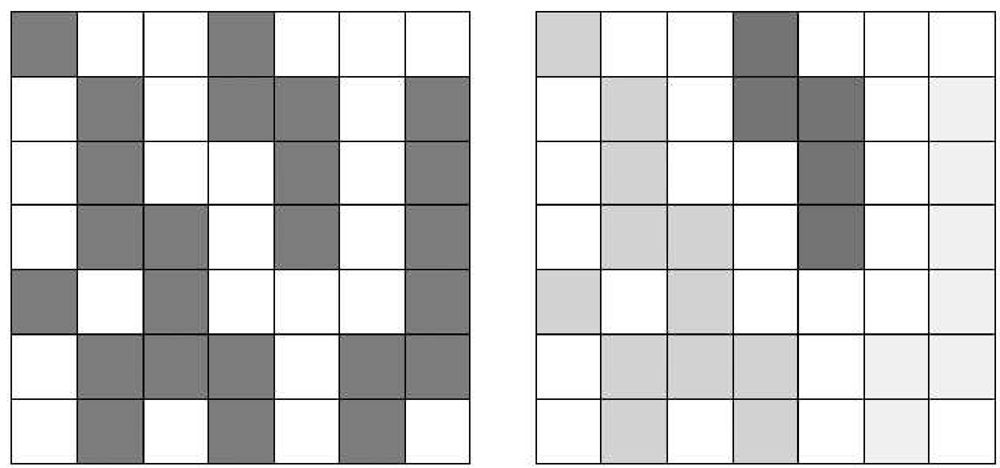
Parallel Algorithms IV

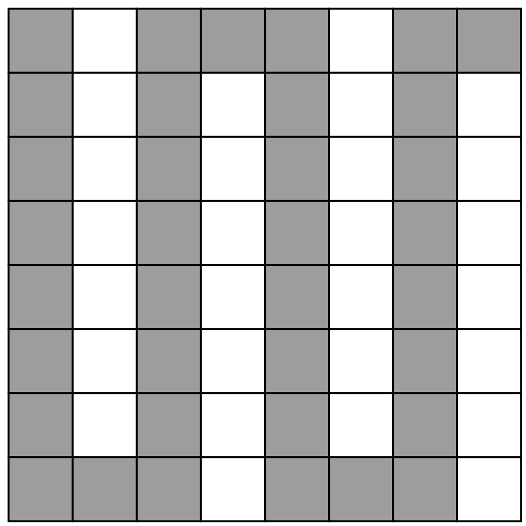
- Topics: image analysis algorithms

Component Labeling

- Given a 2d array of N pixels holding 0 or 1, assign labels to all 1-pixels so that connected pixels have the same label
- Trivial algorithm: assign the co-ordinates of each pixel as its label and repeatedly re-label contiguous pixels by the smaller co-ordinate until no re-labeling occurs
- Execution time: O(N)



