Parallelism: Lecture 2
Parallel techniques and algorithms
- Working with collections
- Divide and conquer
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
4. Randomization
   - Symmetry breaking and random sampling
Working with Collections

reduce \( \odot \) [a, b, c, d, ...]
\[= a \odot b \odot c \odot d + ...\]

scan \( \odot \) ident [a, b, c, d, ...]
\[= \text{[ident, a, a} \odot b, a \odot b \odot c, ...\]

sort compF A

collect [(2,a), (0,b), (2,c), (3,d), (0,e), (2,f)]
\[= \text{[(0, [b,e]), (2,[a,c,f]), (3,[d])]}\]

15-853
The parentheses matching problem:
- Check if a set of a single kind of parentheses match
- E.g. ((())()) matches (((())))(()() does not
- Easy to do serially by scanning left to right keeping a counter.
- How do we do this in parallel
Example of scan: parentheses matching

The parentheses matching using a scan:

\[
\text{function parenthesesMatch}(S) = \\
\text{let} \\
A = \{\text{if } c == '(' \text{ then } 1 \text{ else } -1: c \text{ in } S\}; \\
\text{Sums} = \text{scan}(\text{add}, 0, A); \\
in \\
\quad (\text{reduce}(\text{min}, \text{Sums}) >= 0)
\]

Can also do it with a map and reduce, or with recursion.
Example of Collect: Building an Index

Problem: Given a set of documents each a string, compute an index that maps words to documents.

[(1,"this is the first document"),
(2,"this is the second"),
(3,"the third"),
(4,"and the fourth")]

[(("and",[4]),...,("first",[1]),...,("is",[1,2]),..., 
("the",[1,2,3,4]),("this",[1,2]),("third",[3]))]
Example of Collect: Building an Index

Problem: Given a set of documents each with a sequence of words, compute an index that maps words to documents.

function makeIndex D =
    let
        a = flatten({{(w,i) : w in wordify(d)} : (i,d) in D})
    in collect(a);
MapReduce

function mapReduce(MAP, REDUCE, documents) =
    let
        temp = flatten({MAP(d) : d in documents});
    in flatten({REDUCE(k, vs) : (k, vs) in collect(temp)});

function mapRed(M, R) = (D => mapReduce(M, R, D));

wordcount = mapReduce(d => {(w, 1) : w in wordify(d)},
    (w, c) => [(w, sum(c))]);

wordcount(["this is is document 1",
    "this is document 2"]);
Technique 2: Divide-And-Conquer

- Merging
- Matrix multiplication
- Matrix inversion
- FFT
- K-d trees
Example: Merging

\[
\text{Merge}(\text{nil}, l2) = l2
\]
\[
\text{Merge}(l1, \text{nil}) = l1
\]
\[
\text{Merge}(h1::t1, h2::t2) =
\begin{align*}
&\text{if } (h1 < h2) \quad h1::\text{Merge}(t1, h2::t2) \\
&\text{else } \quad h2::\text{Merge}(h1::t1, t2)
\end{align*}
\]

What about in parallel?
The Split Operation

fun split (p, empty) = (empty, empty)
  | split (p, node(v, L, R)) = 
    if p < v then
      let val (L1, R1) = split(p, L)
      in (L1, node(v, R1, R)) end
    else
      let val (L1, R1) = split(p, R)
      in (node(v, L, L1), R1) end;
Merging

\[
\text{Merge}(A, B) = \\
\text{let} \\
\quad \text{Node}(A_L, m, A_R) = A \\
\quad (B_L, B_R) = \text{split}(B, m) \\
\text{in} \\
\quad \text{Node}(\text{Merge}(A_L, B_L), m, \text{Merge}(A_R, B_R))
\]

Span = \(O(\log^2 n)\)
Work = \(O(n)\)

Merge in parallel

Span = \(O(\log^2 n)\)
Work = \(O(n)\)
MergeSort

function mergeSort(S) =
if (#S < 2) S
else merge(mergesort(S[0:#S/2]),
            mergesort(S[#S/2:#S]))

\[ W(n) = 2 \ W(n/2) + O(n) = O(n \log n) \]

What about the span?
Matrix Multiplication

Fun A*B {
    if #A < k then baseCase..
    C_{11} = A_{11} \times B_{11} + A_{12} \times B_{21}
    C_{12} = A_{11} \times B_{12} + A_{12} \times B_{22}
    C_{21} = A_{21} \times B_{11} + A_{22} \times B_{21}
    C_{22} = A_{21} \times B_{12} + A_{22} \times B_{22}
    return C
}

\[
A = \begin{bmatrix}
A_{11} & A_{12} \\
A_{21} & A_{22}
\end{bmatrix}
\]

\[
B = \begin{bmatrix}
B_{11} & B_{12} \\
B_{21} & B_{22}
\end{bmatrix}
\]

\[
W_*(n) = 8W(n/2) + O(n^2)
\]

\[
= O(n^3)
\]

\[
D(n) = D(n/2) + O(1)
\]

\[
= O(log n)
\]

\[
\text{Parallelism} = \frac{W}{D} = O\left(\frac{n^3}{\log n}\right)
\]
Matrix Inversion

fun invert(M) {
    if small baseCase
        D⁻¹ = invert(D)
        S = A - BD⁻¹C
        S⁻¹ = invert(S)
        E = S⁻¹
        F = S⁻¹BD⁻¹
        G = -D⁻¹CS⁻¹
        H = D⁻¹ + D⁻¹CS⁻¹BD⁻¹
    
    \[
    M = \begin{bmatrix}
        A & B \\
        C & D
    \end{bmatrix}
    \]
    \[
    M^{-1} = \begin{bmatrix}
        E & F \\
        G & H
    \end{bmatrix}
    \]

    W(n) = 2W(n/2) + 6W*(n/2)  \quad  D(n) = 2D(n/2) + 6D*(n/2)
    = O(n^3)  \quad \quad  = O(n)

    Parallelism = \frac{W}{D} = O(n^2)

Fourier Transform

function fft(a,w) =
if #a == 1 then a
else
    let r = {fft(b, even_elts(w)):
        b in [even_elts(a),odd_elts(a)]}
in {a + b * w : a in r[0] ++ r[0];
    b in r[1] ++ r[1];
    w in w};

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

Parallelism = \frac{W}{D} = O(n)
Spatial Decompositions: Revisited

Typically consist of:

- Split the data points into some constant number of parts. This is similar to the selection in Quicksort.

- Recursively subdivide within each part.

Both of these are easy to parallelize, but problematic if highly imbalanced.
Callahan-Kosaraju: Build Tree

Function Tree(P)
if |P| = 1 then return leaf(P)
else
    \[ d_{\text{max}} = \text{dimension of } l_{\text{max}} \]
    \[ P_1, P_2 = \text{split } P \text{ along } d_{\text{max}} \text{ at midpoint} \]
    Return Node(Tree(P_1), Tree(P_2), l_{\text{max}})
**KK: Generating the Realization**

```plaintext
function wsr(T)
    if leaf(T) return Ø
    else return wsr(left(T)) ∪ wsr(right(T))
        ∪ wsrP(left(T), right(T))

function wsrP(T₁, T₂)
    if wellSep(T₁, T₂) return {(T₁, T₂)}
    else if l_max(T₁) > l_max(T₂) then
        return wsrP(left(T₁), T₂) ∪ wsrP(right(T₁), T₂)
    else
        return wsrP(T₁, left(T₂)) ∪ wsrP(T₁, right(T₂))
```

15-853
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

4. Randomization
   - Symmetry breaking and random sampling