CMSC 641, Design and Analysis of Algorithms, Spring 2010

Sandipan Dey,
Homework Assignment - 12

May 14, 2010

1. Saving Space in Parallel Matrix Multiply

a.

```
P-MATRIX-MULTIPLY-REC-NOTMP(C, A, B)
n = A.rows
if n == 1
    c11 = c11 + a11 b11          { Add with previous result}
else
    partition A, B, C into n/2 × n/2 submatrices
        A11, A12, A21, A22; B11, B12, B21, B22; C11, C12, C21, C22 respectively
    spawn P-MATRIX-MULTIPLY-REC-NOTMP(C11, A11, B11)
    spawn P-MATRIX-MULTIPLY-REC-NOTMP(C12, A11, B12)
    spawn P-MATRIX-MULTIPLY-REC-NOTMP(C21, A21, B11)
    P-MATRIX-MULTIPLY-REC-NOTMP(C22, A21, B12)
    sync
    spawn P-MATRIX-MULTIPLY-REC-NOTMP(C11, A12, B21)
    spawn P-MATRIX-MULTIPLY-REC-NOTMP(C12, A12, B22)
    spawn P-MATRIX-MULTIPLY-REC-NOTMP(C21, A22, B21)
    P-MATRIX-MULTIPLY-REC-NOTMP(C22, A22, B22)
    sync
    parallel for i = 1 to n
        parallel for j = 1 to n
            c_ij = c_ij + t_ij
```

b. Thus, the recurrence for the work $M_1(n)$ is

$M_1(n) \;=\; 8M_1(n/2) + \Theta(n^2) = \Theta(n^3)$

Hence, the recurrence for the span $M_\infty(n)$ of P-MATRIX-MULTIPLY-REC-**NOTMP** is

$M_\infty(n) = M_\infty(n/2) + \Theta(1) = \Theta(n)$

c. Hence, parallelism $M_1(n)/M_\infty(n) \;=\; \Theta(n^3/n) = \Theta(n^2)$

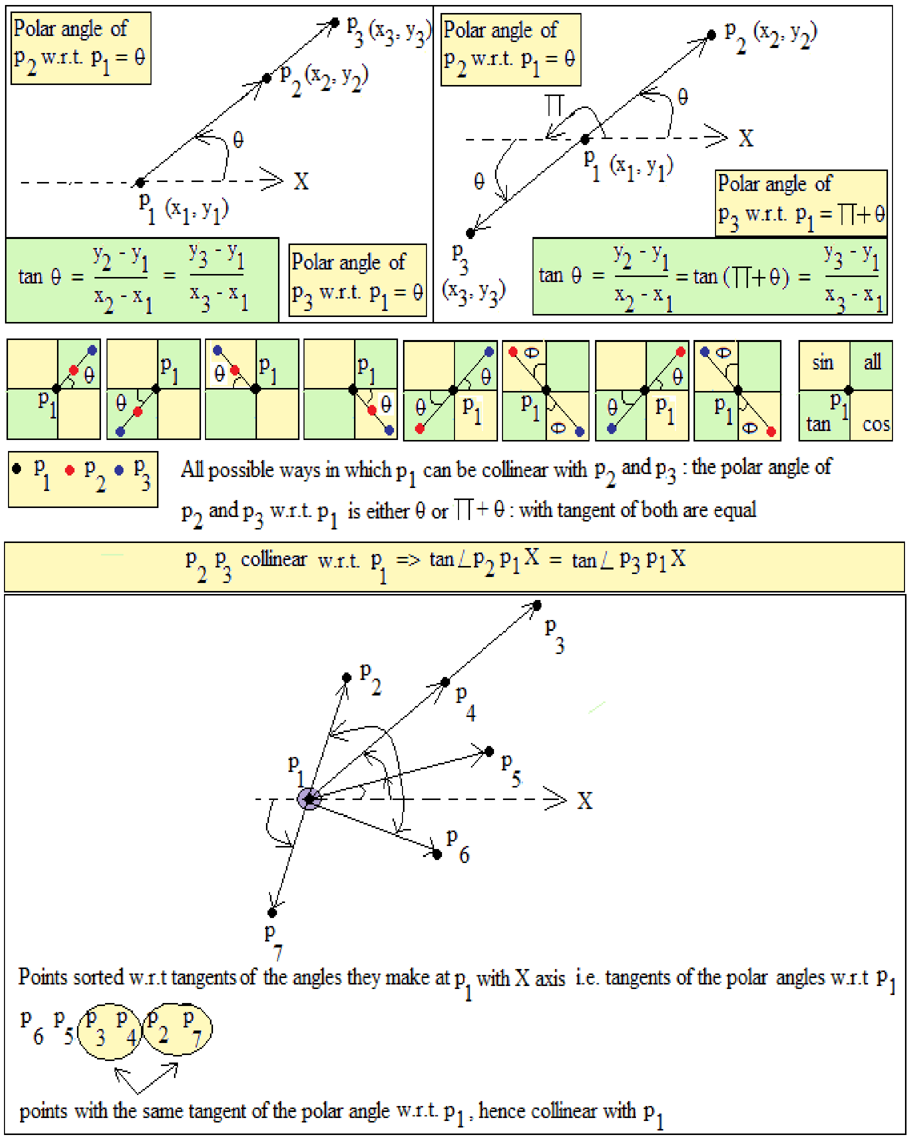
Hence, ignoring the constants, in the Θ-notation,

