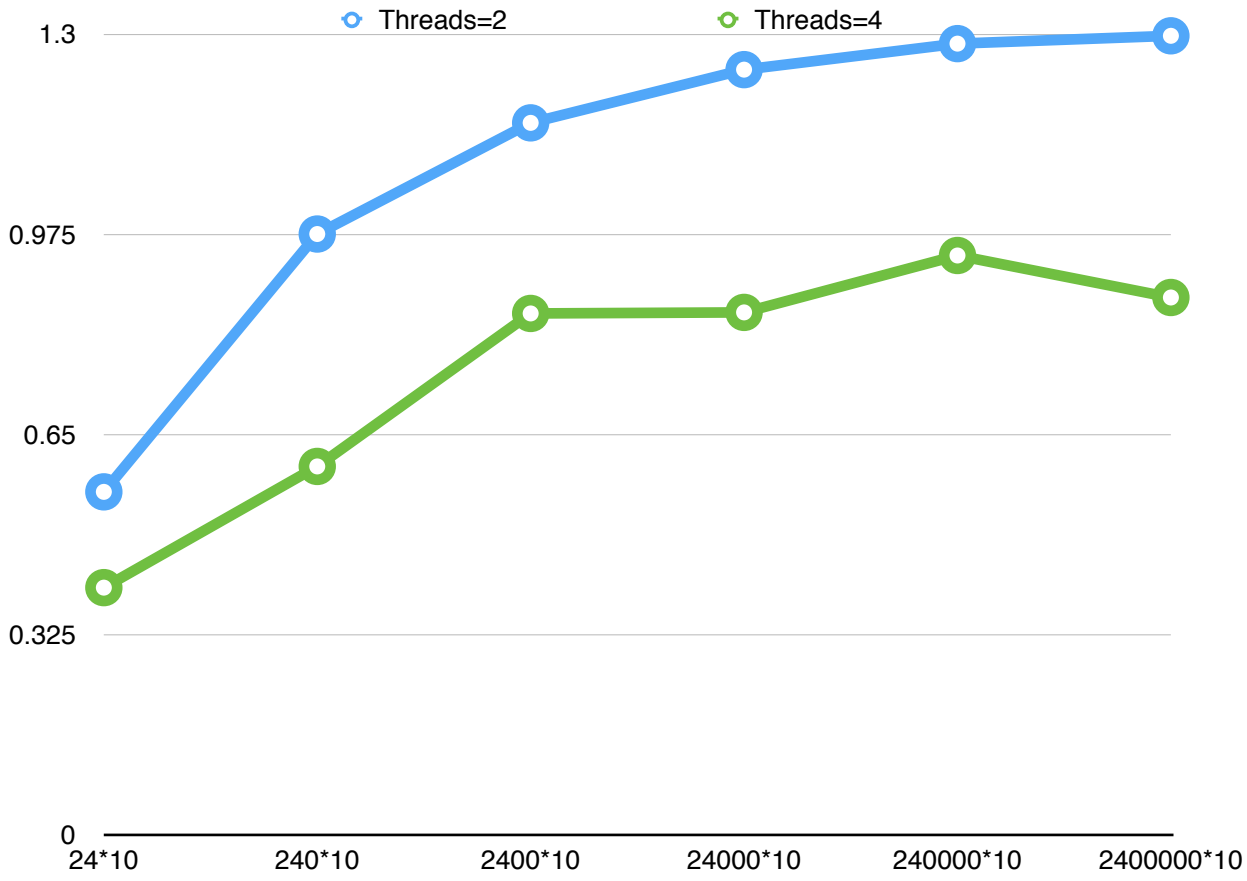
---

Convex hull method to get the convex hull of a given 1 bilevel image.
First the points which are 1 were extracted from the images and their coordinates stored.
Now we had to find convex hull of these points i.e. those points which on joining bound all the other remaining points to make a convex shape and minimum such no. of points.

Convex hull strategy was to first find the leftmost L and rightmost points R, then see which points lie above and lie below to divide them into 2 sets. **We can run a parallel strategy to work on these 2 sets side by side(divide into 2 threads t1 and t0)**. After this, in each set, we again divide it in 3 parts:
First we get the furthest point F in that set from line joining L and R. Then 3 REGIONS we get:
(1) Triangle joining F,L,R -> These points shouldn't be considered since they lie inside.
(2) Points external to LF line
(3) Points external to RF line.

(2) and (3) will require to run parallel loops again to divide the work among the threads. So we can consider them nested loops which require the original thread to be forked again. This was done by omp_set_nested(1); strategy. So original t0 can split into t0 and t2 thread while t1 into t1 and t3 and so on into more threads after each recursion of the quickHull function.
So my strategy to parallelise was basically fork each thread into two at each stage unless and until we run out of available threads set by the user.

Speed up vs problem size



Speed up increases very gradually in the case of n=2 and n=4 threads, especially when 4 threads. Efficiency is also very low in 4 threads case while in 2 threads case it gradually approaches to 0.7 mark for the given problem sizes.