#include <stdio.h>

#include <math.h>

#include <omp.h>

#include <stdlib.h>

#define LEFT 2the

#define RIGHT 1

#define int long long int

#define size 100000005

#define print 0

int px[size][2];

int hull[size];

/\* Orientation uses the idea of slope of the line segments to get the value of val “

If slope of p-q is more than q-r than clockwise orientation of p-q-r

Else If slope of p-q is less than q-r than counter-clockwise orientation of p-q-r

Else they are collinear

The formula of val is the derived version of these conditions

\*/

int orientation(int p, int q, int r)

{

int val = (px[q][1] - px[p][1]) \* (px[r][0] - px[q][0]) -

(px[q][0] - px[p][0]) \* (px[r][1] - px[q][1]);

if (val == 0) return 0;

return (val > 0)? RIGHT: LEFT;

}

void convexHull\_parallel(int n, int nthr)

{

// There must be at least 3 points

if (n < 3) return;

int ptr=0;

// Find the leftmost point

int q[nthr], i, l;

for(i=0;i<nthr;i++){

q[i] = 0;

}

#pragma omp parallel for

for (i = 1; i < n; i++){

if (px[i][0] < px[q[omp\_get\_thread\_num()]][0]){

q[omp\_get\_thread\_num()] = i;

w}

}

for(i = 1 ; i < nthr ; i++){

if(px[q[i]][0]<px[q[0]][0]){

q[0]=q[i];

}

}

l=q[0];

// Start from leftmost point, keep moving counterclockwise

// until reach the start point again. This loop runs O(h)

// times where h is number of points in result or output.

int p = l;

do

{

// Add current point to result

hull[ptr++]=p;

/\* update q[thnum] for each thread by doing following

Search for a point 'q[thnum]' such that orientation(p, x,

q[thnum]) is counterclockwise for all points 'x'. The idea

is to keep track of last visited most counterclock-

wise point in q[thnum]. If any point 'i' is more counterclock-

wise than q[thnum], then update q[thnum].

for(i=0;i<nthr;i++){

q[i] = (p+1)%n;

}

#pragma omp parallel for

for (i = 0 ;i < n; i++)

{

//thnum = omp\_get\_thread\_num();

// If i is more counterclockwise than current q, then

// update q

if (orientation(p, i, q[omp\_get\_thread\_num()]) == LEFT)

q[omp\_get\_thread\_num()] = i;

}

int qans=q[0];

for(i=0;i<nthr;i++){

if (orientation(p, q[i], qans) == LEFT)

qans = q[i];

}

/\* Among all the q array elements find the point that is most counter clock wise among all of them as done for each thread before .

Now qans is the most counterclockwise with respect to p

Set p as qans for next iteration, so that q is added to

result 'hull'\*/

p = qans;

} while (p != l); // While we don't come to first point

int index = -1;

// Print Result

if(print)

{for (i = 0; i < ptr; i++){

index = hull[i];

printf("(%lld, %lld)\n",px[index][0],px[index][1]);

}

printf("Size of polygon = %lld\n", ptr);

}

}

void convexHull\_serial(int n)

{

// There must be at least 3 points

if (n < 3) return;

int ptr=0;

// Find the leftmost point

int l = 0,i;

for (i = 1; i < n; i++)

if (px[i][0] < px[l][0])

l = i;

// Start from leftmost point, keep moving counterclockwise

// until reach the start point again. This loop runs O(h)

// times where h is number of points in result or output.

int p = l, q;

do

{

// Add current point to result

hull[ptr++]=p;

// Search for a point 'q' such that orientation(p, x,

// q) is counterclockwise for all points 'x'. The idea

// is to keep track of last visited most counterclock-

// wise point in q. If any point 'i' is more counterclock-

// wise than q, then update q.

q = (p+1)%n;

for (i = 0; i < n; i++)

{

// If i is more counterclockwise than current q, then

// update q

if (orientation(p, i, q) == LEFT)

q = i;

}

// Now q is the most counterclockwise with respect to p

// Set p as q for next iteration, so that q is added to

// result 'hull'

p = q;

} while (p != l); // While we don't come to first point

int index = -1;

// Print Result

if(print)

{for (i = 0; i < ptr; i++){

index = hull[i];

printf("(%lld, %lld)\n",px[index][0],px[index][1]);

}

printf("Size of polygon = %lld\n", ptr);

}

}

int main(int argc, char \*\*argv)

{

int n,i;

if(argc<3){

printf("ERROR!\n Incorrect number of arguments\n");

return 1;

}

n=atoi(argv[1]);

int nthr=atoi(argv[2]);

omp\_set\_num\_threads(nthr);

for(i=0;i<n;i++){

px[i][0]=rand()%10000000;

px[i][1]=rand()%10000000;

}

double start,end,ser,par;

printf("\n\tFor Size -> %d and number of threads -> %lld\n",atoi(argv[1]),nthr);

start = omp\_get\_wtime();

convexHull\_parallel(n, nthr);

end = omp\_get\_wtime();

par=end-start;

printf("Parallel Time : %.10f\n", end-start);

start = omp\_get\_wtime();

convexHull\_serial(n);

end = omp\_get\_wtime();

ser=end-start;

printf("Serial Time : %.10f\n", end-start);

printf("Speed Up = %.10f\n",ser/par);

printf("----------------------------------\n");

return 0;

}