CSE 613: Parallel Programming

Lecture 10
(Parallel Minimum Spanning Trees)

Rezaul A. Chowdhury

Department of Computer Science
SUNY Stony Brook
Spring 2019

Spanning Tree

A *spanning tree* of a connected undirected graph $G = (V, E)$ is a connected subgraph $T = (V, E')$ such that $E' \subseteq E$ and $|E'| = |V| - 1$.

Since T connects all V vertices of the graph and has only $|V| - 1$ edges, T cannot contain a cycle.

The connectivity algorithms can easily be extended to return a spanning tree.

- We simply keep track of edges used for hooking
- Since each edge will hook together two components that are not connected yet, and only one edge will succeed in hooking the components, the collection of these edges across all steps will form a spanning tree (i.e., they will connect all vertices and there will be no cycles)

Minimum Spanning Tree

A *minimum spanning tree* of a connected weighted undirected graph $G = (V, E)$ with weights $w(e)$ for $e \in E$ is a spanning tree $T = (V, E')$ of G such that $w(T) = \sum_{e \in E'} w(e)$ is minimized.

Let us assume for simplicity that all edge weights are distinct.

Cut Theorem: For any $U \subset V$ suppose $e \in E$ is the minimum weight edge connecting U and $V \setminus U$, then e must be in $MST(G)$.

Corollary: For every $u \in V$ the edge $(u, v) \in E$ with the minimum weight must be in $MST(G)$.

This property can be used to extend the parallel CC algorithms we have seen to output MST.

Randomized Parallel MST with Priority CW

Input: n is the number of vertices, E is the set of edges, and $MST[\ 1: |E|\ ]$ are flags with all of them initially set to 0. For every edge $(\ u, v\ )$ both $(\ u, v\ )$ and $(\ v, u\ )$ are included in E.

Output: For all i, $MST[\ i\ ]$ is set to 1 if edge $E[\ i\ ]$ is included in the MST.

smallest weight edge from u is chosen for hooking

```
Par-Randomized-MST-Priority-CW ( n, E, MST )
array L[ 1 : n ], C[ 1 : n ], R[ 1 : n ]
sort the edges in E in non-decreasing order of edge weights
parallel for v ← 1 to n do L[ v ] ← v
F ← ( |E| > 0 ) ? True : False
while F = True do
    parallel for v ← 1 to n do C[ v ] ← RANDOM{ Head, Tail }
    parallel for i ← 1 to |E| do R[ E[ i ].u ] ← i ( priority: |E| − i )
    parallel for i ← 1 to |E| do
        u ← E[ i ].u, v ← E[ i ].v
        if C[ u ] = Tail and C[ v ] = Head and R[ u ] = i then
            L[ u ] ← v, MST[ i ] ← 1
    parallel for i ← 1 to |E| do E[ i ] ← ( L[ E[ i ].u ], L[ E[ i ].v ] )
    F ← False
    parallel for each ( u, v ) ∈ E do
        if u ≠ v then F ← True
```

highest priority write, i.e., edge with the smallest weight wins

Randomized Parallel MST with Priority CW

```
Par-Randomized-MST-Priority-CW ( n, E, MST )
1.  array L[ 1 : n ], C[ 1 : n ], R[ 1 : n ]
2.  sort the edges in E in non-decreasing
    order of edge weights
3.  parallel for v ← 1 to n do L[ v ] ← v
4.  F ← ( |E| > 0 ) ? True : False
5.  while F = True do
6.      parallel for v ← 1 to n do
            C[ v ] ← RANDOM{ Head, Tail }
7.      parallel for i ← 1 to |E| do
            R[ E[ i ].u ] ← i ( priority: |E| − i )
8.      parallel for i ← 1 to |E| do
9.          u ← E[ i ].u, v ← E[ i ].v
10.         if C[ u ] = Tail and C[ v ] = Head and R[ u ] = i then
11.             L[ u ] ← v, MST[ i ] ← 1
12.     parallel for i ← 1 to |E| do
            E[ i ] ← ( L[ E[ i ].u ], L[ E[ i ].v ] )
13.     F ← False
14.     parallel for each ( u, v ) ∈ E do
15.         if u ≠ v then F ← True
```

Let n = #vertices, and m = #edges in original graph. Then $m \geq n$ - 1 as graph is connected.

Sorting in step 2 does $\Theta(m \log n)$ work and has $\Theta(\log^3 n)$ depth.

Each contraction is still expected to reduce #vertices by a factor of at least $\frac{1}{4}$. [why?]

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

For each contraction step span is $\Theta(\log n)$, and work is $\Theta(n + m)$.

Work: $T_1(n, m) = \Theta\big(m \log n + D(n + m)\big)$
$= \Theta(m \log n)$

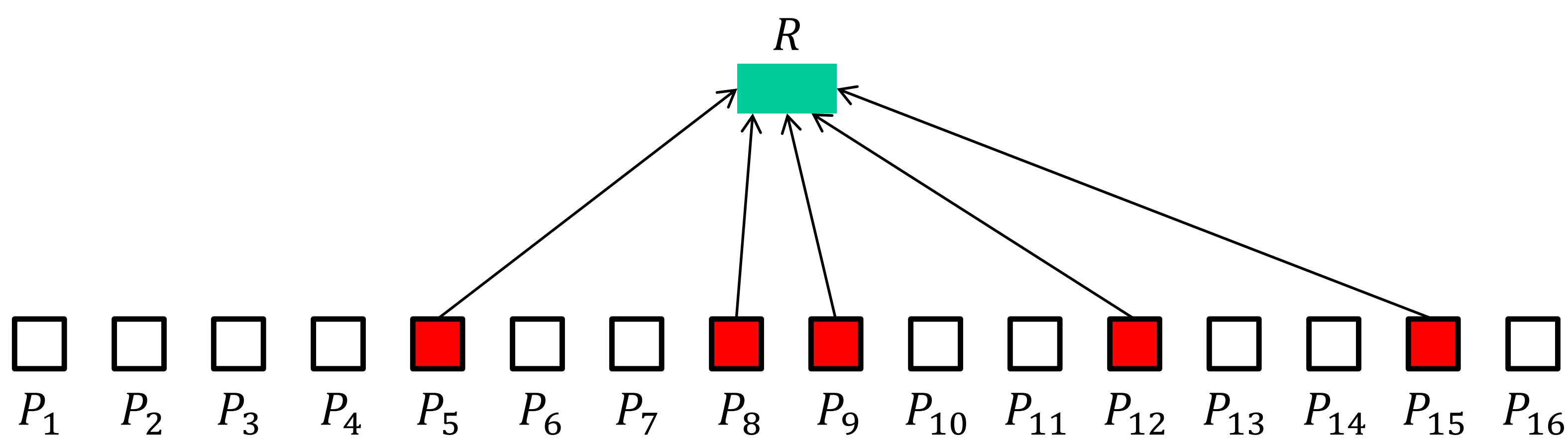
Span: $T_\infty(n, m) = \Theta(\log^3 n + D \log n)$
$= \Theta(\log^3 n)$

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

Concurrent Writes where the Leftmost Writer Wins

Problem: Consider a set of n processors $\{P_1, P_2, \dots, P_n\}$ of which some are trying to write (not necessarily the same value) to a common location. Devise a parallel strategy to identify the leftmost writer (i.e., the writer with the smallest id) assuming that during concurrent writes to the same location an arbitrary writer may succeed.

