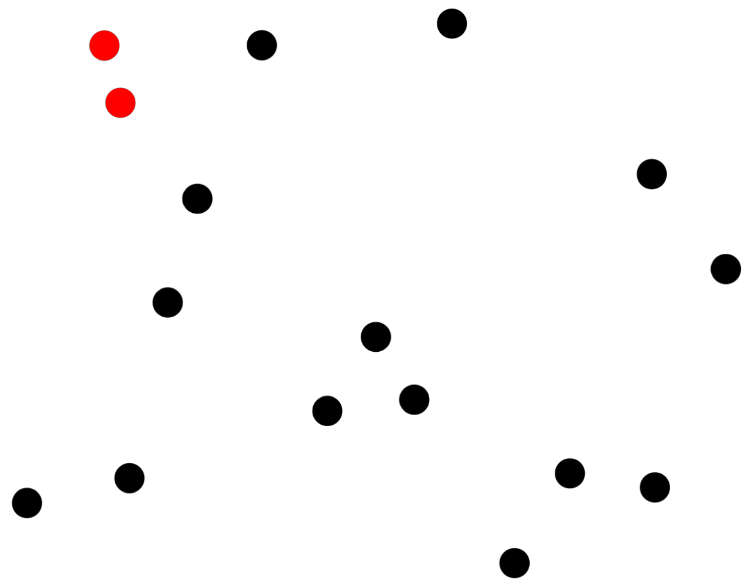
Shortest pair point algorithm

Yifu

Question Statement

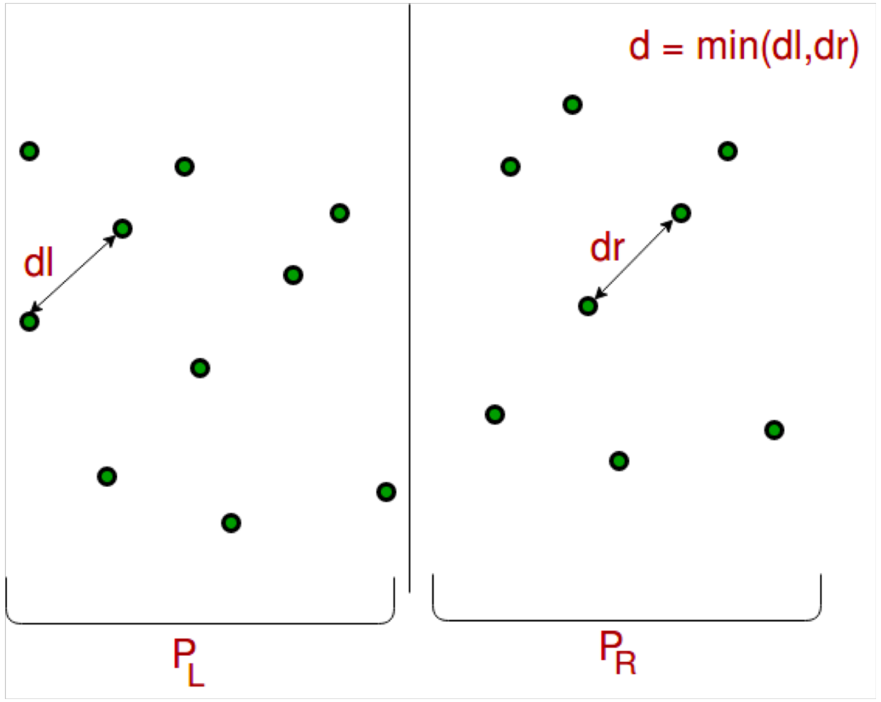
- Input: Sorted points in 2d space by x axis
- Output: position of closest pair of points.



Local / Recursive Sequential Algorithm

- Divide and conquer
- Step 1: divide the points from the middle, until below a constant number.

