CSc 8530
Parallel Algorithms

Spring 2019

February 12th, 2019

Root finding: parallel algorithm

ALGORITHM 2.4

(Pointer Jumping)

Input: *A forest of rooted directed trees, each with a self-loop at its root, such that each arc is specified by* $(i, P(i))$, *where* $1 \leq i \leq n$.

Output: *For each vertex i, the root* $S(i)$ *of the tree containing i.*

begin

1. **for** $1 \leq i \leq n$ **pardo**

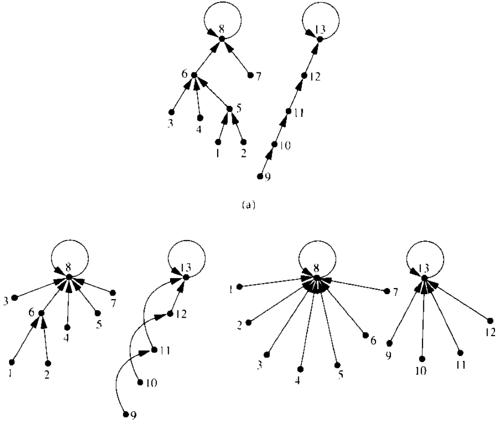
  Set $S(i) := P(i)$

  while $(S(i)) \neq S(S(i))$ **do**

    Set $S(i) := S(S(i))$

end

Root finding: examples



- Notice how the distance to the root is cut in half in each iteration

Root finding: analysis

- Let h be the maximum height of any tree in F
  - In the worst case, $h = O(n)$ (why?)
- Clearly, at the end $S(i)$ is a root, for all i
  - Intuitively, the roots are attractors or steady states of the iterative process
- The convergence rate to a root is 2^h
  - Because we half the distance to the root at each iteration
  - Hence, the number of iterations is $O(\log(h))$
- Each iteration takes $O(1)$ parallel time for $O(n)$ operations (why?)
- **Running time:**
  - $O(\log(h))$
- **Work:**
  - $O(n \log(h))$

Root finding

- We can easily show that this algorithm is not optimal
  - A $T_s(n) = O(n)$ sequential algorithm exists
  - $W(n) > T_s(n)$, so the algorithm cannot be **weakly** optimal
  - Intuitively, we are creating more work for ourselves by operating in parallel
- Could this algorithm be **strongly** optimal?
  - Theoretically yes, if $T(n)$ cannot be improved by *any* other parallel algorithm

Example: parallel prefix

- Assume that each node has a weight $W(i)$
  - Unfortunately, the book uses $W(i)$ for this quantity
  - Not to be confused with $W(n)$, which is work
- We can easily adapt the pointer jumping technique to compute the sum of weights from each node to its corresponding root
  - These sums are similar to the prefix sums we studied last class
- Intuitively, we update both the weights and the successors at the same time

Parallel prefix using pointer jumping

ALGORITHM 2.5

(Parallel Prefix on Rooted Directed Trees)

Input: *A forest of rooted directed trees, each with a self-loop at its root such that (1) each arc is specified by* $(i, P(i))$, *(2) each vertex* i *has a weight* $W(i)$, *and (3) for each root* r, $W(r) = 0$.

Output: *For each vertex* i, $W(i)$ *is set equal to the sum of the weights of vertices on the path from* i *to the root of its tree.*

begin

1. **for** $1 \leq i \leq n$ **pardo**

  Set $S(i) := P(i)$

  while $(S(i)) \neq S(S(i))$ **do**

    Set $W(i) := W(i) + W(S(i))$

    Set $S(i) := S(S(i))$

end

- The running time and work are asymptotically the same as the previous algorithm
  - Because we only added an $O(1)$ operation

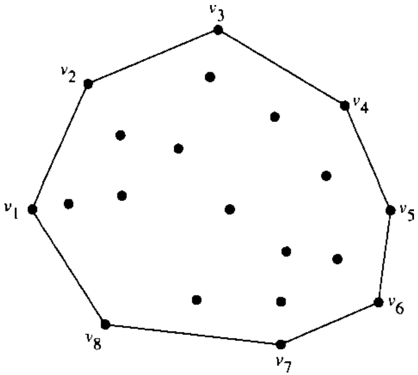
Parallel divide and conquer

- Divide and conquer is one of the most natural ways to exploit parallelism
- Divide and conquer requires three steps:
  1. Partition the input into sub-partitions (ideally of similar size)
  2. Recursively solve the problem for each sub-partition
     - Remember that we can always replace recursion with stack-based iteration
  3. Combine the subproblem solutions into a solution for the overall problem
- Steps 1 and 2 are amenable for parallelization
- Step 3 will typically be more sequential