Example: Suppose among the 16 processors below the red ones are trying to write their ids (i.e., a red P_i is trying to write i) to a common location R.



Eliminating Priority CW by Sorting (e.g., Using Radix Sort)

Input: An array A of n keys, each represented as a b bit integer.

Output: Array A with its keys sorted in non-decreasing order. The output is *stable* meaning keys of equal value retain their input order.

find ranks of all keys with bit k = 0, and of all keys with bit k = 1 by treating them as separate groups

```
Par-Radix-Sort ( A, n, b )
array F_0[ 1 : n ], F_1[ 1 : n ], S_0[ 1 : n ], S_1[ 1 : n ], B[ 1 : n ]
for k ← 0 to b − 1 do
    parallel for i ← 1 to n do
        F_1[ i ] ← SHIFT-RIGHT( A[ i ], k ) mod 2
        F_0[ i ] ← 1 − F_1[ i ]
    S_0 ← Par-Prefix-Sum ( F_0, + )
    S_1 ← Par-Prefix-Sum ( F_1, + )
    parallel for i ← 1 to n do
        if F_1[ i ] = 0 then B[S_0[ i ] ] ← A[ i ]
        else B[ S_0[ n ] + S_1[ i ] ] ← A[ i ]
    parallel for i ← 1 to n do
        A[ i ] ← B[ i ]
```

extract the k-th bit and its negation

use the ranks to order the keys appropriately, and place all keys with bit k=0 ahead of all keys with bit k=1

Eliminating Priority CW by Sorting (e.g., Using Radix Sort)

```
Par-Radix-Sort ( A, n, b )
1.  array F0[ 1 : n ], F1 [ 1 : n ],
          S0[ 1 : n ], S1 [ 1 : n ], B [ 1 : n ]
2.  for k ← 0 to b − 1 do
3.      parallel for i ← 1 to n do
4.          F1 [ i ] ← SHIFT-RIGHT( A[ i ], k ) mod 2
5.          F0 [ i ] ← 1 − F1 [ i ]
6.      S0 ← Par-Prefix-Sum ( F0,  + )
7.      S1 ← Par-Prefix-Sum ( F1,  + )
8.      parallel for i ← 1 to n do
9.          if F1[ i ] = 0 then B[S0[ i ] ] ← A[ i ]
10.         else B[ S0[ n ] + S1[ i ] ] ← A[ i ]
11.     parallel for i ← 1 to n do
12.         A[ i ] ← B[ i ]
```

The *serial for loop* in line 2 iterates b times, and each iteration performs $\Theta(n)$ work and has $\Theta(\log^2 n)$ depth.

Work: $T_1(n) = \Theta(bn)$

Span: $T_\infty(n) = \Theta(b \log^2 n)$

Parallelism: $\frac{T_1(n)}{T_\infty(n)} = \Theta\left(\frac{n}{\log^2 n}\right)$

Eliminating Priority CW by Sorting (e.g., Using Radix Sort)

Input: n is the number of vertices and E is the set of edges.

Output: For $1 \leq u \leq n$, $R[\ u\ ]$ is set to the smallest index i such that $E[\ i\ ].u = u$.

```
Par-Simulate-Priority-CW-using-Radix-Sort ( n, E, R )
1. array A[ 1 : |E| ]
2. k ← ⌈ log |E| ⌉ + 1
3. parallel for i ← 1 to |E| do A[ i ] ← SHIFT-LEFT( E[ i ].u, k ) + i
4. Par-Radix-Sort ( A, |E|, k + ⌈ log n ⌉ )
5. parallel for i ← 1 to |E| do
6.     u ← SHIFT-RIGHT( A[ i ], k )
7.     j ← A[ i ] − SHIFT-LEFT( u, k )
8.     if i = 1 or u ≠ SHIFT-RIGHT( A[ i − 1 ], k ) then R[ u ] ← j
```

Assuming, $m = |E|$. For radix sort $b = \Theta(\log n)$.

Work: $\Theta(bm) = \Theta(m \log n)$

Span: $\Theta(b \log^2 n) = \Theta(\log^3 n)$

Randomized Parallel MST with Priority CW

Input: *n* is the number of vertices, *E* is the set of edges, and *MST*[1: |*E*|] are flags with all of them initially set to 0. For every edge (*u*, *v*) both (*u*, *v*) and (*v*, *u*) are included in *E*.

Output: For all *i, MST*[*i*] is set to 1 if edge *E*[*i*] is included in the MST.

```
Par-Randomized-MST-Priority-CW ( n, E, MST )
array L[ 1 : n ], C[ 1 : n ], R[ 1 : n ]
sort the edges in E in non-decreasing order of edge weights
parallel for v ← 1 to n do L[ v ] ← v
F ← ( |E| > 0 ) ? True : False
while F = True do
    parallel for v ← 1 to n do C[ v ] ← RANDOM{ Head, Tail }
    parallel for i ← 1 to |E| do R[ E[ i ].u ] ← i ( priority: |E| − i )
    parallel for i ← 1 to |E| do
        u ← E[ i ].u, v ← E[ i ].v
        if C[ u ] = Tail and C[ v ] = Head and R[ u ] = i then
            L[ u ] ← v, MST[ i ] ← 1
    parallel for i ← 1 to |E| do E[ i ] ← ( L[ E[ i ].u ], L[ E[ i ].v ] )
    F ← False
    parallel for each ( u, v ) ∈ E do
        if u ≠ v then F ← True
```

Randomized Parallel MST w/o Priority CW

Input: *n* is the number of vertices, *E* is the set of edges, and *MST*[1: |*E*|] are flags with all of them initially set to 0. For every edge (*u*, *v*) both (*u*, *v*) and (*v*, *u*) are included in *E*.