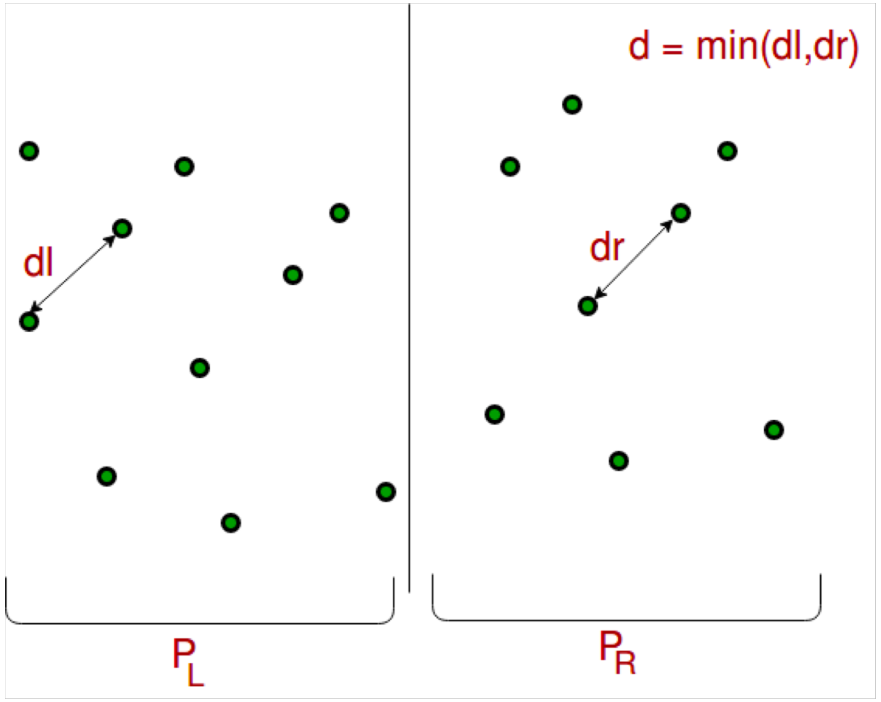
```
struct pair recursive_closest_pair(int start, int end, int min_size){
        if(end-start < min_size){
                return con_size_pair(locs, start, end);
        }
        int ls = start;
        int le = (start+end)/2;
        int rs = (start+end)/2;
        int re = end;

        struct pair lcp = recursive_closest_pair(ls, le, min_size);
        struct pair rcp = recursive_closest_pair(rs, re, min_size);
```



Local / Sequential Algorithm

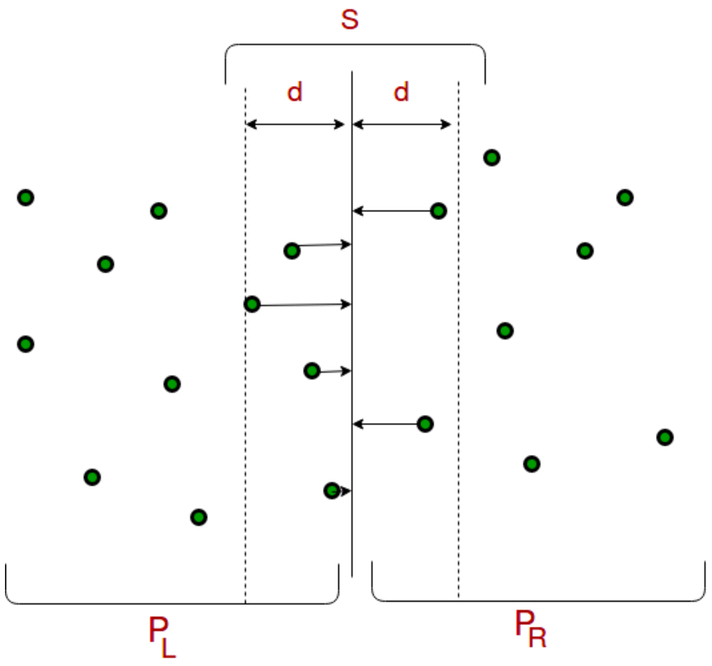
- Step 2: Calculate closest pair using constant time operation on constant size division.

```
struct pair con_size_pair(struct loc* locations, int start, int end){
        int i, j;
        float min_dist = FLT_MAX;
        struct pair min_pair;
        for(i = start; i < end; i++){
                for(j = i+1; j < end; j++){
                        float dist = distance(locations, i, j);
                        if(dist < min_dist){
                                min_dist = dist;
                                min_pair.a = i;
                                min_pair.b = j;
                        }
                }
        }
        return min_pair;
}
```



