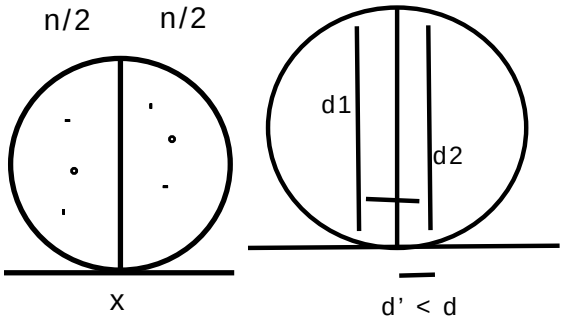
15-499: Parallel Algorithms **Lecturer:** Harsha Simhadri
Topic: Closest Pairs and Fast Fourier Transform **Date:** 3/24/2009
Scribe: Jeffrey Ohlstein

19.1 Closest Pairs

S = $(x_1, y_1), (x_2, y_2)$,...,(x_n, y_n)

O(nlogn) work dig O(n^2) work



d_l = closest-pair(Left)

d_r = closest-pair(Right)

d = min(d_l, d_r)

Must consider points along center line, only the ones within a 2dx2d square. Can only be 6 points within the square that are at least d apart.

W(n) = 2W(n/2) + O(n) = O($n \log n$) D(n) = D(n/2) + log(n) = O($\log^2 n$) Need O(log n) depth to do a pack operation - sort points along y axis.

Basically, need to only check a constant number of points for distance from any given point, gets you the linear time.

19.2 Fast Fourier Transform

a(x) = $\sum_{i=0}^{n-1} a_i x^i$

b(x) = $\sum_{i=0}^{n-1} b_i x^i$

c(x) = a(x)b(x) = $\sum_{i=0}^{2n-2} c_i x^i$

$(a_0 + a_1 x + a_2 x^2 + ...)(b_0 + b_1 x + b_2 x^2 + ...)$

c_i = coeff(x^i) = $\sum_{0 \le j \le n,\ 0 \le i-j \le n} a_j b_{i-j}$ Need to sum n terms for each coefficient of c - O(n^2) work

Discrete Fourier Transform - x $[x_0, x_1, ...x_{n-1}]$ Fourier(x) = $[y_0, y_1, ...y_{n-1}]$ where $y_j = \sum_{k=0}^{n-1} \omega_n^{jk} x_k$, where $\omega_n = e^{\frac{2\pi}{n}}$

Note that the ω_ns represent the n roots of unity of x

Restate formula in terms of a matrix:

$[y_0, y_1, .., y_{n-1}]$ = Matrix($[1, 1, ..]$ $[w_n, (w^2)_n, ..]$ [...] [...] $[...w_n^{jk}]$)

$$\begin{bmatrix} y_0 & y_1 & \cdots & y_{n-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ 1 & \omega_n & \omega_n^2 & \vdots \\ \vdots & \vdots & \vdots & \vdots \\ 1 & \cdots & \cdots & \omega_n^{jk} \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_{n-1} \end{bmatrix}$$

$DFT_{2n}(a) * DFT_{2n}(b) = DFT_{2n}(c)$

So, instead of the algorithm that took O(n^2) where we just compute each coefficient, we can compute the DFT of each set of coefficents, multiply them pairwise, and then inverse DFT the result back in order to get the resulting polynomial. This will take O(nlogn) work instead.

$Fourier_{2n}(a) = \hat{a}$ = $a_0 + \omega_n^{2l} + \omega_n^{4l} + ... + \omega^{2l(n/2)}$ ... = $((a_0 + a_{n/2}) + \omega_n^{2l}(a_1 + a_{n/2+1}) + ...)$

bunch of algebra, basically we can compute the even and odd components separately, using divide and conquer.

$[\hat{a_0}, \hat{a_2}, \hat{a_4}...]$

W(n) = W(n/2) + O(n) = O(n$\log$ n) D(n) = D(n/2) + O(1) = O($\log$ n)

To get back from DFT(c) to c:

$\hat{C} = VC$

$C = V^{-1}\hat{C}$

$V_{jk} = \omega_{2n}^{+jk}$

$V_{jk}^{-1} = \omega_{2n}^{-jk}$