Parallel Techniques

Some common themes in “Thinking Parallel”
1. Working with collections.
   - map, selection, reduce, scan, collect
2. Divide-and-conquer
   - Even more important than sequentially
   - Merging, matrix multiply, FFT, ...
3. Contraction
   - Solve single smaller problem
   - List ranking, graph contraction, Huffman codes
4. Randomization
   - Symmetry breaking and random sampling

Technique 3: Contraction

Consists of:
- Do some work to make a smaller problem
- Solve smaller problem recursively
- Use result to create solution of full problem

The code for scan was based on this, i.e.:
- Pairwise add neighbors in array
- Solve scan that is half as large
- Use results along with original values to generate overall result

Contraction: Graph Connectivity
**Graph Connectivity**

Representing a graph as an edge list:

\[ E = \{(0,1), (0,2), (1,0), (1,3), (1,5), (2,0), (2,3), (3,1), (3,2), (3,4), (3,5), (3,6), (4,3), (4,6), (5,1), (5,3), (5,6), (6,3), (6,4), (6,5)\} \]

Here every edge is represented once in each direction.

Use an array of pointers, one per vertex to point to parent in connected tree. Initially everyone points to self.

\[ L = [0, 1, 2, 3, 4, 5, 6] \] (initially)
\[ L = [1, 1, 1, 2, 1, 0, 0] \] (possible final)

Randomly flip coins

\[ FL = \{\text{coinToss(.5)} : x \in [0:|#L|]\}; \]
\[ FL = [0, 1, 0, 0, 0, 0, 1] \]

Randomly flip coins

\[ H = \{(u,v) \in E \mid \text{not(FL[u]) and FL[v]}\} \]
\[ H = [(0,1), (1,3), (5,1), (3,6), (4,6), (5,6)] \]

Every edge link from black to red
Graph Connectivity

**H** = \[0,1), (3,1), (5,1), (3,6), (4,6), (5,6)\]

\[L = L \leftarrow H\]

\[L = [\frac{1}{2}, 1, 2, \frac{1}{2}, 6, \frac{1}{2}, 6]\]

Randomly flip coins

Every edge link from black to red

"Hook"

Randomly flip coins

Every edge link from black to red

Relabel edges and remove self edges

\[L = [\frac{1}{2}, 1, 2, \frac{1}{2}, 6, \frac{1}{2}, 6]\]

\[E = \{(L[u],L[v]), (u,v) \in E | L[u]=L[v]\}\]

\[E = [(1,2), (2,1), (2,1), (1,2), (1,6), (1,6), (6,1), (1,6), (6,1),(6,1)]\]

Graph Connectivity

L = Vertex Labels,  E = Edge List

function connectivity(L, E) =

if #E = 0 then L
else let
    FL = {coinToss(.5) : x in [0:#L]};
    H = {(u,v) in E | not(FL[u]) and FL[v]};
    L = L \leftarrow H;
    E = {(L[u],L[v]) : (u,v) in E | L[u]=L[v]};
in connectivity(L,E);  D = O(log n)
W = O(m log n)

List Ranking (again)

P = [7, 6, 0, 1, 3, 2, 9, 8, 4, 9]
W = [1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
**List Ranking**

\[ FL = \{\text{coinToss}(0.5) : x \text{ in } [0:\#P]\}; \]
\[ FL = [1, 0, 0, 1, 0, 0, 1, 0, 1, 1] \]

**List Ranking**

\[ D = \{FL[i] \text{ and not(FL[P[i]])} : i \text{ in } [0:\#P]\}; \]
\[ D = [1, 0, 0, 1, 0, 0, 0, 0, 1, 0] \]

**List Ranking**

\[ D = [1, 0, 0, 1, 0, 0, 0, 0, 1, 0] \]

**List Ranking**

\[ NI = \text{plusScan}({\text{not}(x) : x \text{ in } D}); \]
\[ NI = [0, 0, 1, 2, 3, 4, 5, 6, 6] \]

