COMP 633 - Parallel Computing

Lecture 4
Thu Sep 2, 2021

PRAM (3)
PRAM algorithm design techniques

Topics

- Parallel connected components algorithm
  - representation of undirected graph and components
  - Illustration of symmetry breaking technique

- We will skip material on Euler tour representation of trees
  - section 3.4 of PRAM handout (not assigned)

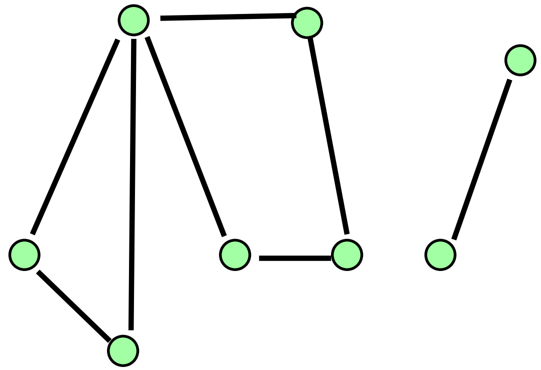
Algorithm Design Technique: Symmetry breaking

- Technique used to distinguish between identical-looking elements
  - graph: all vertices look similar when inspected in parallel
  - labeling to break symmetry
    - create local differences to be exploited by parallel algorithms
      - deterministic, e.g. based on memory address
      - random, breaking symmetry on average

- Sample problem
  - finding connected components of an undirected graph

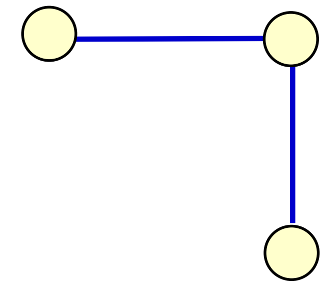
Connected components: definitions

- Undirected graph $G = (V, E)$
  - Undirected edge (u, v) connects vertices u and v
  - Path from v_1 to v_k is a sequence of vertices $(v_1, ..., v_k)$ with $(v_i, v_{i+1}) \in E$
- Connected subgraph
  - subset of V with a path between all pairs of vertices
- Connected component (CC)
  - maximal connected subgraph
- Finding connected components: sequential complexity
  - lower bound
    - must examine all V and E
    - $T_S(V, E) = \Omega(|V| + |E|)$
  - upper bound
    - use DFS and marking
    - $T_S(V, E) = O(|V| + |E|)$

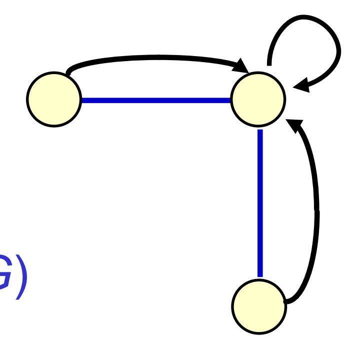


Connected Components Algorithm: representation

- Input: undirected graph $G = (V,E)$ with n vertices, m edges
  - vertices V: integers in the range 1 .. n
  - edges E: length m sequence of (u,v) pairs
    - each edge in G represented by one pair only



- Auxiliary graph: directed forest $H = (V, P)$
  - vertices V are the vertices of G
  - edges: each vertex u has exactly one outgoing edge $(u, P[u])$
    - u is a root if $u = P[u]$
    - no cycles other than self-cycle at a root
    - P defines a set of directed trees in H
    - a tree with height ≤ 1 is a rooted star
  - interpretation of a tree in H
    - $P[u] = v \Rightarrow$ (u and v are in same component of G)
      - each tree is a (not necessarily maximal) connected subgraph of G

