Title: Closest Pair of Points using Divide and Conquer algorithm

Name:[1]Anurag Sinha,[2]Rohan Sharma,[3] Jhabar Singh Bhati ,[4] Mayank Sharma

(USN:[1]1MS18CS022,[2]1MS18CS047,[3]1MS18CS052,[4]1MS18CS070)

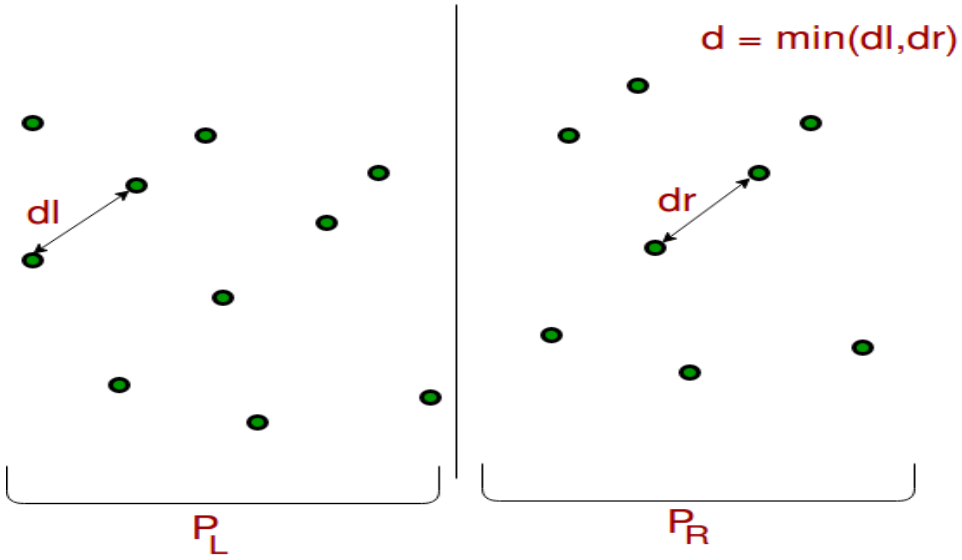
Abstract:

The **closest pair of points problem** or **closest pair problem** is a problem of computational geometry given points in metric space, finding a pair of points with the smallest distance between them. The closest pair problem for points in the Euclidean plane was among the first geometric problems that were treated at the origins of the systematic study of the computational complexity of geometric algorithms.

This paper compares the performance of sequential and parallel methods of the closest pair of points using the Divide and Conquer algorithm.

Introduction:

We are given n points in the plane, and the problem is to find out the closest pair of points in the array. This problem arises in a number of applications. For example, in air-traffic control, you



may want to monitor planes that come too close together, since this may indicate a possible collision. Recall the following formula for distance between two points p and q.

The Brute force solution is O(n^2), compute the distance between each pair and return the smallest. We can calculate the smallest distance in O(nLogn) time using Divide and Conquer strategy. In this post, an O(n x (Logn)^2) approach is discussed. We will be discussing an O(nLogn) approach in a separate post.

This problem can be solved parallelly and if we do so the time would drastically reduce which would eventually enhance the performance.

Method:

1. Serial code

```
from math import sqrt, pow
import time

def distance(a, b):
  return sqrt(pow(a[0] - b[0], 2) + pow(a[1] - b[1], 2))


def bruteMin(points, current=float("inf")):
  if len(points) < 2: return current
  else:
    head = points[0]
    del points[0]
    newMin = min([distance(head, x) for x in points])
    newCurrent = min([newMin, current])
    return bruteMin(points, newCurrent)

def divideMin(points):
  half = len(sorted(points))/2
  minimum = min([bruteMin(points[:half]), bruteMin(points[half:])])
  nearLine = filter(lambda x: x[0] > half -
                minimum and x[0] < half + minimum, points)
  return min([bruteMin(nearLine), minimum])
```

```
list1 = []
counter = 1500

with open("testcases.txt", "a+") as rfile:
    data = rfile.readlines()

    for i in data[0:counter]:
        aa = i.split(",")
        list1.append((int(aa[0][1:]), int(aa[1][:-2])))

start = time.time()
print("MINIMUM CARTESIAN DISTANCE IS ", divideMin(list1))
end = time.time()
TIME = (end - start)
print("TIME TAKEN TO EXECUTE THE CODE IS ", TIME)
```