**List Ranking**

\[ NI = [0, 0, 1, 2, 2, 3, 4, 5, 6, 6] \]

if \( D[P[i]] \) then
\[ (W[i] + W[P[i]], NI[P[i]]) \]
else \( (W[i], NI[P[i]]) \)
List Ranking

\[
\begin{align*}
W &= [1, 2, 2, 1, 1, 2, 1] \\
P &= [4, 5, 0, 1, 6, 2, 6]
\end{align*}
\]

\[
\text{start} \\
1 & 2 \quad 2 & 1 & 1 & 1 & 1 & 1
\]

\[
LR = \text{listRank}(W', P');
\]

\[
\begin{align*}
\text{function listRank}(W, P) = \\
\quad & \text{if} \ #P == 1 \ \text{then} \ \{W[0]\} \\
\quad & \text{else let} \\
\quad & \ \quad FL = \{\text{coinToss(.5)} : i \ \text{in} \ [0:\#P]\}; \\
\quad & \ \quad D = \{FL[i] \ \text{and} \ \text{not}(FL[P[i]]) : i \ \text{in} \ [0:\#P]\}; \\
\quad & \ \quad NI = \text{plusScan}(\{\text{not}(x) : x \ \text{in} \ D\}); \\
\quad & \ \quad (W', P') = \text{unzip} \{ \\
\quad & \ \quad \quad \text{if} \ D[P[i]] \ \text{then} \ \{W[i] + W[P[i]], NI[P[P[i]]]\} \\
\quad & \ \quad \quad \quad \quad \text{else} \ \{W[i], NI[P[i]]\} \\
\quad & \ \quad \quad : i \ \text{in} \ [0:\#P] \ \text{and not}(D[i]); \} \\
\quad & \ \quad \ \text{LR} = \text{listRank}(W', P'); \\
\quad & \ \quad \text{in} \{\text{if} \ D[i] \ \text{then} \ \text{LR}[NI[P[i]]] + W[i] \\
\quad & \ \quad \quad \text{else} \ \text{LR}[NI[i]] \\
\quad & \ \quad \quad : i \ \text{in} \ [0:\#P] \};
\end{align*}
\]
Greedy: Huffman Codes

Huffman Algorithm:
Each p in P is a probability and a tree
function Huffman(P) =
  if (#P == 1) then return
  else let
    ((p1,t1),(p2,t2),P') = extract2mins(P)
    pt = (p1+p2, newNode(t1,t2)
    in Huffman(insert(pt,P'))

Example
\[ p(a) = .1, \ p(b) = .2, \ p(c) = .2, \ p(d) = .5 \]
\[ a(.1) \quad b(.2) \quad c(.2) \quad d(.5) \]
\[ a=.1 \quad b=.2 \quad (2) \quad c=.2 \quad (3) \quad d=.5 \]
\[ a=.1 \quad b=.2 \quad (2) \quad c=.2 \quad (3) \quad d=.5 \]
\[ a=.1 \quad b=.2 \quad (2) \quad c=.2 \quad (3) \quad d=.5 \]
\[ a=.1 \quad b=.2 \quad (2) \quad c=.2 \quad (3) \quad d=.5 \]
\[ d(.5) \quad d(.5) \]
\[ a=000, \ b=001, \ c=01, \ d=1 \]

Greedy: Huffman Codes

Huffman Algorithm:
How do we do it in parallel?
Function Huffman(P) =
  if #P == 1 then return
  else let
    ((p1,t1),(p2,t2),P') = extract2mins(P)
    pt = (p1+p2, newNode(t1,t2)
    in Huffman(insert(pt,P'))

Primes Sieve

function primes(n) =
  if n == 2 then [2] int
  else let sqprimes = primes(ceil(sqrt(float(n))));
    sieves = flatten([2*primes[p]: p in sqprimes];
    flags = dist(t,n) <= (i,f): i in sieves;
    in drop({i in [0:n]; flags | flags}, 2) ;

W(n) = \( O(n \log \log n) \)
D(n) = \( D(\sqrt{n}) + O(\log n) \)
\[ = O(\log n) \]
Parallel Techniques

Some common themes in “Thinking Parallel”
1. Working with collections.
   - map, selection, reduce, scan, collect
2. Divide-and-conquer
   - Even more important than sequentially
   - Merging, matrix multiply, FFT, ...
3. Contraction
   - Solve single smaller problem
   - List ranking, graph contraction, Huffman codes
4. Randomization
   - Symmetry breaking and random sampling