Output: For all *i*, *MST*[*i*] is set to 1 if edge *E*[*i*] is included in the MST.

```
Par-Randomized-MST-Priority-CW ( n, E, MST )
    array L[ 1 : n ], C[ 1 : n ], R[ 1 : n ]
    sort the edges in E in non-decreasing order of edge weights
    parallel for v ← 1 to n do L[ v ] ← v
    F ← ( |E| > 0 ) ? True : False
    while F = True do
        parallel for v ← 1 to n do C[ v ] ← RANDOM{ Head, Tail }
        Par-Simulate-Priority-CW-using-Radix-Sort ( n, E, R )
        parallel for i ← 1 to |E| do
            u ← E[ i ].u, v ← E[ i ].v
            if C[ u ] = Tail and C[ v ] = Head and R[ u ] = i then
                L[ u ] ← v, MST[ i ] ← 1
        parallel for i ← 1 to |E| do E[ i ] ← ( L[ E[ i ].u ], L[ E[ i ].v ] )
        F ← False
        parallel for each ( u, v ) ∈ E do
            if u ≠ v then F ← True
```

Randomized Parallel MST w/o Priority CW

```
Par-Randomized-MST-Priority-CW ( n, E, MST )
1.  array L[ 1 : n ], C[ 1 : n ], R[ 1 : n ]
2.  sort the edges in E in non-decreasing
    order of edge weights
3.  parallel for v ← 1 to n do L[ v ] ← v
4.  F ← ( |E| > 0 ) ? True : False
5.  while F = True do
6.      parallel for v ← 1 to n do
            C[ v ] ← RANDOM{ Head, Tail }
7.      Par-Simulate-Priority-CW-using-Radix-Sort ( n, E, R )
8.      parallel for i ← 1 to |E| do
9.          u ← E[ i ].u, v ← E[ i ].v
10.         if C[ u ] = Tail and C[ v ] = Head and R[ u ] = i then
11.             L[ u ] ← v, MST[ i ] ← 1
12.     parallel for i ← 1 to |E| do
            E[ i ] ← ( L[ E[ i ].u ], L[ E[ i ].v ] )
13.     F ← False
14.     parallel for each ( u, v ) ∈ E do
15.         if u ≠ v then F ← True
```

Let n = #vertices, and m = #edges in original graph. Then $m \geq n - 1$ as graph is connected.

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

For each contraction step span is $\Theta(\log^3 n)$, and work is $\Theta(n + m \log n)$.

Work:

$$T_1(n, m) = \Theta\big(m \log n + D(n + m \log n)\big)$$
$$= \Theta(m \log^2 n)$$

Span:

$$T_\infty(n, m) = \Theta(\log^3 n + D \log^3 n)$$
$$= \Theta(\log^4 n)$$

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

Ranking integer Keys Using Counting Sort

Input: An array $S[\ 1{:}\ n\ ]$ of keys, each represented as an d bit integer.

Output: Stable ranking of the keys in S when sorted in non-decreasing order.

Approach:

— Suppose $P_1, P_2, \dots, P_p$ are the available processing elements.

— Split S into p segments of approximately $\frac{n}{p}$ keys each.
Let S_i denote the i-th $(1 \leq i \leq p)$ such segment.

— Assign S_i to P_i.

— Since the ranking must be stable, all occurrences of v in S_i must be ranked ahead of all occurrences of v in S_{i+1}.

— For $v \in [0, 2^d - 1]$, let $f[v][i]$ be the frequency of v in $P_1, P_2, \dots, P_i$.

— Then $\sum_{u=0}^{v-1} f[u][p]$ is the total number of keys in S with value $< v$.

— Clearly, the first occurrence of v in P_i must have a global rank of $1 + \sum_{u=0}^{v-1} f[u][p] + f[v][i-1]$ (assuming $f[v][0] = 0$).

Ranking integer Keys Using Counting Sort

Input: An array $S[\,1{:}\,n\,]$ of keys, each represented as an d bit integer.

Output: Array $r[\,1{:}\,n\,]$ with $r[\,i\,]$ giving the rank of $S[\,i\,]$ when the keys in S are sorted in non-decreasing order. The ranking is *stable*.

```
Par-Counting-Rank ( S, n, d, r )     { p = #processing elements }
array f[ 0 : 2^d − 1 ][ 1 : p ], r_1[ 0 : 2^d − 1 ][ 1 : p ],
        j_s[ 1 : p ], j_e[ 1 : p ], ofs[ 1 : p ]
parallel for i ← 1 to p do
    for j ← 0 to 2^d − 1 do f[ j ][ i ] ← 0
    j_s[ i ] ← ( i − 1 )⌊ n / p ⌋ + 1,  j_e[ i ] ← ( i < p ) ? ( i⌊ n / p ⌋ ) : n
    for j ← j_s[ i ] to j_e[ i ] do f[ S[ j ] ][ i ] ← f[ S[ j ] ][ i ] + 1
for j ← 0 to 2^d − 1 do
    f[ j ][ 1 : p ] ← Par-Prefix-Sum ( f[ j ][ 1 : p ],  + )
parallel for i ← 1 to p do
    ofs[ i ] ← 1
    for j ← 0 to 2^d − 1 do
        r_1[ j ][ i ] ← ( i = 1) ? ofs[ i ] : ( ofs[ i ] + f[ j ][ i − 1 ] )
        ofs[ i ] ← ofs[ i ] + f[ j ][ p ]
    for j ← j_s[ i ] to j_e[ i ] do
        r[ j ] ← r_1[ S[ j ] ][ i ]
        r_1[ S[ j ] ][ i ] ← r_1[ S[ j ] ][ i ] + 1
```

for each key $j \in [\,0,\ 2^d - 1\,]$ count the frequency of j in processors $\le i$

find the ranks of all occurrences of each key based on the ranks of their first occurrences

processor i counts frequency of each key $\in [\,0,\ 2^d - 1\,]$

processor i finds the overall rank of the first occurrence of each key $\in [\,0,\ 2^d - 1\,]$ in its segment

Ranking integer Keys Using Counting Sort

```
Par-Counting-Rank ( S, n, d, r )      { p = #proc elements }
1.  array f[ 0 : 2^d − 1 ][ 1 : p ], r_1[ 0 : 2^d − 1 ][ 1 : p ],
            j_s[ 1 : p ], j_e[ 1 : p ], ofs[ 1 : p ]
2.  parallel for i ← 1 to p do
3.      for j ← 0 to 2^d − 1 do f[ j ][ i ] ← 0
4.      j_s[ i ] ← ( i − 1 )⌈ n / p ⌉ + 1
        j_e[ i ] ← ( i < p ) ? ( i⌈ n / p ⌉ ) : n
5.      for j ← j_s[ i ] to j_e[ i ] do
            f[ S[ j ] ][ i ] ← f[ S[ j ] ][ i ] + 1
6.  for j ← 0 to 2^d − 1 do
7.      f[ j ][ 1 : p ] ← Par-Prefix-Sum ( f[ j ][ 1 : p ],  + )
8.  parallel for i ← 1 to p do
9.      ofs[ i ] ← 1
10.     for j ← 0 to 2^d − 1 do
11.         r_1[ j ][ i ] ← ( i = 1) ? ofs[ i ]
                                  : ( ofs[ i ] + f[ j ][ i − 1 ] )
12.         ofs[ i ] ← ofs[ i ] + f[ j ][ p ]
13.     for j ← j_s[ i ] to j_e[ i ] do
14.         r[ j ] ← r_1[ S[ j ] ][ i ]
15.         r_1[ S[ j ] ][ i ] ← r_1[ S[ j ] ][ i ] + 1
```

We will analyze running time on *p* processing elements.

$$T'_p(n,d) = \Theta\left(\log(p+1) + 2^d + \frac{n}{p}\right) \quad [\text{ L: 2-5 }]$$

$$+\,\Theta\left(2^d \log^2(p+1)\right) \quad [\text{ L: 6-7 }]$$

$$+\,\Theta\left(\log(p+1) + 2^d + \frac{n}{p}\right) \quad [\text{ L: 8-15 }]$$

$$= \Theta\left(\frac{n}{p} + 2^d \log^2(p+1)\right)$$

Radix Sort with Ranking Using Counting Sort

Input: An array A of n keys, each represented as a b bit integer.