Example: the convex hull problem

- Let $S = \{p_1, p_2, \ldots, p_n\}$ be n points in the plane
  - i.e., $p_i = (x_i, y_i)$, for all i
- The planar convex hull of S is the smallest convex polygon that contains all of S
- Formally, a polygon Q is convex if the line segment connecting any two points inside it lies wholly within Q
- The convex-hull problem is to determine the *ordered* list $CH(S)$ of points that define the convex hull's boundary

Convex hull example



- $CH(S) = \{v_1, v_2, \ldots, v_8\}$

The convex hull problem

- We can use a divide and conquer strategy to sequentially find $CH(S)$ in $O(n \log(n))$
  - Furthermore, sorting can be reduced to the convex hull problem
  - Thus $T^*(n) = \Theta(n \log(n))$
- We can directly parallelize this sequential approach
  - Intuitively, we assign subsets of points to each processor
- The parallel algorithm is thus weakly optimal (why?)

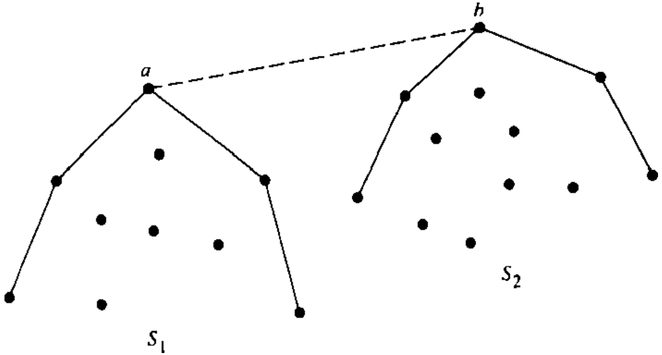
Parallel convex-hull algorithm

- Let p and q be the points in S with the largest and smallest x coordinates, resp.
- Clearly, $p, q \in CH(S)$
- p and q divide the convex hull into two parts:
  - **Upper hull** $UH(S)$: all the points from p to q (clockwise)
  - **Lower hull** $LH(S)$: all the points from q to p (clockwise)
- For simplicity, we will focus on finding $UH(S)$
  - The problem for $LH(S)$ is identical
- We also assume for simplicity that all x and y coordinates are unique and that $n = 2^k$

Parallel convex-hull algorithm

- We first sort all the points by their x coordinates
  - As noted, this takes $\Theta(n \log(n))$
- We can parallel sort n numbers in $T(n) = O(\log(n))$ and $W(n) = O(n \log(n))$
  - We will prove this result in a few classes
- Let $x(p_1) < x(p_2) < \ldots < x(p_n)$
- Let $S_1 = (p_1, p_2, \ldots, p_{n/2})$ and $S_2 = (p_{n/2+1}, \ldots, p_n)$
- Suppose that we already have $UH(S_1)$ and $UH(S_2)$
- Then, the **upper common tangent** is the closest line that lies above both $UH(S_1)$ and $UH(S_2)$

Upper common tangent example



- Both a and b have to be part of the convex hull of S

Parallel convex hull algorithm

- Determining the upper common tangent can be done sequentially with binary search ($O(\log(n))$)
  - We will explore faster parallel alternatives later in the course
- Let $UH(S_1) = (q_1, \ldots, q_s)$ and $UH(S_2) = (q'_1, \ldots, q'_t)$ be the left-to-right sorted lists of points
- Note that $q_1 = p_1$ and $q'_t = p_n$
- Assume that the upper common tangent is (q_i, q'_j)
- Then, $UH(S) = \{q_1, q_2, \ldots, q_i, q'_j, \ldots, q'_t\}$
- Once we find (q_i, q'_j), we can compute $UH(S)$ and its size in $O(1)$ parallel time, using $O(n)$ operations

Parallel convex hull pseudocode

ALGORITHM 2.6
(Simple Upper Hull)