Local / Sequential Algorithm

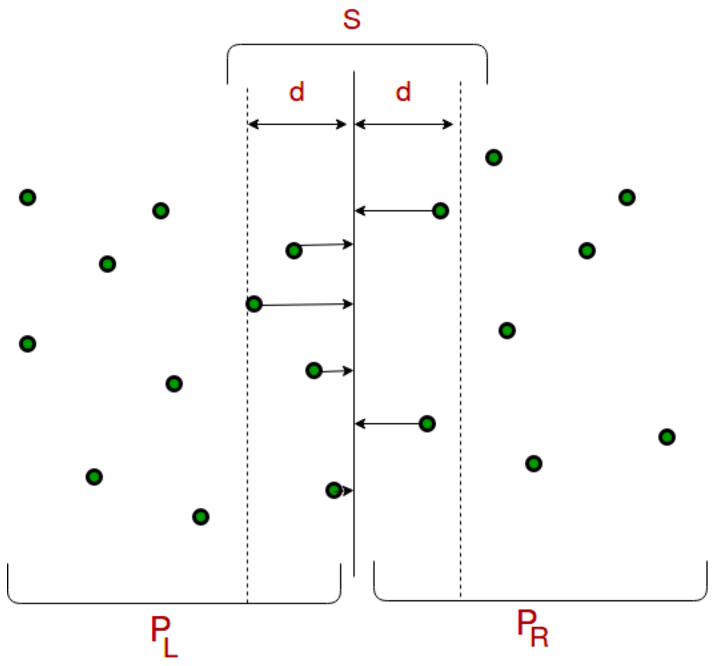
- Step 3 Merging: Get inputs and outputs from both sides, Find minimal distance from both side, get array of points in the middle strip.




```
float mf = lf < rf ? lf : rf;
float mp = (locs[le-1].x + locs[le].x)/2;

int up = ubf(mp, mf, le, end);
int lp = lbf(mp, mf, le, ls);
printf("is lp?mf:%f,  %d, %d, %d\n",mf,  lp, up , le);
struct loc* us = lis( mp, lp, up );
int len = up-lp;
```

Local / Sequential Algorithm

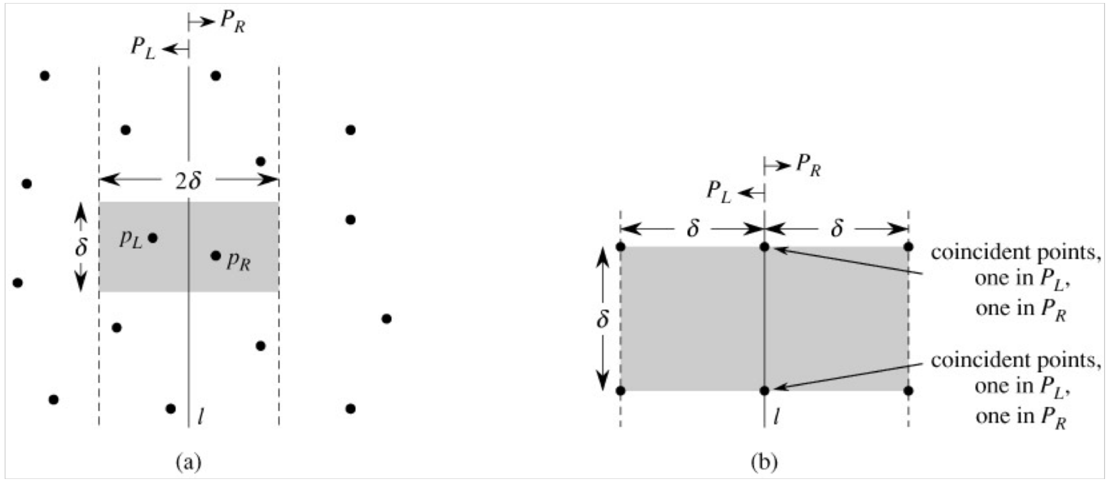
- Step 4 Merging: Sort the middle strip by y position. (O(n log n), can be optimized into O(n))




```
qsort(us,len, sizeof(struct loc), cmpfunc );
```

Local / Sequential Algorithm

- Step 5: since there can not be over 6 points in the same box, and any points outside of that box would have longer distance, we can find shortest pair in this sorted strip in O(n) time by comparing each point to its next 6 neighbor.




```c
struct pair lowpair(struct loc* sot, int size){
        int i, j;
        float min_dist = FLT_MAX;
        struct pair min_pair;
        for(i = 0; i < size; i++){
                for(j = i+1; j< i+8 && j < size; j++ ){
                        float dist = distance(sot, i, j);
                        if(dist < min_dist){
                                min_dist = dist;
                                min_pair.a = i;
                                min_pair.b = j;
                        }
                }
        }

        return min_pair;
}
```

Local / Sequential Algorithm

- Step 6: Return the closest pair from left, right or middle region recursively.