Output: Array A with its keys sorted in non-decreasing order. The output is *stable* meaning keys of equal value retain their input order.

```
Par-Radix-Sort-with-Counting-Rank ( A, n, b )
    array S[ 1 : n ], r [ 1 : n ], B [ 1 : n ]
    d ← ⌈ log( n / ( p log n ) ) ⌉
    for k ← 0 to b − 1 by d do
        q ← ( k + d ≤ b ) ? d : ( b − k )
        parallel for i ← 1 to n do
            S [ i ] ← EXTRACT-BIT-SEGMENT( A[ i ], k, k + q − 1 )
        Par-Counting-Rank ( S, n, q, r )
        parallel for i ← 1 to n do
            B[ r[ i ] ] ← A[ i ]
        parallel for i ← 1 to n do
            A[ i ] ← B[ i ]
```

Radix Sort with Ranking Using Counting Sort

```
Par-Radix-Sort-with-Counting-Rank ( A, n, b )
    array S[ 1 : n ], r [ 1 : n ], B [ 1 : n ]
    d ← ⌈ log( n / ( p log n ) ) ⌉
    for k ← 0 to b − 1 by d do
        q ← ( k + d ≤ b ) ? d : ( b − k )
        parallel for i ← 1 to n do
            S [ i ] ← EXTRACT-BIT-SEGMENT( A[ i ], k, k + q − 1 )
        Par-Counting-Rank ( S, n, q, r )
        parallel for i ← 1 to n do
            B[ r[ i ] ] ← A[ i ]
        parallel for i ← 1 to n do
            A[ i ] ← B[ i ]
```

We assume that $1 \le p \le \frac{n}{2\log n}$,

and $b = \quad(\log n)$.

We will analyze running time on p processing elements.

$$T_p(n) = \Theta\left(\frac{b}{d}\left(\frac{n}{p} + T'_p(n,d)\right)\right)$$

$$= \Theta\left(\frac{b}{d}\left(\frac{n}{p} + 2^d \log^2(p+1)\right)\right)$$

Then **work**: $T_1(n) = \Theta\left(\frac{b}{\log n - \log\log n}\left(n + \frac{n}{\log n}\right)\right) = \Theta\left(\frac{bn}{\log n}\right) = O(n)$

and **span:** $T_\infty(n) = T_{\frac{n}{2\log n}}(n) = \Theta\big(b(\log n + \log^2 n)\big) = \Theta(b\log^2 n) = O(\log^3 n)$

Then **parallelism:** $\frac{T_1(n)}{T_\infty(n)} = \Theta\left(\frac{n}{\log^3 n}\right)$

Eliminating Priority CW by Sorting (Using Radix Sort with Ranking by Counting Sort)

Input: n is the number of vertices and E is the set of edges.

Output: For $1 \le u \le n$, $R[\,u\,]$ is set to the smallest index i such that $E[\,i\,].u = u$.

```
Par-Simulate-Priority-CW-using-Radix-Sort-2 ( n, E, R )
array A[ 1 : |E| ]
k ← ⌈ log |E| ⌉ + 1
parallel for i ← 1 to |E| do A[ i ] ← SHIFT-LEFT( E[ i ].u, k ) + i
Par-Radix-Sort-with-Counting-Rank ( A, |E|, k + ⌈ log n ⌉ )
parallel for i ← 1 to |E| do
    u ← SHIFT-RIGHT( A[ i ], k )
    j ← A[ i ] − SHIFT-LEFT( u, k )
    if i = 1 or u ≠ SHIFT-RIGHT( A[ i − 1 ], k ) then R[ u ] ← j
```

Assuming, $m = |E|$. For radix sort $b = \Theta(\log n)$.

Work: $\Theta(m) = \Theta(m)$

Span: $\Theta(\log^3 n)$

Randomized Parallel MST with Priority CW

Input: *n* is the number of vertices, *E* is the set of edges, and *MST*[1: |*E*|] are flags with all of them initially set to 0. For every edge (*u*, *v*) both (*u*, *v*) and (*v*, *u*) are included in *E*.

Output: For all *i*, *MST*[*i*] is set to 1 if edge *E*[*i*] is included in the MST.

```
Par-Randomized-MST-Priority-CW ( n, E, MST )
array L[ 1 : n ], C[ 1 : n ], R[ 1 : n ]
sort the edges in E in non-decreasing order of edge weights
parallel for v ← 1 to n do L[ v ] ← v
F ← ( |E| > 0 ) ? True : False
while F = True do
    parallel for v ← 1 to n do C[ v ] ← RANDOM{ Head, Tail }
    parallel for i ← 1 to |E| do R[ E[ i ].u ] ← i ( priority: |E| − i )
    parallel for i ← 1 to |E| do
        u ← E[ i ].u, v ← E[ i ].v
        if C[ u ] = Tail and C[ v ] = Head and R[ u ] = i then
            L[ u ] ← v, MST[ i ] ← 1
    parallel for i ← 1 to |E| do E[ i ] ← ( L[ E[ i ].u ], L[ E[ i ].v ] )
    F ← False
    parallel for each ( u, v ) ∈ E do
        if u ≠ v then F ← True
```

Randomized Parallel MST w/o Priority CW

Input: n is the number of vertices, E is the set of edges, and $MST[\ 1{:}\ |E|\ ]$ are flags with all of them initially set to 0. For every edge $(\ u, v\ )$ both $(\ u, v\ )$ and $(\ v, u\ )$ are included in E.

Output: For all i, $MST[\ i\ ]$ is set to 1 if edge $E[\ i\ ]$ is included in the MST.

```
Par-Randomized-MST-Priority-CW ( n, E, MST )

array L[ 1 : n ], C[ 1 : n ], R[ 1 : n ]
sort the edges in E in non-decreasing order of edge weights
parallel for v ← 1 to n do L[ v ] ← v
F ← ( |E| > 0 ) ? True : False
while F = True do
    parallel for v ← 1 to n do C[ v ] ← RANDOM{ Head, Tail }
    Par-Simulate-Priority-CW-using-Radix-Sort-2 ( n, E, R )
    parallel for i ← 1 to |E| do
        u ← E[ i ].u, v ← E[ i ].v
        if C[ u ] = Tail and C[ v ] = Head and R[ u ] = i then
            L[ u ] ← v, MST[ i ] ← 1
    parallel for i ← 1 to |E| do E[ i ] ← ( L[ E[ i ].u ], L[ E[ i ].v ] )
    F ← False
    parallel for each ( u, v ) ∈ E do
        if u ≠ v then F ← True
```

Randomized Parallel MST w/o Priority CW

```
Par-Randomized-MST-Priority-CW ( n, E, MST )
array L[ 1 : n ], C[ 1 : n ], R[ 1 : n ]
sort the edges in E in non-decreasing
order of edge weights
parallel for v ← 1 to n do L[ v ] ← v
F ← ( |E| > 0 ) ? True : False
while F = True do
    parallel for v ← 1 to n do
        C[ v ] ← RANDOM{ Head, Tail }
    Par-Simulate-Priority-CW-using-Radix-Sort-2 ( n, E, R )
    parallel for i ← 1 to |E| do
        u ← E[ i ].u, v ← E[ i ].v
        if C[ u ] = Tail and C[ v ] = Head and R[ u ] = i then
            L[ u ] ← v, MST[ i ] ← 1
    parallel for i ← 1 to |E| do
        E[ i ] ← ( L[ E[ i ].u ], L[ E[ i ].v ] )
    F ← False
    parallel for each ( u, v ) ∈ E do
        if u ≠ v then F ← True
```

Let n = #vertices, and m = #edges in original graph. Then $m \geq n - 1$ as graph is connected.

