CSE 638: Advanced Algorithms

Lectures 10 & 11
(Parallel Connected Components)

Rezaul A. Chowdhury
Department of Computer Science
SUNY Stony Brook
Spring 2013
Symmetry Breaking: List Ranking

1. Flip a coin for each list node
2. If a node $u$ points to a node $v$, and $u$ got a head while $v$ got a tail, combine $u$ and $v$
3. Recursively solve the problem on the contracted list
4. Project this solution back to the original list
Symmetry Breaking: List Ranking

In every iteration a node gets removed with probability \( \frac{1}{4} \) (as a node gets head with probability \( \frac{1}{2} \) and the next node gets tail with probability \( \frac{1}{2} \)).

Hence, a quarter of the nodes get removed in each iteration (expected number).

Thus the expected number of iterations is \( \Theta(\log n) \).

In fact, it can be shown that with high probability,

\[
T_1(n) = O(n) \quad \text{and} \quad T_\infty(n) = O(\log n)
\]
Connected Components: A connected component $C$ of an undirected graph $G$ is a maximal subgraph of $G$ such that every vertex in $C$ is reachable from every other vertex in $C$ following a path in $G$.

Problem: Given an undirected graph identify all its connected components.

Suppose $n$ is the number of vertices in the graph, and $m$ is the number of edges.
Problem: Identify all connected components of an undirected graph. Suppose \( n \) is the number of vertices in the graph, and \( m \) is the number of edges.

Serial Algorithms: Easy to solve in \( \Theta(m + n) \) time using

- Depth First Search (DFS)
- Breadth First Search (BFS)

Parallel Algorithms:

- BFS & DFS: Most efficient polylogarithmic depth algorithms are terribly work inefficient
- Graph Contraction: Can reach polylogarithmic depth without giving up too much or even anything at all in work-efficiency
Randomized Parallel Connected Components
Randomized Parallel Connected Components
Randomized Parallel Connected Components
Randomized Parallel Connected Components
Randomized Parallel Connected Components
Randomized Parallel Connected Components
Randomized Parallel Connected Components
Randomized Parallel Connected Components
Randomized Parallel Connected Components
Randomized Parallel Connected Components
Randomized Parallel Connected Components
Randomized Parallel Connected Components
Randomized Parallel Connected Components
Randomized Parallel Connected Components

[Diagram showing connected components with nodes labeled 1 to 14 and edges connecting them.

- Components are shaded differently.
- Edges have weights indicated, e.g., 2 and 8.
- Numbers 7, 10, 11, and 14 are prominently displayed in some nodes.

The diagram illustrates how different components are formed and connected through these numbers and weights.]
Randomized Parallel Connected Components
Randomized Parallel Connected Components
Randomized Parallel Connected Components
Randomized Parallel Connected Components
Randomized Parallel Connected Components
Randomized Parallel Connected Components
Randomized Parallel Connected Components
Randomized Parallel Connected Components
Randomized Parallel Connected Components
Randomized Parallel Connected Components
Randomized Parallel Connected Components
Randomized Parallel Connected Components (CC)

**Input:** $n$ is the number of vertices in the graph numbered from 1 to $n$, $E$ is the set of edges, and $L[1:n]$ are vertex labels with $L[v] = v$ initially for all $v$.

**Output:** An array $M[1:n]$ where for all $v$, $M[v]$ is the unique id of the connected component containing $v$.

```
Par-Randomized-CC (n, E, L)
1. if |E| = 0 then return L
2. array C[1:n], M[1:n], S[1:|E|]
3. parallel for v ← 1 to n do C[v] ← RANDOM{Head, Tail}
4. parallel for each (u, v) ∈ E do
5.     if C[u] = Tail and C[v] = Head then L[u] ← L[v]
6.     parallel for i ← 1 to |E| do
7.         if L[E[i].u] ≠ L[E[i].v] then S[i] ← 1 else S[i] ← 0
8.     S ← Par-Prefix-Sum(S, +)
9.     array F[1:S[|E|]]
10.    parallel for i ← 1 to |E| do
11.       if L[E[i].u] ≠ L[E[i].v] then
12.           F[S[i]] ← (L[E[i].u], L[E[i].v])
13.    M ← Par-Randomized-CC(n, F, L)
14.    parallel for each (u, v) ∈ E do
15.        if v = L[u] then M[u] ← M[v]
16.    return M
```

unbiased coin toss at each vertex

prepare to remove intra-group edges

Map results back to the original graph

find the rank of each inter-group edge among all such edges

find CC in the contracted graph

copy the inter-group edges to $F$
Randomized Parallel Connected Components (CC)

Suppose $n$ is the number of vertices and $m$ is the number of edges in the original graph.

