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.

```c
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);
    return min(lcp, rcp);
}
```
Local / Sequential Algorithm

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

```c
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.

```c
float mf = 1f < r ? 1f : r;
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 %d%f: %d, %d, %d", 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))

```c
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.
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.

```c
if(global_ranking_identifier < #procs) {
    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.

```c
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.
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