Worst-Case Execution Time



Recursive Algorithm

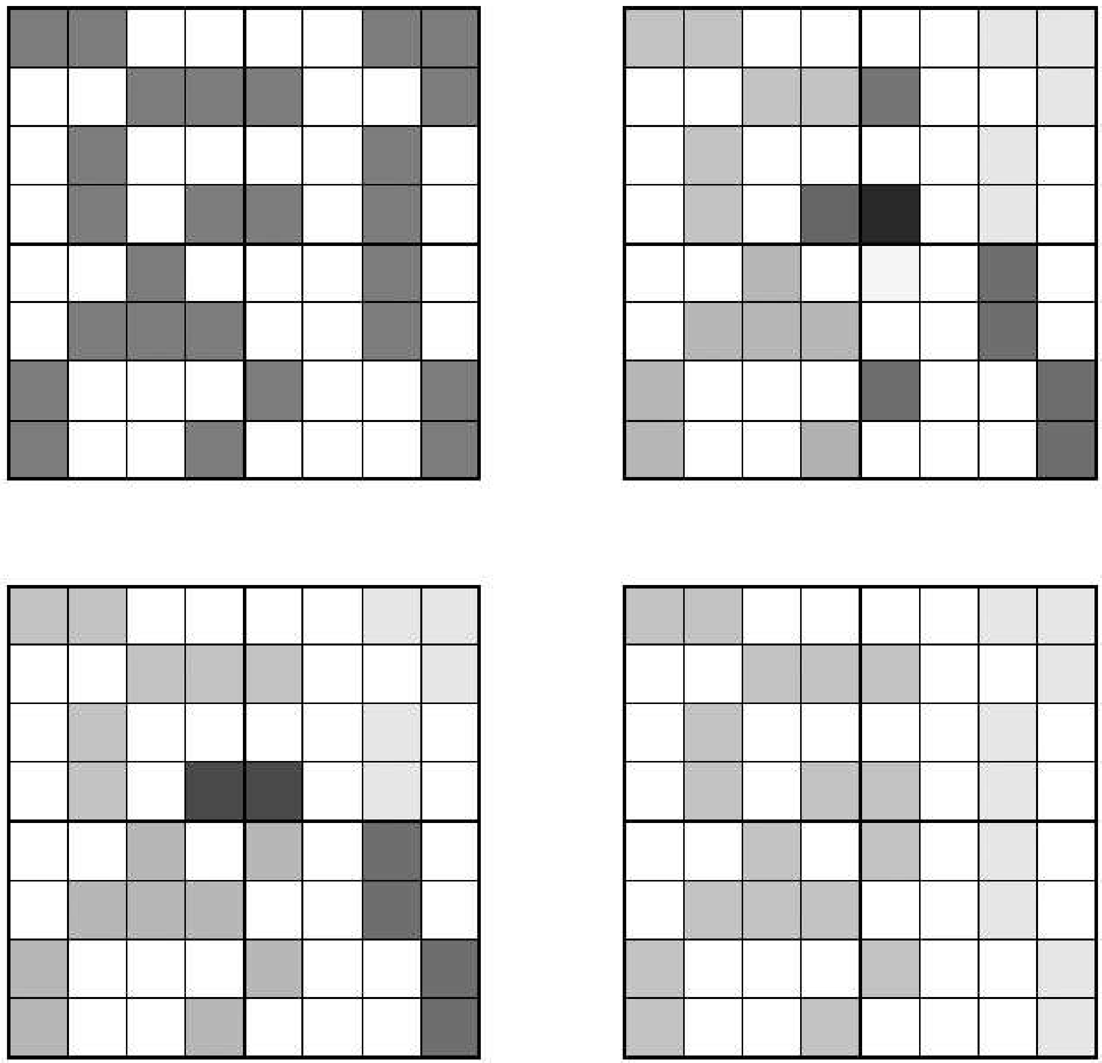
An $O(\sqrt{N})$ step recursive algorithm on a $\sqrt{N} \times \sqrt{N}$ array, where N is a power of 2.

Phase 1 Divide the array into four quadrants and complete labeling within each quadrant (by recursive calls).

Phase 2 *Relabel* by joining horizontally adjacent quadrants.

Phase 3 *Relabel* by joining vertically adjacent quadrants.

Example



Complexity Analysis

- Let $T(m)$ be the run-time of the algorithm on an $m \times m$ array. Let the run-time of phases 2 and 3 be cm

$$T(m) = cm + T(m/2)$$

Therefore, $T(m) = 2cm = O(sqrt(N))$

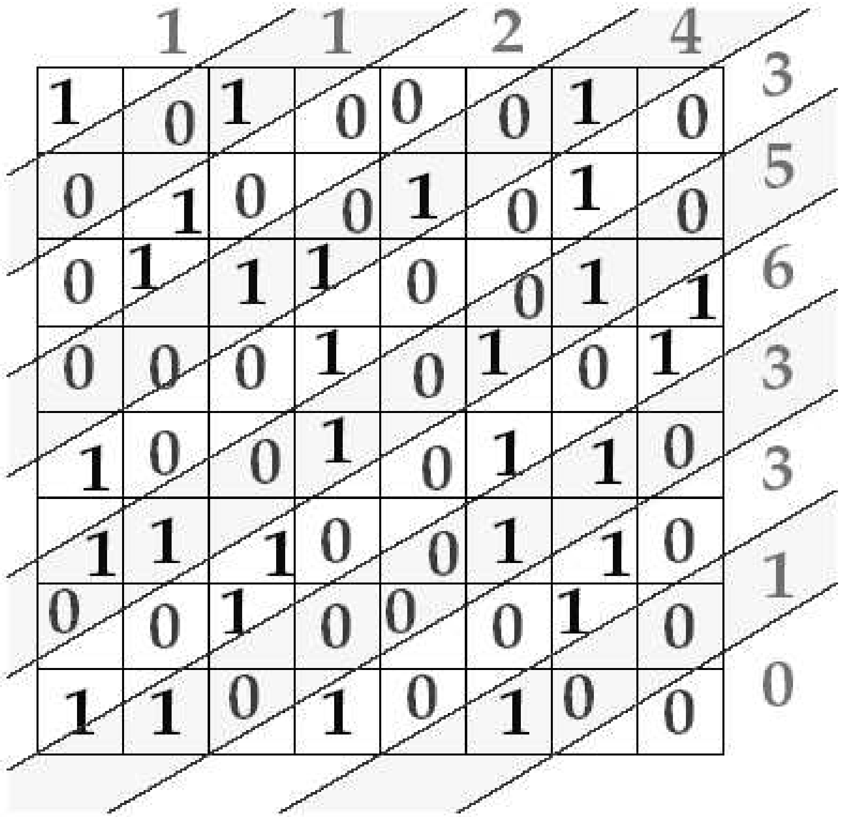
- Executing phases 2 and 3 in O(m) time steps:
  - There are totally m different labels on the boundaries
  - Use the m x m matrix to represent the adjacency matrix for the boundary labels
  - Use transitive closure to compute which labels are reachable from each label
  - The new set of labels is communicated to all pixels in a pipelined manner