Division of tasks

- Use python to generate sorted input, x will be in order of index, y will be totally random
- Every point have minimum distance of 1, Move 2 points closer than 1 to “generate” correct answer.

```
./
gen.py*
gen.sh*
input_0.txt
input_1.txt
input_10.txt
input_11.txt
input_12.txt
input_13.txt
input_14.txt
input_15.txt
input_16.txt
input_17.txt
input_18.txt
input_19.txt
input_2.txt
input_20.txt
input_21.txt
input_22.txt
input_23.txt
input_3.txt
input_4.txt
input_5.txt
input_6.txt
input_7.txt
input_8.txt
input_9.txt
```

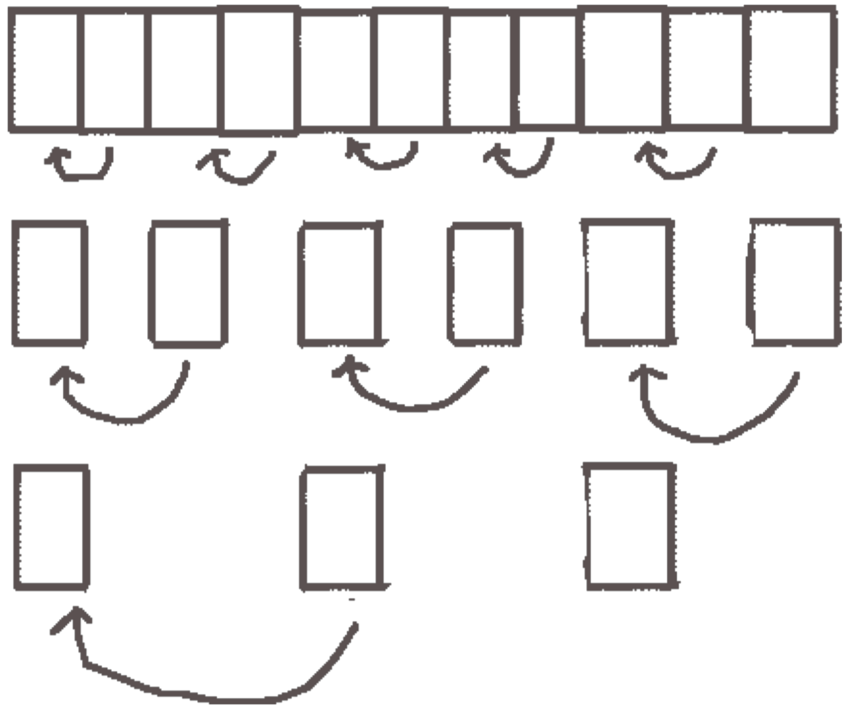
```
100
0 -735.472838717
1 1426.89413444
2 -858.55315694
3 -1727.60874467
4 -858.268614116
5 -2072.4327035
6 -754.281324355
7 -1806.63745023
8 1687.23285464
9 -2167.56727578
10 -1579.55393503
11 1051.74547553
12 -639.658544059
13 -674.63842776
14 517.368835594
15 -1046.79182928
16 -1488.50886725
17 -1310.44128673
18 8.8178874194
19 623.966411584
20 -1798.18061115
21 742.752886484
22 299.996878365
23 -1640.43940762
24 -1230.42027812
25 1911.10231047
26 -1554.20568801
27 1591.23758032
28 373.644496392
29 -2239.66325488
30 912.016726096
31 -2340.64741673
32 -969.540063154
33 1111.31440013
34 2339.76323991
35 1176.90275803
36 544.336727379
37 267.358131401
38 -2094.58904343
39 -915.010628232
40 974.896661714
41 -2394.41214556
42 2000.63984657
```

Parallel Algorithm

- We can partition data into n files, run sequential algorithm on n cores, and merge it using MPI to send the closest pair and middle half strip to its neighbor cores.
- Number of tasks is

currently limited to

Power of 2.