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

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

Work:

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

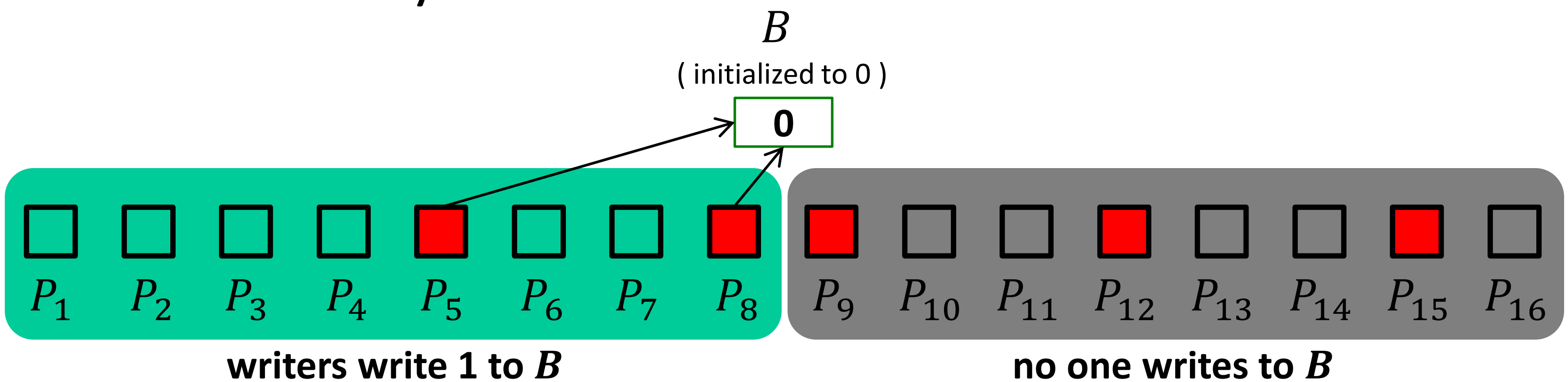
Span:

$$T_\infty(n,m) = \Theta(\log^3 n + D\log^3 n)$$
$$= \Theta(\log^4 n)$$

Parallelism: $\frac{T_1(n,m)}{T_\infty(n,m)} = \Theta\left(\frac{m}{\log^3 n}\right)$

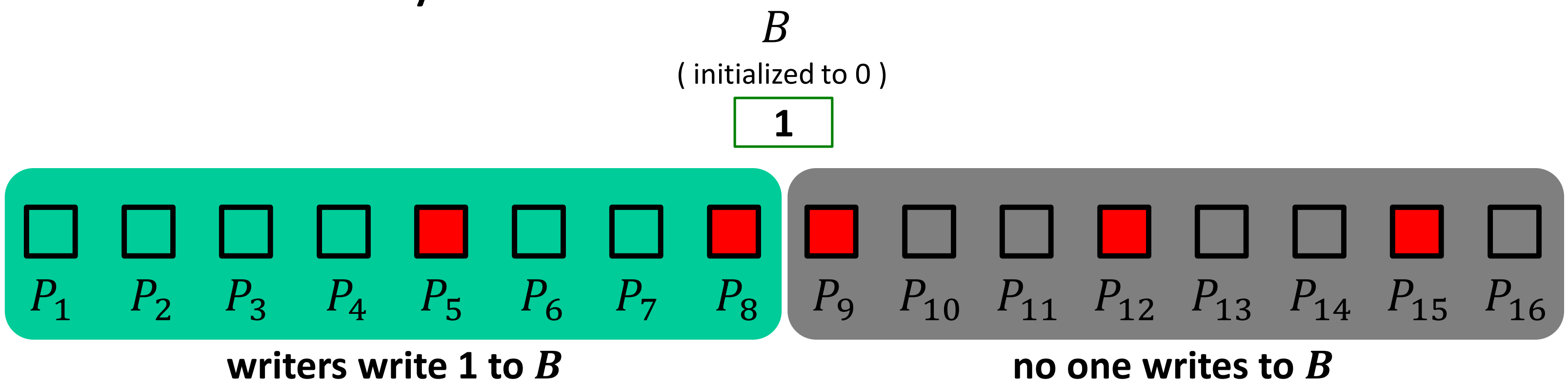
Concurrent Writes where the Leftmost Writer Wins

Solution: Use binary search.



Concurrent Writes where the Leftmost Writer Wins

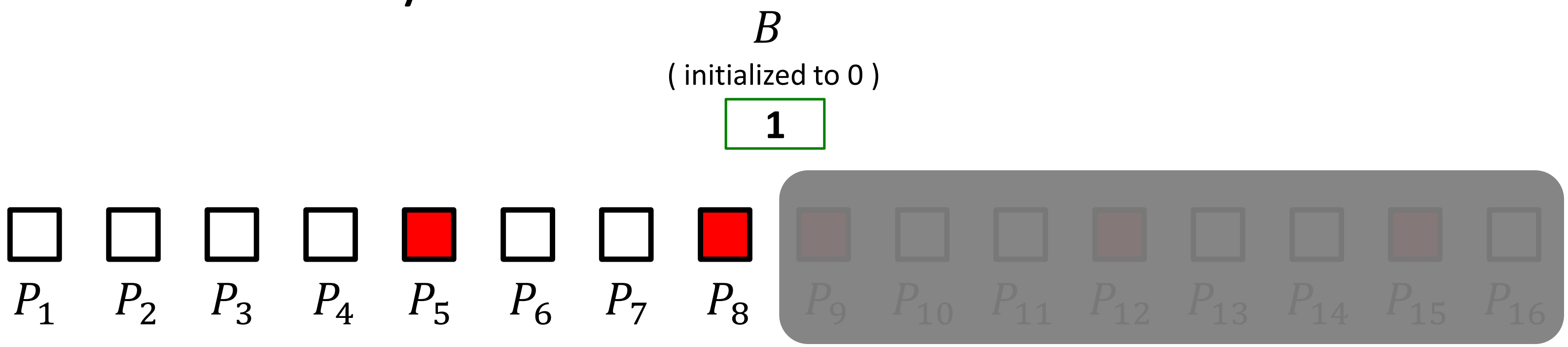
Solution: Use binary search.