the parallelism for multiplying 1000×1000 matrices comes to $1000^2 = 10^6$, for P-MATRIX-MULTIPLY-REC-**NOTMP**.

the parallelism for multiplying 1000×1000 matrices comes to approximately $1000^3/10^2 \;=\; 10^7$, for P-MATRIX-MULTIPLY-RECURSIVE

2. Collinear Points

First we observe that two points p_2, p_3 are collinear with another point p_1 iff the tangent of the polar angles of p_2 and p_3 w.r.t. p_1 are equal, as explained in the following figure.



Hence, we propose the following algorithm, in order to find whether any three points in a set of n points $P = \{p_1, p_2, \ldots, p_n\}$ is collinear.

Algorithm

Algorithm 1 Find whether any three points in a set of n points $P = \{p_1, p_2, \ldots, p_n\}$ is collinear

---

1: **for** $i = 1$ to $n,\ p_i \in P$ **do**

2: Compute the polar tangent $tan(\theta_{ij})$ of every other point $p_j \in P,\ j \neq i$ with respect to p_i by $tan(\theta_{ij}) = \begin{cases} \frac{y_j - y_i}{x_j - x_i} & x_j \neq x_i \\ \infty & \text{otherwise} \end{cases}$

3: Sort the points $p_j \in P,\ j \neq i$ w.r.t. their polar tangents $tan(\theta_{ij})$, relative to p_i.

4: Check if any two adjacent points $p_k,\ p_l \neq p_i$ in the sorted order have the same polar tangent w.r.t. p_i, then they are collinear. (since $tan(\theta_{ik}) = tan(\theta_{il}) \Rightarrow p_i,\ p_k,\ p_l$ are collinear, see figure).

5: **end for**

Analysis

- Step 2 is $O(1)$, step 3 is $O(nlgn)$ and step 4 is $O(n)$.
- The total running time for step $2-4$ is $O(1)+O(nlgn)+O(n) = O(nlgn)$.
- The whole process (for loop $1-5$) is repreated for $O(n)$ points.
- Hence total running time of the algorithm $= O(n^2lgn)$.

3. Plane Sweep

In order to find whether there exists an intersection point in between any two disks in a set of n disks in a set, we just modify the plane sweep algorithm for line segments in the following manner:

1. For each disk (represented by $(C_i, r_i) \equiv (x_i, y_i, r_i)$), we find the top point $y_i + r_i$ (red) and bottom point $y_i - r_i$ (blue), these points will be the event points (total $2n$ of them).
2. Sort the disks (in increasing order) w.r.t. their top points ($O(nlgn)$).
3. Start plane sweep (with horizontal sweep lines, keep moving downwords) from above all the disks considering the disks in sorted order w.r.t. their top points.
4. Whenever a new top point is encountered insert the point in the underlying balanced red black tree data structure, presenting the sweep line status.
5. Whenever a bottom point is encountered, delete the point from the underlying balanced red black tree data structure.