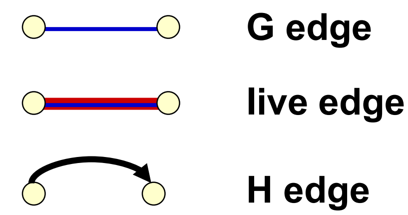


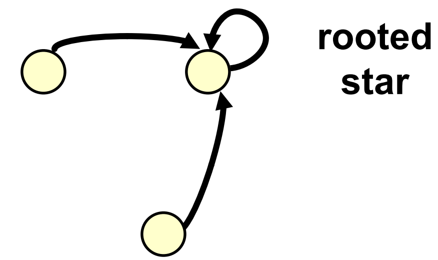
Merging trees in H

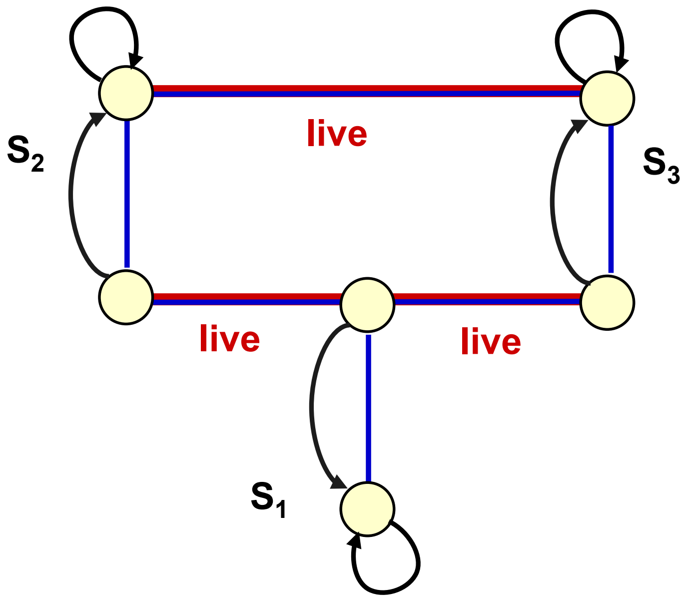
- (*u, v*) is a *live edge* if
  - *u* and *v* are in rooted stars in *H*
  - (*u, v*) is an edge in *G*
  - $P[u] \neq P[v]$
- rooted stars joined by a live edge (u,v) can be *merged*

```
P[P[u]] := P[v]
```

- which merge when multiple choices available?
  - arbitrary
- how to prevent long chains and/or cycles as a result of merging
  - symmetry breaking via random mate
  - pointer doubling step restores rooted star property
- when done?
  - when no live edges remain

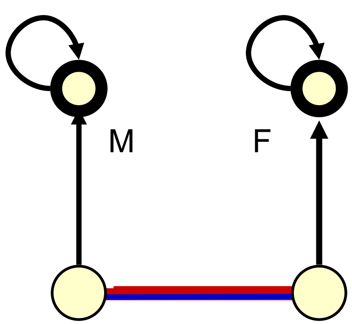




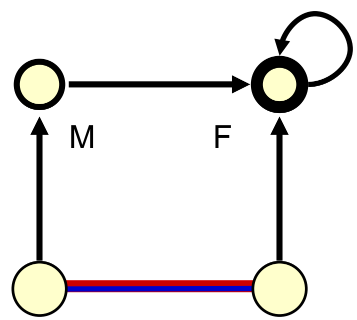


Parallel CC: random mate

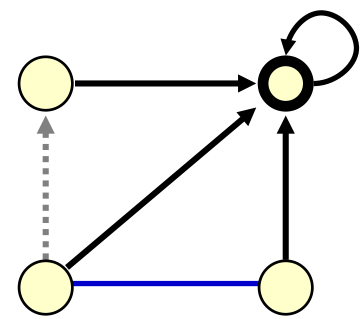
- Basic idea
  - assign random label from the set {M,F} to each rooted star
  - merge rooted stars of opposite label connected by a live edge
    - asymmetry – merge roots M to F direction only
    - cannot generate merge chains of length > 1 or cyclic chains
  - compress trees to rooted stars



