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
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.

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

Randomly flip coins

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

Every edge link from black to red

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

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

$L = L \leftarrow H$

$L = [1, 1, 2, 1, 6, 1, 6]$

Randomly flip coins

Every edge link from black to red

"Hook"

E = $\{(L[u],L[v]) : (u,v) \text{ in } E \mid L[u] \neq L[v]\}$

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

Relabel edges and remove self edges

List Ranking (again)

$P = [7, 6, 0, 1, 3, 2, 9, 8, 4, 9]$

$W = [1, 1, 1, 1, 1, 1, 1, 1, 1]$

start
**List Ranking**

`FL = {coinToss(.5) : x in [0:#P]};
FL = [1, 0, 0, 1, 0, 0, 1, 0, 1, 1]`

**List Ranking**

`D = {FL[i] and not(FL[P[i]]) : i 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 = plusScan({not(x) : x in D});
NI = [0, 0, 1, 2, 2, 3, 4, 5, 6, 6]`

**List Ranking**

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

*Add (shortcut)*

*Remove*
List Ranking

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

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

function listRank(W, P) =
if \#P == 1 then [W[0]]
else let
    FL = {\text{coinToss}(0.5) : i in [0:#P]};
    D = {FL[i] and not(FL[P[i]]) : i in [0:#P]};
    NI = plusScan({not(x) : x in D});
    (W', P') = unzip {
        if D[P[i]] then (W[i] + W[P[i]], NI[P[P[i]]])
            else (W[i], NI[P[i]]);  
    : i in [0:#P];
    };
    LR = listRank(W', P');
in {if D[i] then LR[Ni[P[i]]] + W[i]
            else LR[Ni[i]]
        : i in [0:#P];
    }.
**Greedy: Huffman Codes**

**Huffman Algorithm:**
Each \( p \) in \( P \) is a probability and a tree

```java
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)   b(.2)   c(.2)   d(.5)
a(.1)   b(.2)
```

Step 1

```
a(.1)   b(.2)   c(.2)   d(.5)
```

Step 2

```
a(.1)   b(.2)   c(.2)   d(.5)
```

Step 3

```
a(.1)   b(.2)   c(.2)   d(.5)
```

**Primes Sieve**

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

\[ W(n) = O(n \log \log n) \]

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

\[ D(n) = O(\log n) \]
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