Convex Hull
Report

Saket Dingliwal 2015CS10254

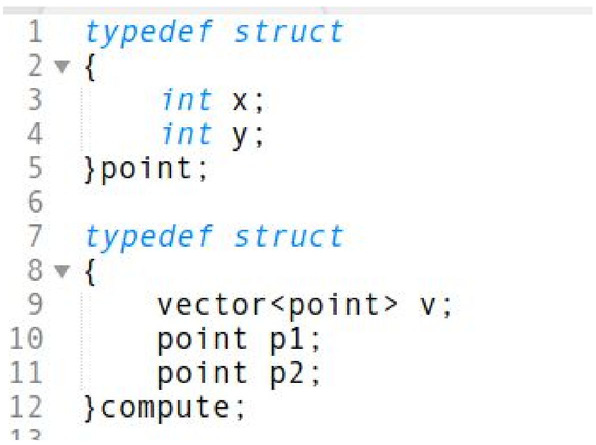
Algorithm

The algorithm used is quick-hull algorithm. In this, divide and conquer technique is used. A list of points and 2 extreme points form a single computation. For these 2 extreme points, the point with the maximum distance from is identified and put in the convex hull and then the list is separated into 2 new lists. This continues till the list of the computation becomes empty. For parallelizing the algorithm, omp pragma for construct is used. The task is divided into two parts and put in a new array. After all the threads divide their piece of task into two parts, the new list of tasks formed is solved by next team of threads. The answer is updated in a critical block.

Implementation

A structure compute is made which represents a unit of computation. Two compute lists are made. The parallel for construct works on the compute list and divide the task into two halves, and populate the new_compute list in a critical section. After all the computes have been evaluated, the new_compute list is copied into the original compute list. This continues till the compute list gets empty. So in each solving of compute there are at most O(n) computations. And the number of such computes can be equal to number of points in the Answer. Since such computes are handled in parallel so the time taken for these computes is O(answer/p). So the worst case complexity of the algorithm is O(answer*n/p). However the complexity for the average case may be lower.

The structure defined is as follows->



```
typedef struct
{
    int x;
    int y;
}point;

typedef struct
{
    vector<point> v;
    point p1;
    point p2;
}compute;
```

Plots

The algorithm is run on a sample image file and different number of threads. The laptop used has the specification Intel® Core™ i3-5005U CPU @ 2.00GHz × 4.
On this 4 core processor, the average of 20 runs with different values of p are tabulated.
The speed up and efficiency curves are also tabulated. The time taken with num_threads = 1 is used to calculate the serial time and hence speed up.
The image file used is Seahorse.pbm for testing.

No of Threads	Time	Speed up	Efficiency
1	0.988889	1	1
2	0.781259	1.25	0.62
4	0.773588	1.28	0.32
8	0.80804	1.23	0.15
16	0.901466	1.1	0.07

Analysis

Small speed ups are being achieved because of the large nature of the input image and fewer points. There is a critical section which prevents all threads to bring a good speed up. Lowest time is obtained for p = 4 which is almost same for p=2. The efficiency is highest for p = 2 and can be clearly seen in the results. This is because the code is run on a quad-core processor in which mostly 2 of the processors are available. The speed ups are limited mainly due to overheads of critical section, large serial parts of input.