Each contraction is expected to reduce

#vertices of +ve degree by a factor $\geq \frac{1}{4}$. [why?]

So, the expected number of contraction steps,

$D = O(\log n)$. [show: the bound holds w.h.p.]

For each contraction step span is $\Theta(\log^2 n)$, and work is $\Theta(n + m)$. [why?]

**Work:**

$T_1(n,m) = \Theta(D(n + m))$

$= O((n + m) \log n)$ (w.h.p.)

**Span:**

$T_\infty(n,m) = \Theta(D\log^2 n)$

$= O(\log^3 n)$ (w.h.p.)

**Parallelism:**

$\frac{T_1(n,m)}{T_\infty(n,m)} = \Theta\left(\frac{n+m}{\log^2 n}\right)$
Pointer Jumping

The pointer jumping (or path doubling) technique allows fast processing of data stored in the form of a set of rooted directed trees.

For every node $v$ in the set pointer jumping involves replacing $v \rightarrow \text{next}$ with $v \rightarrow \text{next} \rightarrow \text{next}$ at every step.

Some Applications

- Finding the roots of a forest of directed trees
- Parallel prefix on rooted directed trees
- List ranking
Find-Roots \( (n, P, S) \) \{ Input: A forest of rooted directed trees, each with a self-loop at its root, such that each edge is specified by \((v, P(v))\) for \(1 \leq v \leq n\). Output: For each \(v\), the root \(S(v)\) of the tree containing \(v\). \} 

1. \(\text{parallel for } v \leftarrow 1 \text{ to } n \text{ do}\) 
2. \(S(v) \leftarrow P(v)\) 
3. \(\text{flag} \leftarrow \text{true}\) 
4. \(\text{while flag} = \text{true} \text{ do}\) 
5. \(\text{flag} \leftarrow \text{false}\) 
6. \(\text{parallel for } v \leftarrow 1 \text{ to } n \text{ do}\) 
7. \(S(v) \leftarrow S(S(v))\) 
8. \(\text{if } S(v) \neq S(S(v)) \text{ then flag} \leftarrow \text{true}\)
Let $h$ be the maximum height of any tree in the forest.
Observe that the distance between $v$ and $S(v)$ doubles after each iteration until $S(S(v))$ is the root of the tree containing $v$.
Hence, the number of iterations is $\log h$. Thus (assuming that each parallel for loop takes $\Theta(1)$ time to execute),

**Work:** $T_1(n) = O(n \log h)$ and **Span:** $T_\infty(n) = \Theta(\log h)$

**Parallelism:** $\frac{T_1(n)}{T_\infty(n)} = O(n)$
Deterministic Parallel Connected Components (CC)

Approach

- Form a set of disjoint subtrees
- Use pointer-jumping to reduce each subtree to a single vertex
- Recursively apply the same trick on the contracted graph

Forming Disjoint Subtrees

- Hook each vertex to a neighbor with larger label (if exists)
- Ensures that no cycles are formed
Deterministic Parallel Connected Components (CC)

Forming Disjoint Subtrees

- Hook each vertex to a neighbor with larger label (if exists)
- Ensures that no cycles are formed

But the number of contraction steps can be as large as $n - 1$!
Observation:

Let $G = (V, E)$ be an undirected graph with $n$ vertices in which each vertex has at least one neighbor. Then

$$\text{either } |\{u|(u, v) \in E \land (u < v)\}| \geq \frac{n}{2}$$

$$\text{or } |\{u|(u, v) \in E \land (u > v)\}| \geq \frac{n}{2}$$

Implication:

Between the two directions of hooking (i.e., smaller to larger label, and larger to smaller label) always choose the one that hooks more vertices. Then in each contraction step the number of vertices will be reduced by a factor of at least $\frac{1}{2}$. 