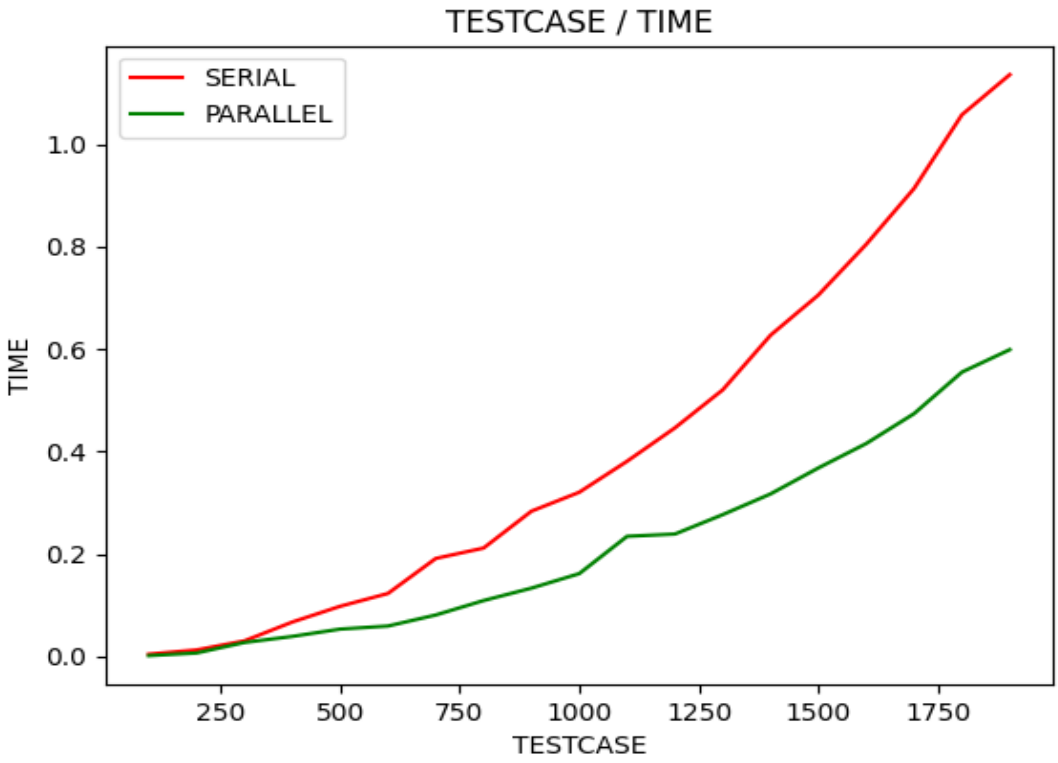
2. Parallel Code:

```
from math import sqrt, pow
import Queue
from threading import Thread
import time

def distance(a, b):
  return sqrt(pow(a[0] - b[0],2) + pow(a[1] - b[1],2))



que = Queue.Queue()


def bruteMin(points, current=float("inf")):
  if len(points) < 2: return current
  else:
    head = points[0]
    del points[0]
    newMin = min([distance(head, x) for x in points])
    newCurrent = min([newMin, current])
    return bruteMin(points, newCurrent)
```

```
def divideMin(points):
  half = len(sorted(points))/2
  a = Thread(target=lambda q, arg1: q.put(bruteMin(arg1)), args=(que, points[:half]))
  b = Thread(target=lambda q, arg1: q.put(bruteMin(arg1)), args=(que, points[half:]))
  a.start()
  b.start()
  a.join()
  b.join()

  a = que.get()
  b = que.get()
  minimum = min([a, b])
  nearLine = filter(lambda x: x[0] > half - minimum and x[0] < half + minimum, points)
  return min([bruteMin(nearLine), minimum])


list1 = []
counter = 1500
counter = intput()
with open("testcases.txt", "a+") as rfile:
    data = rfile.readlines()

    for i in data[0:counter]:
        aa = i.split(",")
        list1.append((int(aa[0][1:]), int(aa[1][:-2])))

start = time.time()
print("MINIMUM CARTESIAN DISTANCE IS ", divideMin(list1))
end = time.time()
TIME = (end - start)
print("TIME TAKEN TO EXECUTE THE CODE IS ", TIME
```

Results:

Graphs between the number of test cases vs Time taken



Output for Serial code and parallel code:

```
jhabarsingh@kmaster:~/mca$ python2 serial.py
('MINIMUM CARTESIAN DISTANCE IS ', 109.11003620199197)
('TIME TAKEN TO EXECUTE THE CODE IS ', 0.6534159183502197)
jhabarsingh@kmaster:~/mca$ python2 parallel.py
('MINIMUM CARTESIAN DISTANCE IS ', 109.11003620199197)
('TIME TAKEN TO EXECUTE THE CODE IS ', 0.36682701110839844)
jhabarsingh@kmaster:~/mca$
```

Conclusion

In this paper, we have explored parallelization as a solution to optimize and make the closest pair of points problem faster. We have used multiprocessing along with shared memory for inter-process communication as a means to achieve parallelization. The paper compares the runtimes of the different methods serially and parallelly, and from the plots, it is evident that indeed, there is a speedup in the operations. The idea of using an MPI to enhance such operations is also an idea that is worth being pursued.