After stage 1: $B = 1$, and so processors $P_9, \ldots, P_{16}$ are eliminated.

Concurrent Writes where the Leftmost Writer Wins

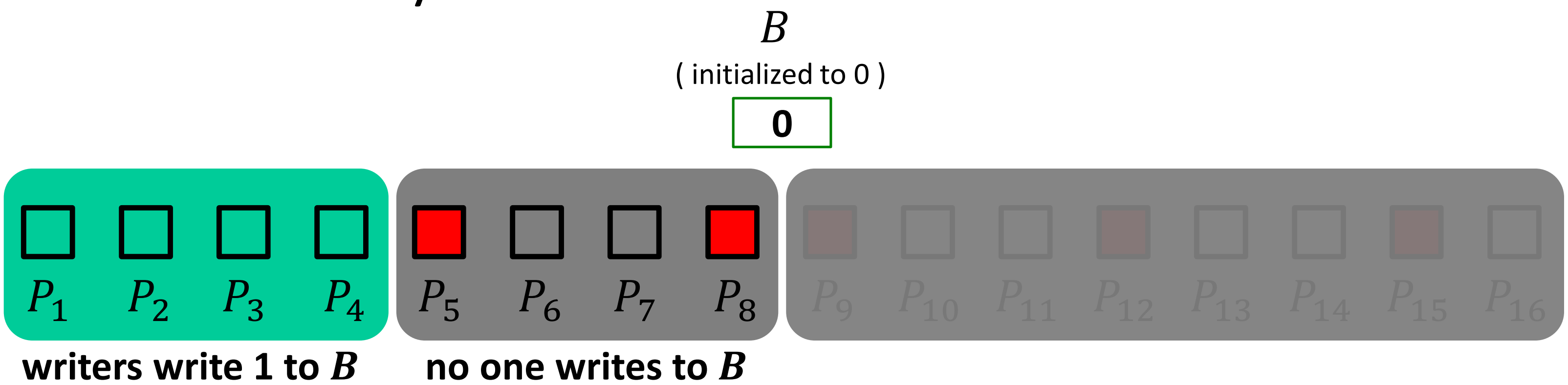
Solution: Use binary search.



<u>After stage 1</u>: $B = 1$, and so processors $P_9, \ldots, P_{16}$ are eliminated.

Concurrent Writes where the Leftmost Writer Wins

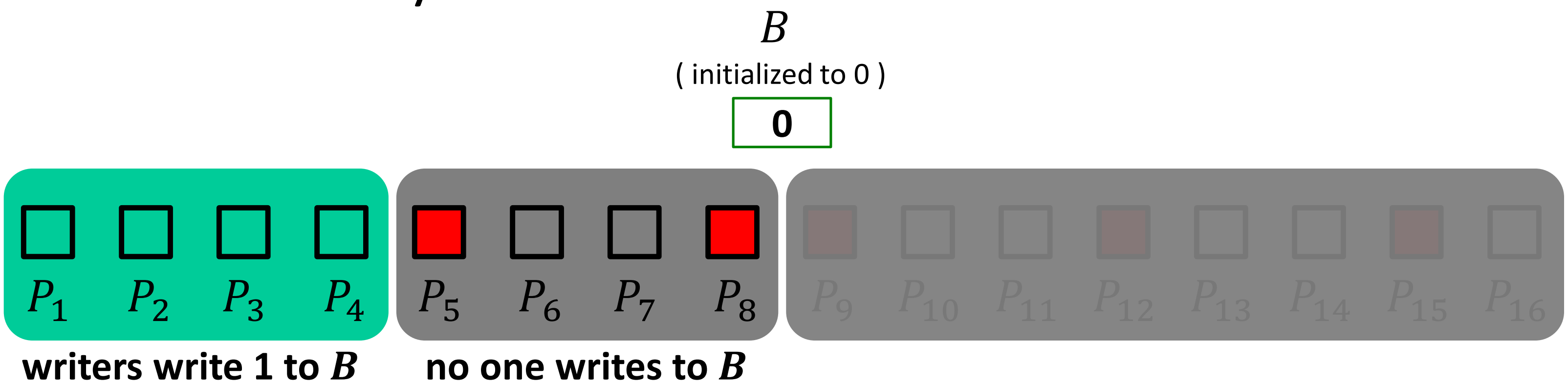
Solution: Use binary search.



After stage 1: $B = 1$, and so processors $P_9, \ldots, P_{16}$ are eliminated.

Concurrent Writes where the Leftmost Writer Wins

Solution: Use binary search.

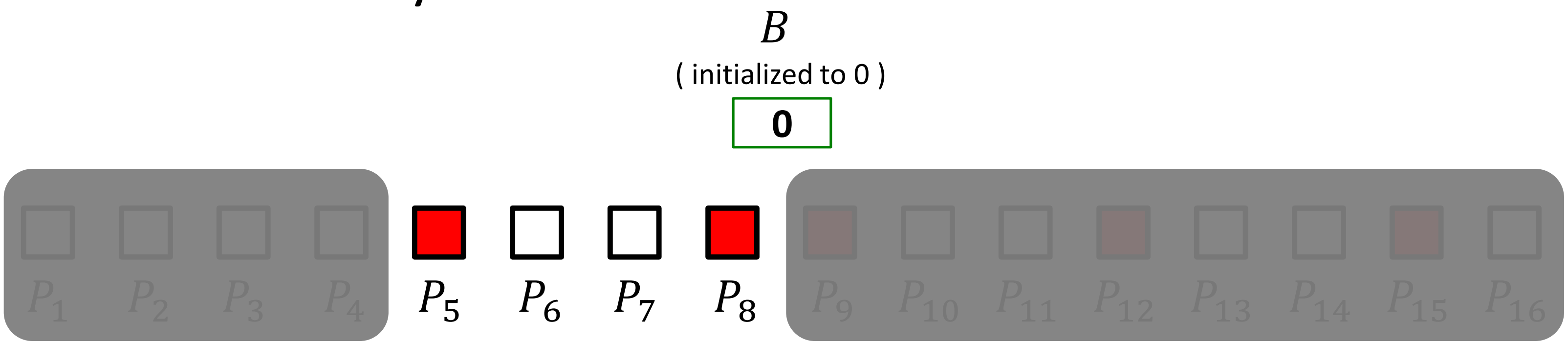


After stage 1: $B = 1$, and so processors $P_9, \dots, P_{16}$ are eliminated.

After stage 2: $B = 0$, and so processors $P_1, \dots, P_4$ are eliminated.

Concurrent Writes where the Leftmost Writer Wins

Solution: Use binary search.