Hough Transform

Split the $M \times M$ pixels into bands of 1-pixel width at the angle of θ, where the lower-left corner is on the boundary.

Then for each band count the number of 1 pixels whose center belongs to it.

Example



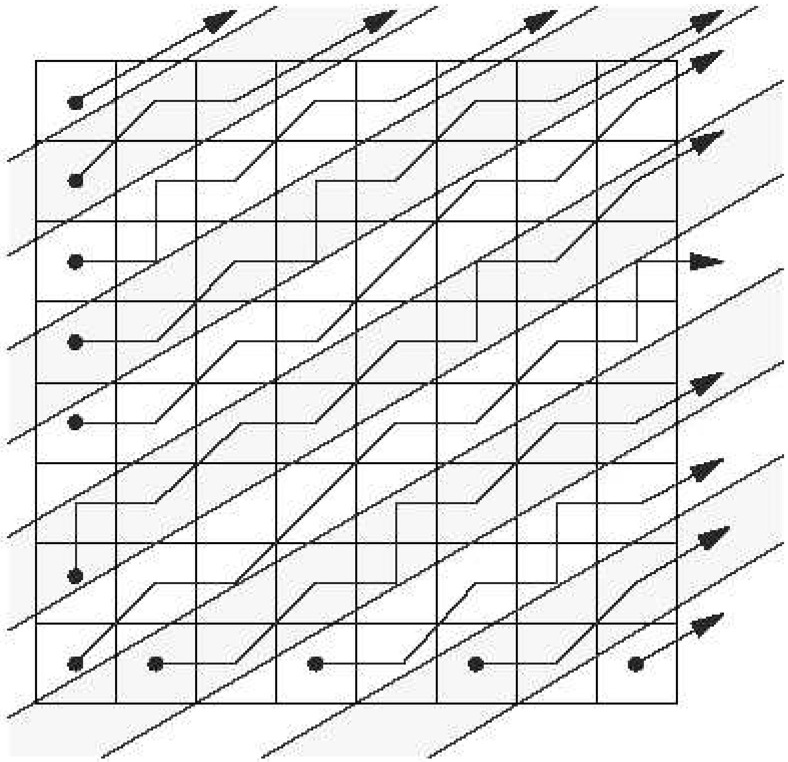
- The width of each band equals the width of a pixel

Algorithm

Assign a counter to each band, and let it travel from the leftmost pixel to the rightmost pixel in the band. When encountering a 1, increment the count. The possible next position is one of the up, the right, or the upper-right contiguous cell. The next position is computable from θ and the locations of the starting cell and the current cell. The running time is $O(M)$.

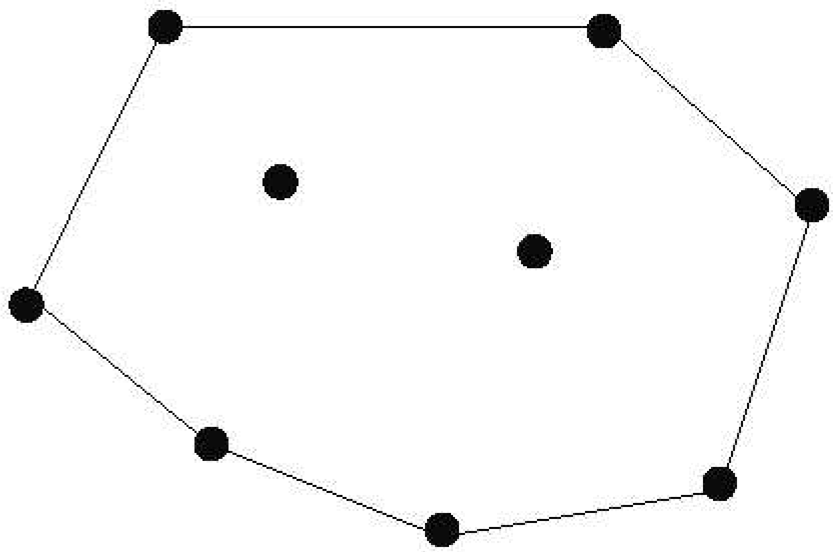
Running the algorithm one after another R times, we can compute the Hough transformation with respect to R angles in $O(R+M)$ steps.