**Deterministic Parallel Connected Components (CC)**
Deterministic Parallel Connected Components (CC)

**Input:** \( n \) is the number of vertices in the graph numbered from 1 to \( n \), \( E \) is the set of edges, and \( L[1:n] \) are vertex labels with \( L[v] = v \) initially for all \( v \).

**Output:** Updated array \( L[1:n] \) where for all \( v \), \( L[v] \) is the unique id of the connected component containing \( v \).

\[\text{Par-Deterministic-CC} \ (n, E, L)\]

1. if \(|E| = 0\) then return \( L \)
2. array \( l2h[1:n], h2l[1:n], S[1:|E|] \)
3. parallel for \( v \leftarrow 1 \) to \( n \) do \( l2h[v] \leftarrow 0, h2l[v] \leftarrow 0 \)
4. parallel for each \((u,v) \in E\) do
5.   if \( u < v \) then \( l2h[u] \leftarrow 1 \) else \( h2l[u] \leftarrow 1 \)
6. \( n_1 \leftarrow \text{Par-Sum} \ (l2h, +), n_2 \leftarrow \text{Par-Sum} \ (h2l, +) \)
7. parallel for each \((u,v) \in E\) do
8.   if \( n_1 \geq n_2 \) and \( u < v \) then \( L[u] \leftarrow v \)
9. else if \( n_1 < n_2 \) and \( u > v \) then \( L[u] \leftarrow v \)
10. \text{Find-Roots} \ (n, L, L)
11. parallel for \( i \leftarrow 1 \) to \(|E|\) do \( S[i] \leftarrow (L[E[i].u] \neq L[E[i].v]) ? 1 : 0 \)
12. \( S \leftarrow \text{Par-Prefix-Sum} \ (S, +) \)
13. array \( F[1:S[|E|]] \)
14. parallel for \( i \leftarrow 1 \) to \(|E|\) do
15.   if \( L[E[i].u] \neq L[E[i].v] \) then \( F[S[i]] \leftarrow (L[E[i].u], L[E[i].v]) \)
16. \( L \leftarrow \text{Par-Deterministic-CC} \ (n, F, L) \)
17. return \( L \)
Deterministic Parallel Connected Components (CC)

Par-Deterministic-CC (n, E, L)
1. if |E| = 0 then return L
2. array l2h[1 : n], h2l[1 : n], S[1 : |E|]
3. parallel for v ← 1 to n do
   l2h[v] ← 0, h2l[v] ← 0
4. parallel for each (u, v) ∈ E do
5.   if u < v then l2h[u] ← 1 else h2l[u] ← 1
6.   n₁ ← Par-Sum (l2h, +), n₂ ← Par-Sum (h2l, +)
7.   parallel for each (u, v) ∈ E do
8.     if n₁ ≥ n₂ and u < v then L[u] ← v
9.     else if n₁ < n₂ and u > v then L[u] ← v
10. Find-Roots (n, L, L)
11. parallel for i ← 1 to |E| do
    S[i] ← (L[E[i].u] ≠ L[E[i].v]) ? 1 : 0
12. S ← Par-Prefix-Sum (S, +)
13. array F[1 : S[|E|]]
14. parallel for i ← 1 to |E| do
15.   if L[E[i].u] ≠ L[E[i].v] then
    F[S[i]] ← (L[E[i].u], L[E[i].v])
16. L ← Par-Deterministic-CC (n, F, L)
17. return L

Each contraction step reduces the number of vertices by a factor of at least $\frac{1}{2}$.

So, number of contraction steps, $D = O(\log n)$.

For contraction step $k \geq 0$ span is $O(\log^2 n)$, and work is $O(n \log n + m)$. [why?]

**Work:**

$T_1(n, m) = O(\sum_{0 \leq i < D} (n \log n + m))$

$= O((n \log n + m)D)$

$= O((n \log n + m) \log n)$

**Span:**

$T_\infty(n, m) = O(D\log^2 n)$

$= O(\log^3 n)$

How to get $T_1(n, m) = O((n + m) \log n)$?