Parallel Algorithm

- Algorithm uses a variable global_ranking_identifier on each core to identify which round. This variable multiply by two each time and loop will end when the number equal to number of cores
- Following code is used to determine which core get to send and receive.

```
  if(myid % global_ranking_identifier == 0){

  }else if((myid - global_ranking_identifier/2) % global_ranking_identifier == 0){
```

- Everytime "odd number" nodes send the front package and back package to the "even number" nodes. Front and end package size is corresponding to stripe of minimum size

Parallel Algorithm

- After front package is send to the ”even number” node, it combines with back package from ”even number” node to form a middle stripe.
- Then the middle stripe is used to find smallest pair in between.
- ”even number” node saved the back package and prepare to send it or combine it in the future.

```
int dest = (myid - global_ranking_identifier/2);
struct loc* pkg = make_send_pkg(loc1, loc2);
MPI_Send ( pkg , 2 , LocsType , dest ,11 ,MPI_COMM_WORLD);

//printf("sending0: %d, %d\n", myid, pkg[0]);
MPI_Send ( front_pack , totals_front , LocsType, dest ,11 ,MPI_COMM_WORLD);
//printf("sending1: %d, %d\n", myid, totals_front );
MPI_Send ( back_pack , totals_back , LocsType, dest ,11 ,MPI_COMM_WORLD);
//printf("sending2: %d, %d\n",myid, totals_back );
```

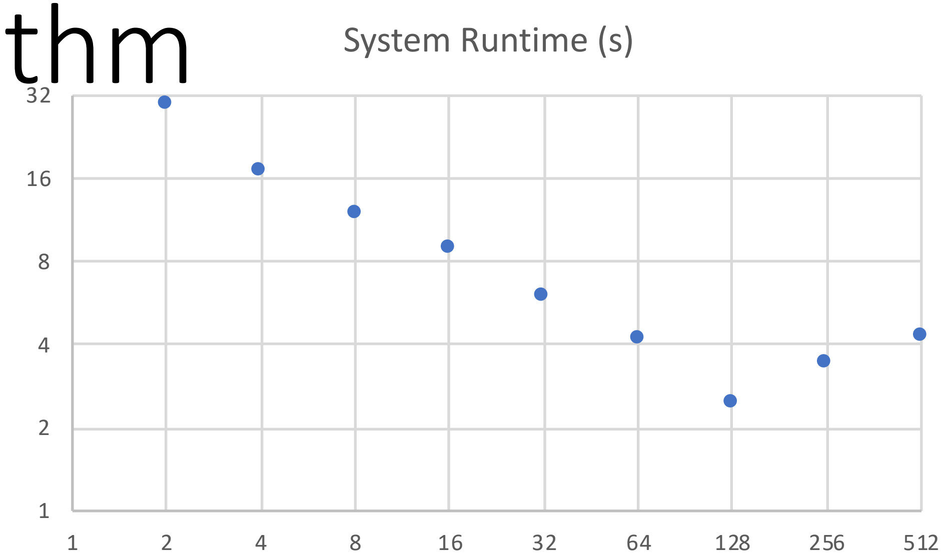
Running on slurm.

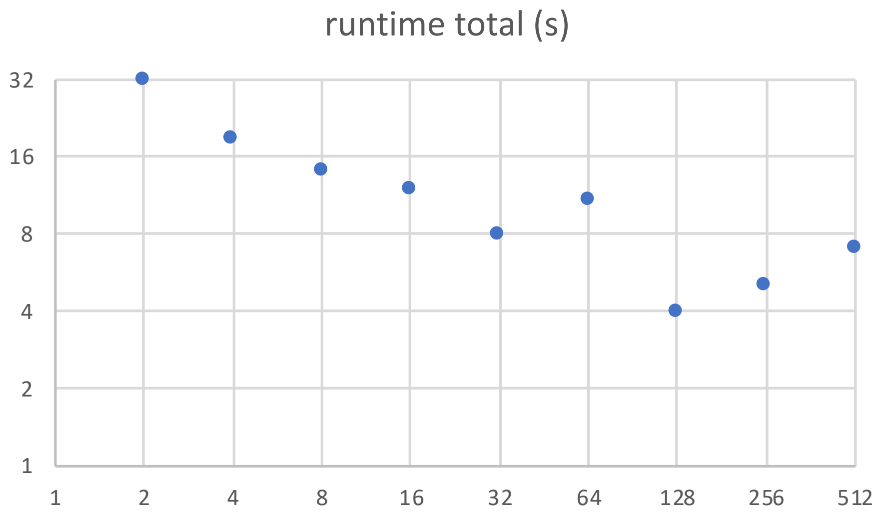
- Increase number of ntasks-per-node first, then increase number of nodes.
- Skylake cpu xeon gold 6130
  - 16 cores, 32 threads
  - 2.10 GHz
- Use Two timing mechanisms, srun time from /usr/bin/time, and total time returned from CCR-email.

```
#!/bin/bash
#SBATCH --nodes=16
#SBATCH --ntasks-per-node=32
#SBATCH --cpus-per-task=1
#SBATCH --exclusive
#SBATCH --constraint=CPU-Gold-6130
#SBATCH --partition=skylake
#SBATCH --qos=skylake
#SBATCH --time=00:30:00
#SBATCH --mail-type=END
#SBATCH --mail-user=yifuyin@buffalo.edu
#SBATCH --output=slurmQ.out
#SBATCH --job-name=omp
#SBATCH --mem=48000
find . -name "core*" -delete
module load intel intel-mpi
export I_MPI_PMI_LIBRARY=/usr/lib64/libpmi.so
mpiicc -o testing.impi testing.c
/usr/bin/time srun ./testing.impi
```