live edge



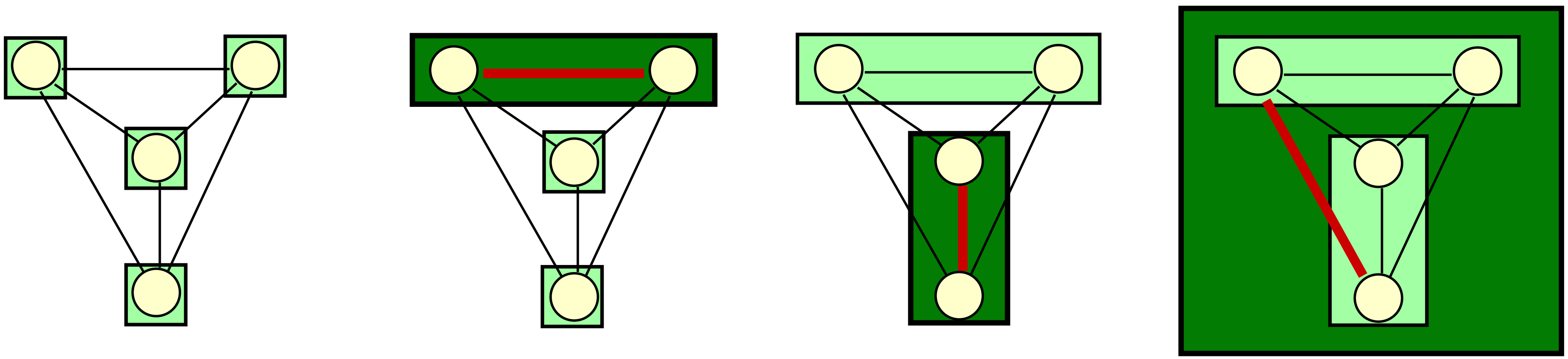
merge roots M to F only



compress paths

Parallel CC: progress

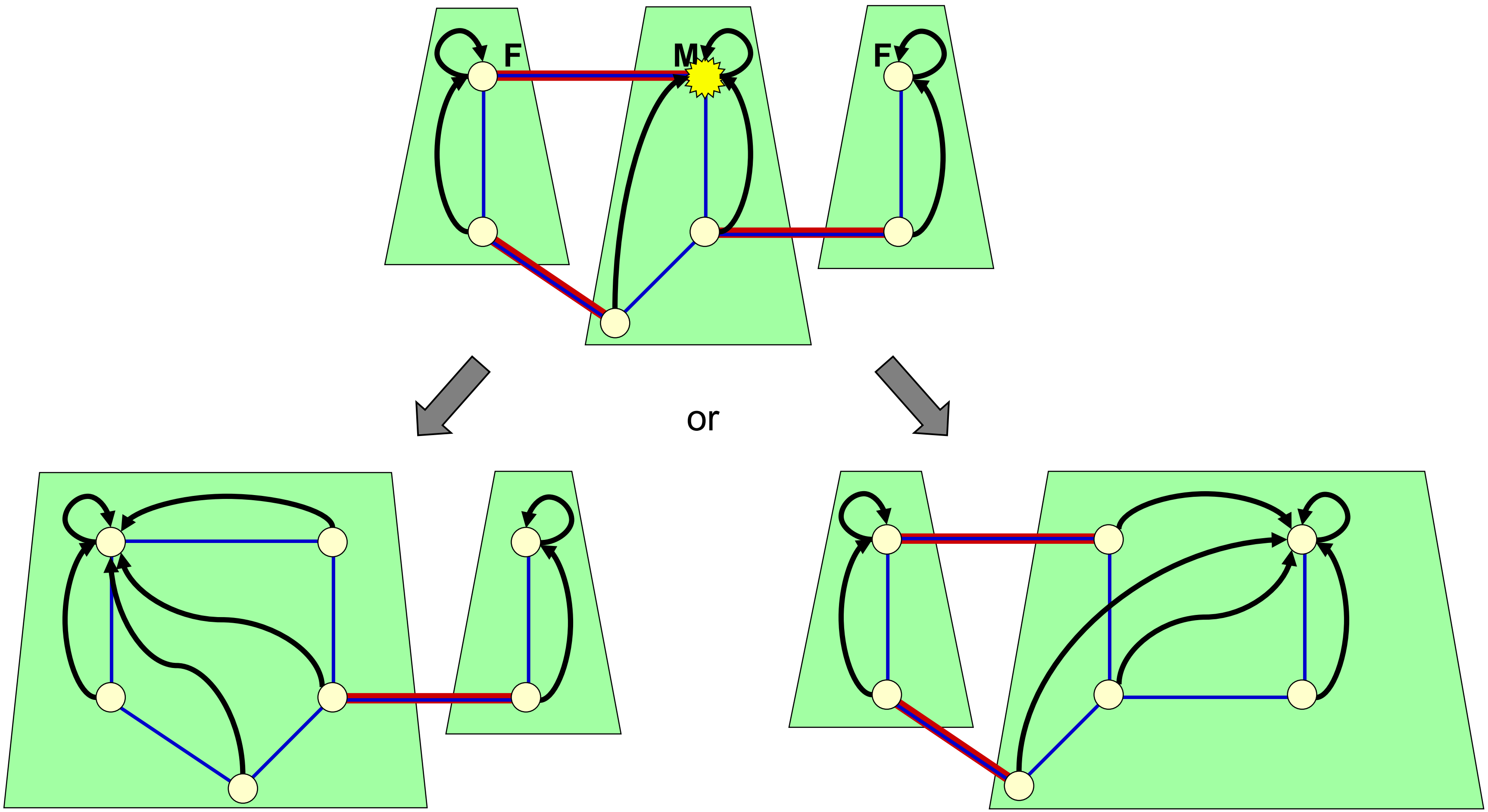
- Initial configuration of H
  - every vertex is its own connected subgraph
  - P[v] = v
- Each step may merge one or more rooted trees in H
- Termination when no live edges remain



