

## Parallel algorithms for fundamental graph-theoretic problems:

- We already used a parallelization of dynamic programming to solve the all-pairs-shortest-path problem.
- Here we are interested in:
  - ▶ determining connected components,
  - ▶ computing minimal spanning trees,
  - ▶ finding maximal independent sets and
  - ▶ coloring graphs.
- Symmetry breaking and the use of randomness turn out to be important new techniques.

# Connected Components

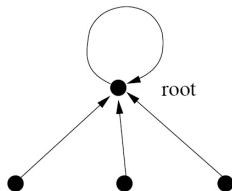
Let  $G = (V, E)$  be an undirected graph.

- A subset  $W \subseteq V$  is **connected** iff any two nodes in  $W$  are connected by a path in  $G$ .
- A connected subset  $W \subseteq V$  is a **connected component** iff no super set of  $W$  is connected.

Determine all connected components of  $G$ .

- Any two connected components are node-disjoint.
- Sequentially, connected components are computed in linear time  $O(|V| + |E|)$ :
  - ▶ The set of all nodes reachable from a node  $v$  is a connected component.
  - ▶ Use depth-first search to determine the set of reachable nodes.
- The problem: there are no efficiently parallelizable graph traversals.

- Throughout we represent the nodes of a component by a **star**:
  - ▶ exactly one root is distinguished,
  - ▶ all nodes including the root point to the root.



- Initially we have to assume that each node is its own connected component.
  - ▶ We start with  $n$  singleton loops, one singleton loop per node.
- Why stars? Merge two stars by
  - ▶ making one root point to the other root and
  - ▶ applying pointer jumping to restore the star property.

# Symmetry Breaking

We may merge two stars  $S_1$  and  $S_2$ , if there is an edge from a node in  $S_1$  to an node in  $S_2$ .

- There will be many edges connecting a given star to other stars. Which merger should we perform?
  - ▶ If we merge  $S_1$  and  $S_2$ , as well as  $S_2$  and  $S_3$  as well as  $\dots$ , then our approach is sequential.
  - ▶ We should forbid long chains, but merging, say,  $S_1$  with  $S_2, S_3, \dots$  is OK, since we can easily represent the result by a single star.
- To decide on mergers immediately, we “**break symmetry**”.
  - ▶ Each star chooses a gender **at random**.
  - ▶ Each male star chooses one female star and attaches himself below the female star.
  - ▶ How to choose the female star?

# The Algorithm

- (0) We have one processor per edge and one processor per node.
- (1) for  $i = 1$  to  $n$  pardo  $\text{parent}[i] = i$ .      // We create singleton loops.
- (2) while (at least one edge of  $G$  connects different stars) do
  - (a) each root  $w$  chooses a gender at random and assigns its gender to its children.
  - (b) if a processor  $e$  is responsible for an edge connecting a male star  $m$  with a female star  $f$ , then  $e$  tries to write  $f$  into the register of  $m$ .
  - (c) The **arbitrary-write resolution** chooses one female star  $f(m)$  for each male star  $m$ . The root of  $m$  is made to point to the root of  $f(m)$ .
  - (d) for  $i = 1$  to  $n$  pardo  
     $\text{parent}[i] = \text{parent}[\text{parent}[i]]$ ;  
    // One round of pointer jumping repairs the star property.

# The Analysis I

- Correctness:
  - ▶ Stars are merged only if they belong to the same connected component.
  - ▶ Say that an edge is **alive** iff it connects two different stars.  
(If no edge is alive, then all connected components are found.)
- We utilize **concurrent read** (when determining whether an edge connects a male and a female star) as well as **concurrent write** (when edge processors try to write into the register of a male star).
- Running time:
  - ▶ The time per iteration is constant.
  - ▶ How many iterations do we have?

# The Analysis II

Say that a star is **alive** iff there is an edge connecting it to a **different** star.

- The root  $r$  of a fixed alive star  $S$  disappears with probability at least  $\frac{1}{4}$ . Why?
  - ▶ Let  $\{u, v\}$  be an edge connecting  $S$  with a different star  $T$ .
  - ▶ With probability  $\frac{1}{4}$ :  $S$  becomes male and  $T$  female. If this happens, then  $r$  will disappear.
- Fix some node  $w$  and determine the probability  $p_w$  that  $w$  does not disappear as root after  $5 \cdot \log_2 n$  iterations. Then

$$\begin{aligned} p_w &\leq \left(1 - \frac{1}{4}\right)^{5 \cdot \log_2 n} = \left(\frac{3}{4}\right)^{5 \cdot \log_2 n} = \left(\frac{243}{1024}\right)^{\log_2 n} \\ &\leq \left(\frac{1}{4}\right)^{\log_2 n} = 2^{-2 \cdot \log_2 n} = n^{-2}. \end{aligned}$$

# Summary

- A **fixed** node survives  $5 \cdot \log_2 n$  iterations with probability at most  $n^{-2}$ .
- **Some** node survives  $5 \cdot \log_2 n$  iterations with probability at most  $n \cdot n^{-2} = n^{-1}$ .

The algorithm runs in expected time  $O(\log_2 n)$  with  $n + m$  processors on a CRCW-PRAM in the arbitrary mode.

In particular  $5 \cdot \log_2 n$  iterations of the while-loop suffice with probability at least  $1 - \frac{1}{n}$ .

- Why expected time  $O(\log_2 n)$ ?