15-853:Algorithms in the Real World

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, …

= [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)]

= [(0, [b,e]), (2,[a,c,f]), (3,[d])]

Example of scan: parentheses matching

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:

```
function parenthesesMatch(S) =
let
  A = {if c == '( then 1 else -1: c in S};
  Sums = scan(add,0,A);
in
  (reduce(min,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

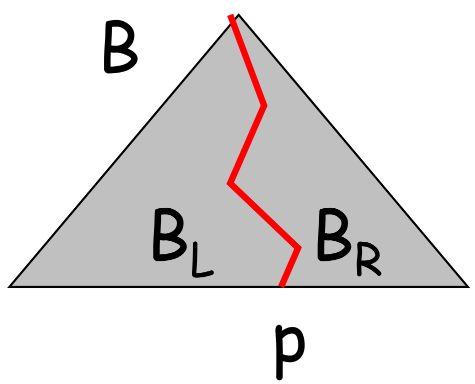
Example: Merging

```
Merge(nil,l2) = l2
Merge(l1,nil) = l1
Merge(h1::t1, h2::t2) =
    if (h1 < h2) h1::Merge(t1,h2::t2)
    else h2::Merge(h1::t1,t2)
```

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;
```

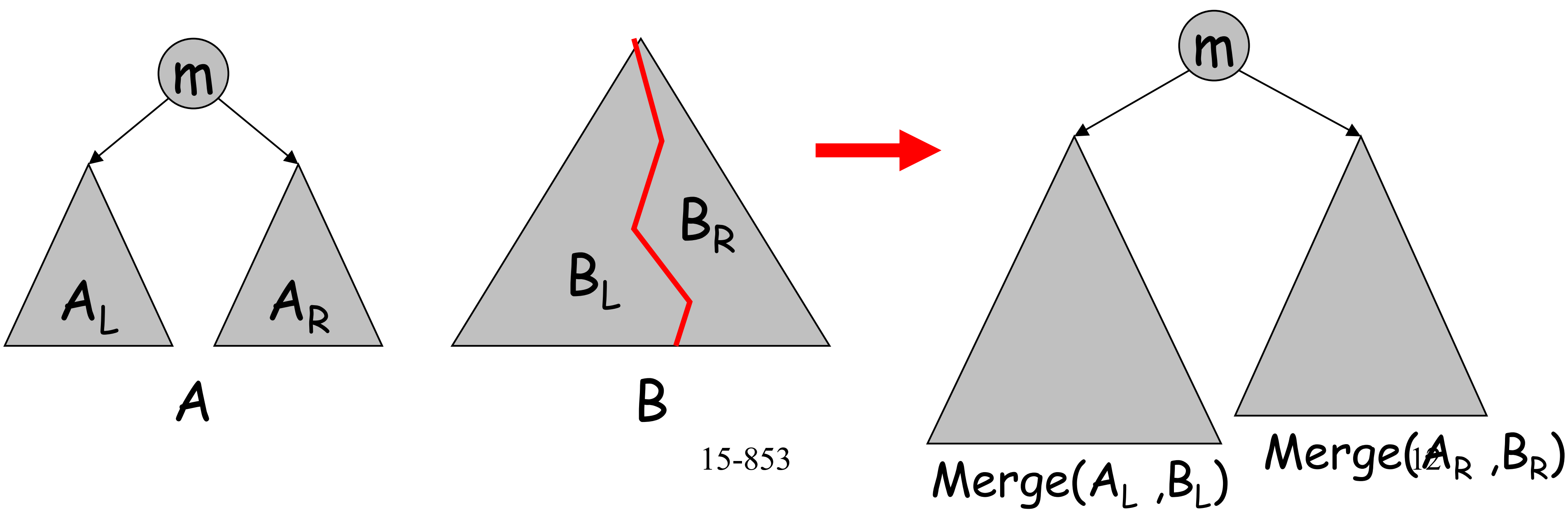


Merging

```
Merge(A,B) =
  let
    Node(A_L, m, A_R) = A
    (B_L ,B_R) = split(B, m)
  in
    Node(Merge(A_L,B_L), m, Merge(A_R,B_R))
```

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

Merge in parallel



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..
  C11 = A11*B11 + A12*B21
  C12 = A11*B12 + A12*B22
  C21 = A21*B11 + A22*B21
  C22 = A21*B12 + A22*B22
  return C
}
```

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

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

$$\begin{aligned} W_*(n) &= 8W(n/2) + O(n^2) \\ &= O(n^3) \end{aligned}$$

$$\begin{aligned} D(n) &= D(n/2) + O(1) \\ &= O(\log n) \end{aligned}$$

$$Parallelism = \frac{W}{D} = O\left(\frac{n^3}{\log n}\right)$$

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}
```

$$M = \begin{bmatrix} A & B \\ C & D \end{bmatrix}$$

$$M^{-1} = \begin{bmatrix} E & F \\ G & H \end{bmatrix}$$

$$\begin{aligned} W(n) &= 2W(n/2) + 6W_*(n/2) \\ &= O(n^3) \end{aligned} \qquad \begin{aligned} D(n) &= 2D(n/2) + 6D_*(n/2) \\ &= O(n) \end{aligned}$$

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

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};
```

$$\begin{aligned} W(n) &= 2W(n/2) + O(n) \\ &= O(n \log n) \end{aligned}$$

$$\begin{aligned} D(n) &= D(n/2) + O(1) \\ &= O(\log n) \end{aligned}$$

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

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)

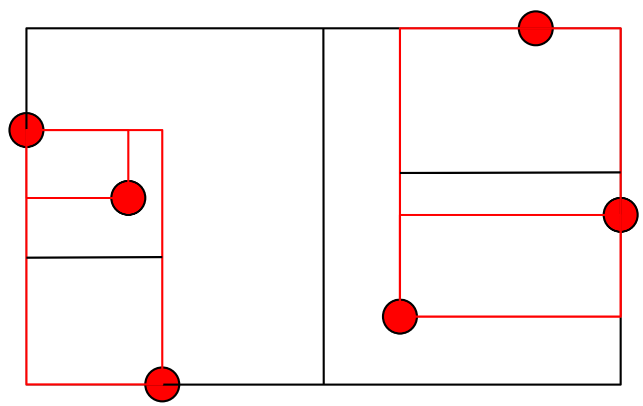
if $|P| = 1$ then **return** leaf(P)

else

d_{max} = dimension of l_{max}

P_1, P_2 = split P along d_{max} at midpoint

Return Node(Tree(P_1),Tree(P_2), l_{max})



KK: Generating the Realization

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

function wsrP(T_1, T_2)
if wellSep(T_1,T_2) **return** {(T_1,T_2)}
else if $l_{max}(T_1) > l_{max}(T_2)$ **then**
return wsrP(left(T_1), T_2) ∪ wsrP(right(T_1), T_2)
else
return wsrP(T_1, left(T_2)) ∪ wsrP(T_1, right(T_2))

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