Runtime for Parallel algorithm

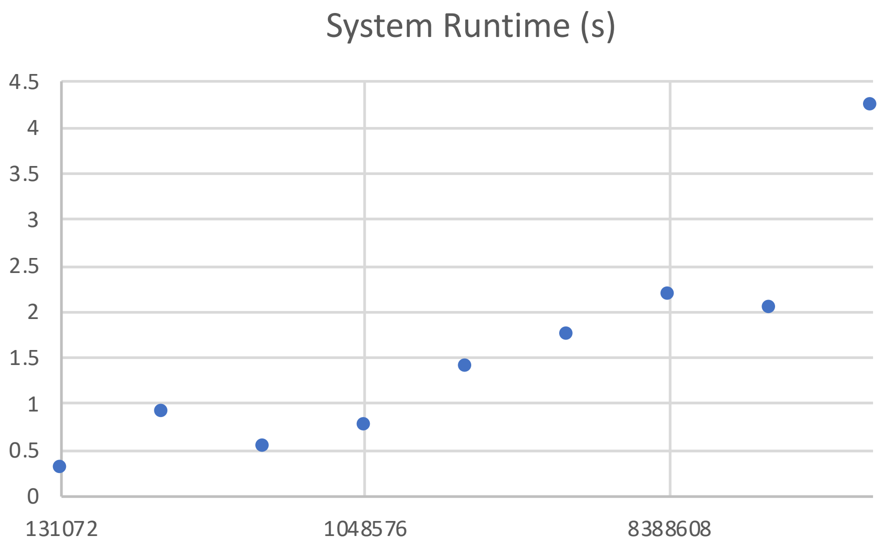
- Total Data Points: 33 million 554 thousands 432
- Split data points into total nodes number of files.
- Generate new dataset each run.
- Measured only one run per task.
- Conclusion: Exponential increase in nodes leads to exponential increase in performance until 128 nodes.

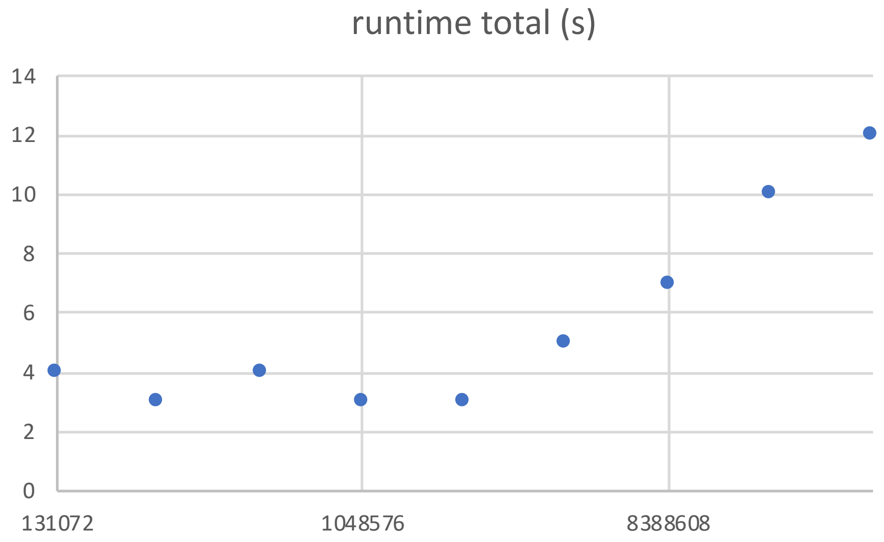




Parallel runtime (increase data points and nodes)

- Increase data points and nodes by 2x every measure.
- Was not able to get 200 million data points due to disk size.
- Measured multiple times and take the mode.
- Used same dataset.
- Generally shows linear increase.





End of slides

- Thank you