Input: *A set S of n points in the plane, no two of which have the same x or y coordinates such that $x(p_1) < x(p_2) < \cdots < x(p_n)$, where n is a power of 2.*
Output: *The upper hull of S.*

begin

1. *If $n \leq 4$, then use a brute-force method to determine $UH(S)$, and exit.*
2. *Let $S_1 = (p_1, p_2, \cdots, p_{\frac{n}{2}})$ and $S_2 = (p_{\frac{n}{2}+1}, \ldots, p_n)$. Recursively, compute $UH(S_1)$ and $UH(S_2)$ in parallel.*
3. *Find the upper common tangent between $UH(S_1)$ and $UH(S_2)$, and deduce the upper hull of S.*

end

- The pseudocode for $LH(S)$ is identical
- The final convex hull is then given by $CH(S) = UH(S) \cup LH(S)$

Parallel convex hull – analysis

- We can prove that the previous algorithm is correct by induction
  - The base case is valid by brute force
  - Then, intuitively, if $UH(S_1)$ and $UH(S_2)$ are correct, the upper common tangent method will connect them correctly
    - Because, by definition, every point lies on the lower half plane defined by the tangent
- Assume the algorithm takes $T(n)$ steps using $W(n)$ work
  - We will now define these values in terms of a recurrence relation

Parallel convex hull – analysis

- Step 1 takes $O(1)$ sequential time
- Step 2 takes $T(n/2)$ using $2W(n/2)$ operations
- Step 3:
  - Upper common tangent: $O(\log(n))$
  - Combining upper hulls: $O(1)$ parallel time and $O(n)$ work
- Thus:

$$T(n) \leq T\left(\frac{n}{2}\right) + a\log(n)$$

$$W(n) \leq 2W\left(\frac{n}{2}\right) + bn$$

  a and b are positive constants
- Total running time and work:

Parallel convex hull – analysis

- Step 1 takes $O(1)$ sequential time
- Step 2 takes $T(n/2)$ using $2W(n/2)$ operations
- Step 3:
  - Upper common tangent: $O(\log(n))$
  - Combining upper hulls: $O(1)$ parallel time and $O(n)$ work
- Thus:

$$T(n) \leq T\left(\frac{n}{2}\right) + a\log(n)$$
$$W(n) \leq 2W\left(\frac{n}{2}\right) + bn$$

  a and b are positive constants
- Total running time and work:
  - $T(n) = O(\log^2(n))$

Parallel convex hull – analysis

- Step 1 takes $O(1)$ sequential time
- Step 2 takes $T(n/2)$ using $2W(n/2)$ operations
- Step 3:
  - Upper common tangent: $O(\log(n))$
  - Combining upper hulls: $O(1)$ parallel time and $O(n)$ work
- Thus:

$$T(n) \leq T\left(\frac{n}{2}\right) + a\log(n)$$
$$W(n) \leq 2W\left(\frac{n}{2}\right) + bn$$

  a and b are positive constants
- Total running time and work:
  - $T(n) = O(\log^2(n))$
  - $W(n) = O(n\log(n))$

Parallel convex hull – other analyses

- The previous algorithm requires the CREW PRAM model (why?)
- For p processors, the algorithm runs in $O\left(\frac{n \log(n)}{p} + \log^2(n)\right)$
  - What values of p achieve an optimal speedup?

Parallel convex hull – other analyses

- The previous algorithm requires the CREW PRAM model (why?)
  - Merging $UH(S_1)$ and $UH(S_2)$ may require accessing the same point simultaneously
- For p processors, the algorithm runs in $O\left(\frac{n\log(n)}{p} + \log^2(n)\right)$
  - What values of p achieve an optimal speedup?

Parallel convex hull – other analyses

- The previous algorithm requires the CREW PRAM model (why?)
    - Merging $UH(S_1)$ and $UH(S_2)$ may require accessing the same point simultaneously
- For p processors, the algorithm runs in $O\left(\frac{n \log(n)}{p} + \log^2(n)\right)$
    - What values of p achieve an optimal speedup?
    - We want $\frac{n \log(n)}{p} \leq \log^2(n)$ to eliminate the left-hand term

Parallel convex hull – other analyses

- The previous algorithm requires the CREW PRAM model (why?)
  - Merging $UH(S_1)$ and $UH(S_2)$ may require accessing the same point simultaneously
- For p processors, the algorithm runs in $O\left(\frac{n\log(n)}{p} + \log^2(n)\right)$
  - What values of p achieve an optimal speedup?
  - We want $\frac{n\log(n)}{p} \leq \log^2(n)$ to eliminate the left-hand term
  - Thus:

$$\frac{n\log(n)}{p} \leq \log^2(n)$$
$$n\log(n) \leq p\log^2(n)$$
$$\frac{n}{\log(n)} \leq p$$

  - Any value of p in this range will be asymptotically dominated by the $\log^2(n)$ term