Example



Convex Hull

- For a set of points S in a plane, the convex hull is the smallest convex polygon that contains all points in S



Algorithm on an N-Cell Linear Array

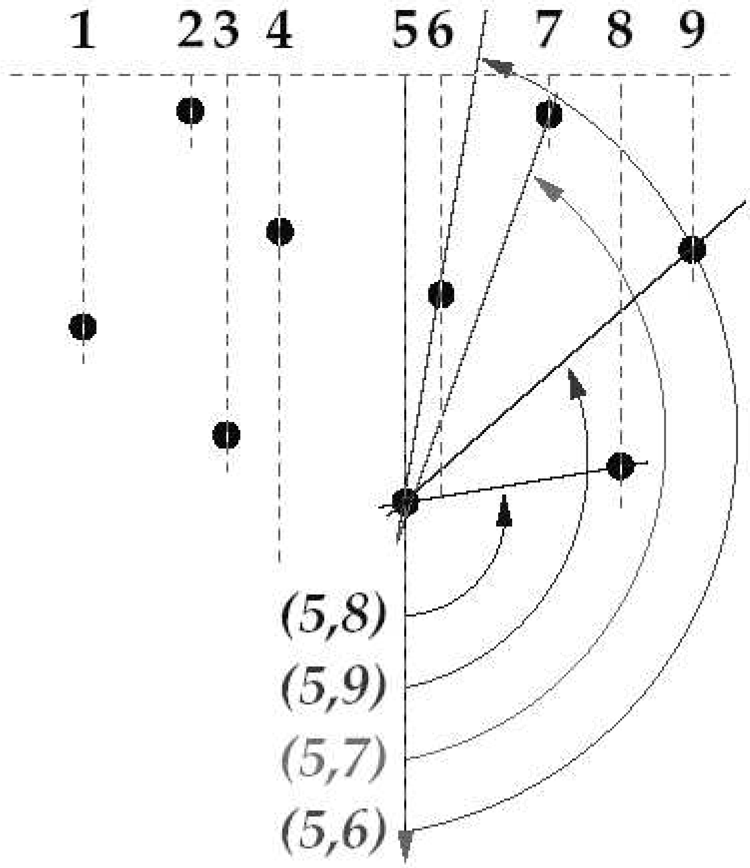
Sort the points, where $(x, y) < (u, v)$ if $x < u$ or $x = u$ and $y < v$. Let $p_i = (x_i, y_i)$ be the point in the ith cell. Clearly, p_1 and p_N belong to the hull.

Then compute the points in the upper hull as well as those in the lower hull.

For each $i, j, 1 \le i < j \le N$, $\theta_{i,j} =_{\text{def}}$ the angle of the line $\overline{p_i p_j}$ with respect to the negative vertical line.

Define $r(i)$ to be the k such that $\theta_{i,k}$ is the largest of all $\theta_{i,j}, i+1 \le j \le N$.

Example for Lower Hull Point



Algorithm Complexity

The algorithm:

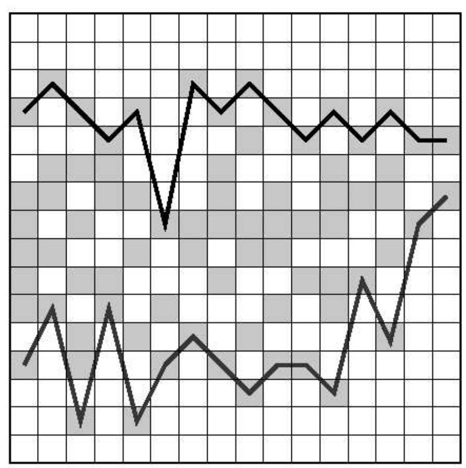
1. Sort the points using odd-even sort.
2. For each i, compute $r(i)$ with the parallel maximum computation.
3. For each i, check whether $(\forall j < i)[r(j) \leq i]$ using the parallel minimum computation.

The running time is $O(N)$.

Reducing Complexity

- If the image is represented by an N x N matrix, we may have as many as N^2 points, leading to $O(N^2)$ complexity for the convex hull computation

- However, for any column (except the right and left ends), only the highest and lowest 1-pixels can be part of the convex hull – by restricting the computation to only these points, the complexity is reduced to O(N)



Title

- Bullet