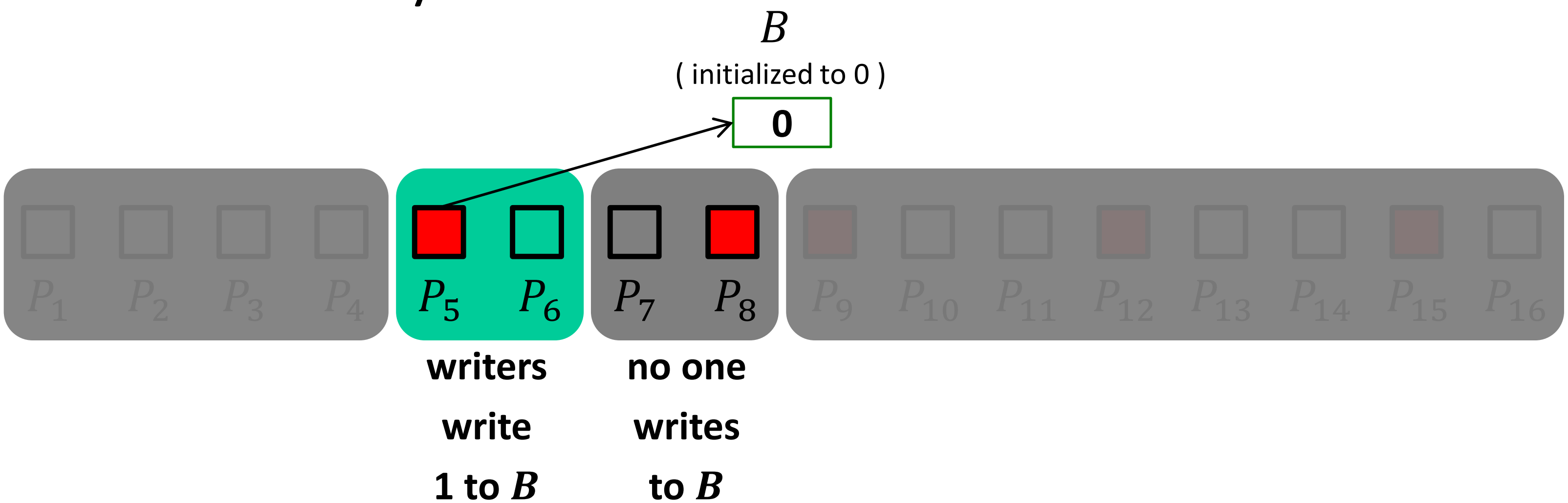


After stage 1: $B = 1$, and so processors $P_9, \ldots, P_{16}$ are eliminated.

After stage 2: $B = 0$, and so processors $P_1, \ldots, P_4$ are eliminated.

Concurrent Writes where the Leftmost Writer Wins

Solution: Use binary search.

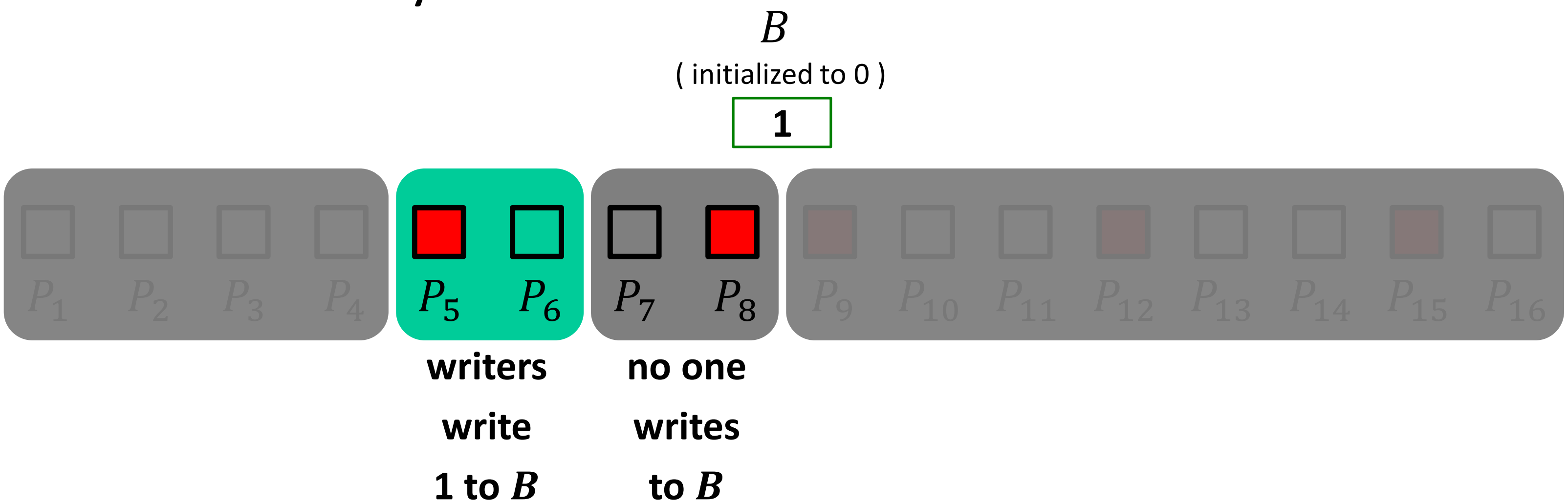


After stage 1: $B = 1$, and so processors $P_9, \dots, P_{16}$ are eliminated.

After stage 2: $B = 0$, and so processors $P_1, \dots, P_4$ are eliminated.

Concurrent Writes where the Leftmost Writer Wins

Solution: Use binary search.



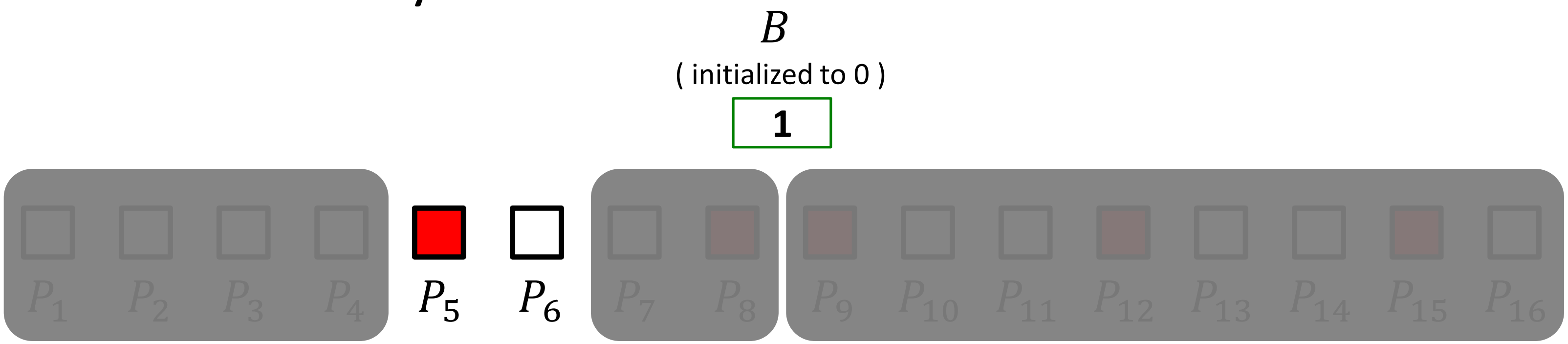
After stage 1: $B = 1$, and so processors $P_9, \ldots, P_{16}$ are eliminated.