(1) (2) (3) (4)

Non-determinism due to concurrent writes

- What if a rooted M star has live edges to multiple rooted F stars?
  - concurrent write resolution determines result



Random mate CC: code

Input: G = (V, E) with |V| = n, |E| = m
Output: `P[1:n]`, with (P[*u*] = P[*v*]) ⇔ (*u* and *v* in same component of G)
Auxiliary: `g[1:n]`

```
forall v in V do
   P[v] := v
end do
while exist-live-edges(G) do
   forall v in V do
      gender[v] := random({M, F})
   enddo
   forall (u, v) in E do
      if label[P[u]] = M and label[P[v]] = F then
         P[P[u]]:=P[v]
      endif
   end do
   forall v in V do
      P[v] := P[P[v]]
   end do
end do
```

Random mate CC: detecting termination

- Are there any remaining live edges?
  - An edge (u,v) is live if it connects vertices in different rooted stars
    - $P[u] \neq P[v]$
  - Test all edges, combine results using CW
    - O(1) step complexity
    - O(m) work complexity

```
exist-live-edges(G) =
      b := false
      forall (u, v) in E do
            if P[u] ≠ P[v] then b := true
      enddo
      return b
```

Random mate CC: correctness

- loop invariant
  - H is a directed forest that includes all vertices in G
  - each tree in H is a rooted star
  - every rooted star is contained within a component of G

- termination condition
  - no live edges

- correctness: **(P[*u*] = P[*v*]) $\Leftrightarrow$ (*u* and *v* in same component of G)**
  - P[u] = P[v] $\Rightarrow$ u,v in same component
    - follows from invariant
  - u,v in same component $\Rightarrow$ P[u] = P[v]
    - by contradiction
      - assume u,v in same component, therefore path $u = w_1 , w_2 ,\ldots, w_n = v$ in G
      - if $P[u] \neq P[v]$, there must exist (w_i,w_{i+1}) in E with $P[w_i] \neq P[w_{i+1}]$
      - (w_i , w_{i+1}) is live edge
      - contradiction to termination

Random mate CC: complexity

- Each iteration of *while*-loop
  - O(1) steps
  - O(n+m) work

- Probability that at a given iteration a live root is joined to another root is at least 1/4
  - probability(live root has label M) = ½
  - probability(live neighbor root has label F) $\geq$ ½

- Probability that a given vertex is a live root after 5 lg n iterations is at most $1/n^2$

- Probability that any vertex is a live root after 5 lg n iterations is at most $1/n$

- With probability $1 - (1/n^{\alpha})$, RM will have terminated after $5\alpha \lg n$ iterations
  - this is definition of “with high probability”

Random mate: summary

- Complexity
  - O(lg *n*) steps with high probability
  - O((*n* +*m*) lg *n*) work with high probability
    - not quite work-efficient

- Memory access model
  - CR in pointer doubling step
  - CW in merging step, termination detection
    - arbitrary CRCW

- Improving work-efficiency
  - eliminate edges, vertices within each supervertex at each iteration
    - factor of 2 reduction in each iteration expected, but not guaranteed
      - depends on sparsity and structure of the graph
      - O(n + m) work complexity
    - step complexity is increased
      - O(lg^2 n) step complexity