6. We observe that if a disk intersects properly with another disk (one not contained entirely inside the other one), we must have the intersection point(s) with the other disk as neighbors at least once in the total ordering represented by the sweep line status. Hence, whenever a new top point is inserted, it's enough to check the immediate left and right disk for intersection. If they intersect then return true.

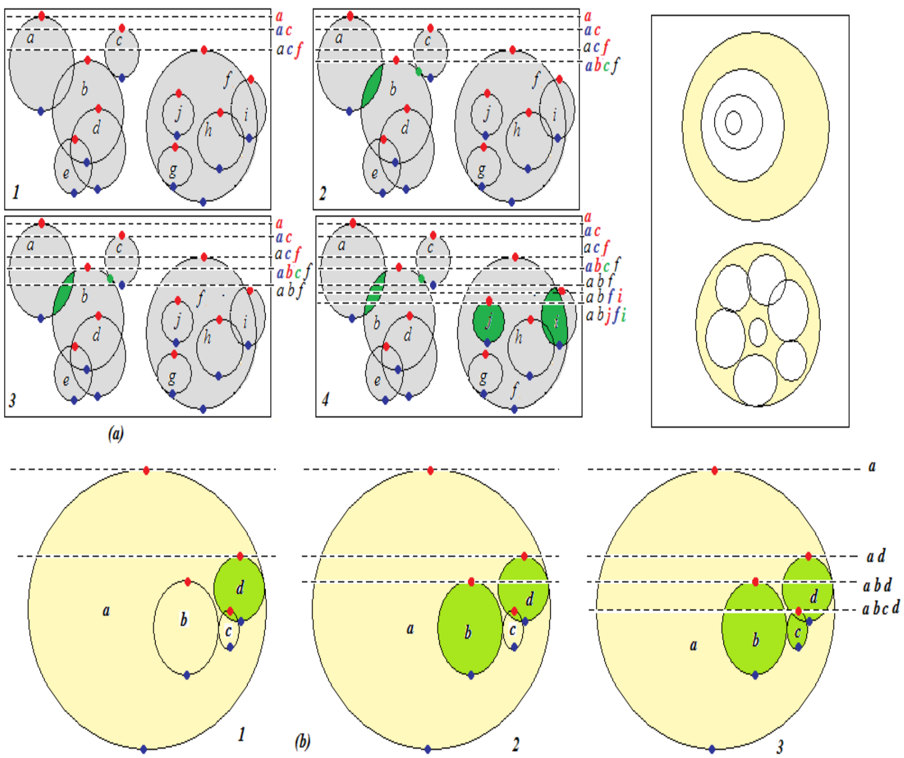
7. Check for intersection: two disks (C_i, r_i), (C_j, r_j) intersect iff $d(C_i, C_j) \leq r_i + r_j$, which is $O(1)$.

8. However, if a disk in contained inside the another one entirely, the monotone property may not be true in general, i.e., the intersecting disks may be non-adjacent in all the sweep line status (e.g., in the following figure (b) the disk c is contained inside disk a, but in none of the sweep line status total orderings a and c are adjacent).

9. Correctness: since we need to find whether any two of the disks intersect or not, the above algorithm always returns true whenever there is an intersection, otherwise returns false.

10. Sweep line insertion and deletion into balanced tree is $O(lgn)$ and there are $O(n)$ event points (sweep line keeps moving down until all the points are deleted from the underlying tree).

11. Hence the run time of the algorithm is $O(nlgn)$.


a
a c
a c f
a b c f
a b f
a b f i
a b j f i
a d
a b d
a b c d
(a)
(b)