After stage 2: $B = 0$, and so processors $P_1, \ldots, P_4$ are eliminated.

After stage 3: $B = 1$, and so processors P_7, P_8 are eliminated.

Concurrent Writes where the Leftmost Writer Wins

Solution: Use binary search.



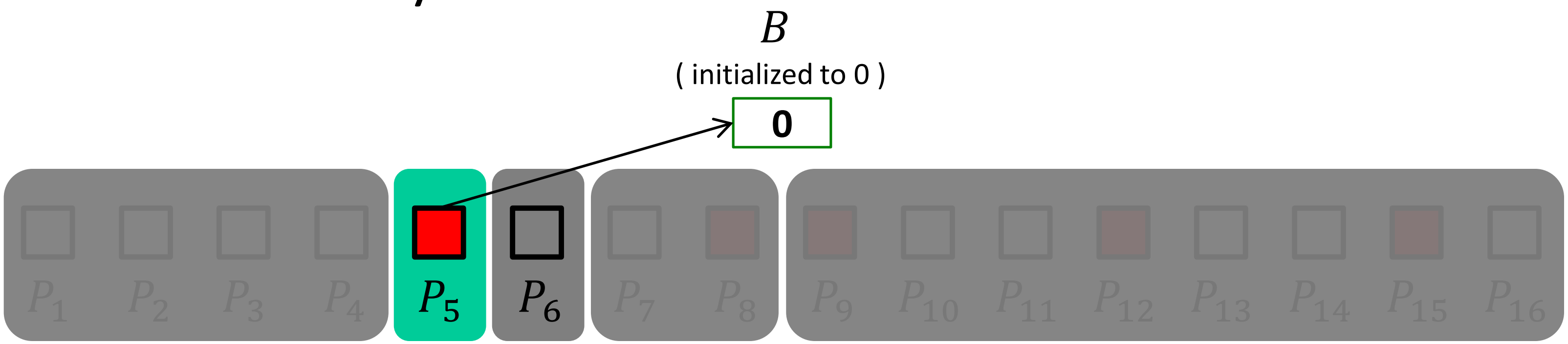
<u>After stage 1</u>: $B = 1$, and so processors $P_9, \ldots, P_{16}$ are eliminated.

<u>After stage 2</u>: $B = 0$, and so processors $P_1, \ldots, P_4$ are eliminated.

<u>After stage 3</u>: $B = 1$, and so processors P_7 and P_8 are eliminated.

Concurrent Writes where the Leftmost Writer Wins

Solution: Use binary search.



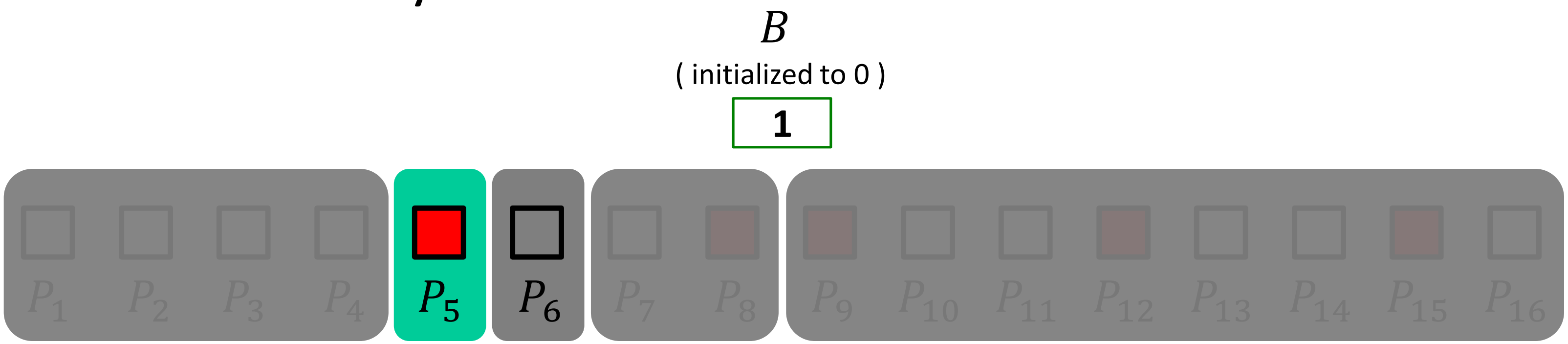
After stage 1: $B = 1$, and so processors $P_9, \dots, P_{16}$ are eliminated.

After stage 2: $B = 0$, and so processors $P_1, \dots, P_4$ are eliminated.

After stage 3: $B = 1$, and so processors P_7 and P_8 are eliminated.

Concurrent Writes where the Leftmost Writer Wins

Solution: Use binary search.



<u>After stage 1</u>: $B = 1$, and so processors $P_9, \ldots, P_{16}$ are eliminated.

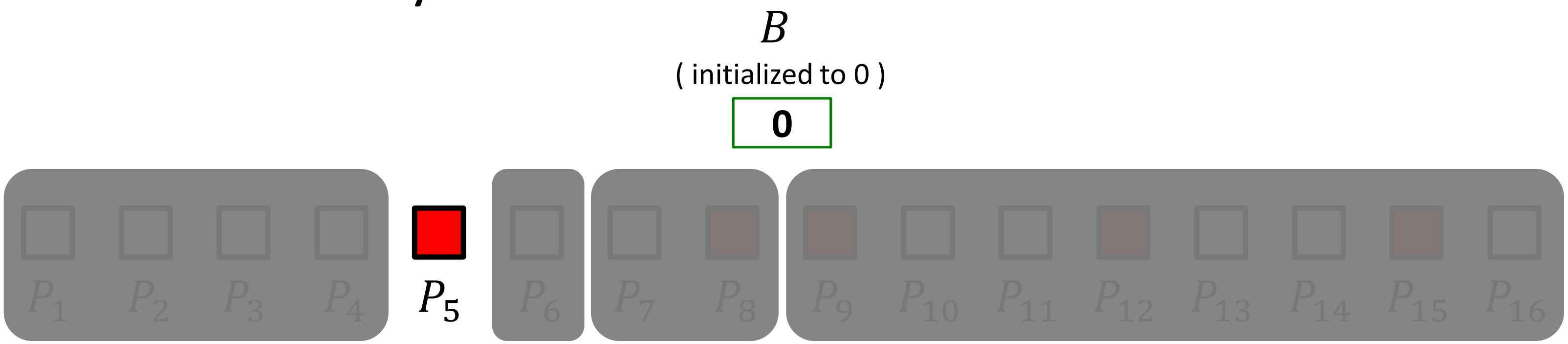
<u>After stage 2</u>: $B = 0$, and so processors $P_1, \ldots, P_4$ are eliminated.

<u>After stage 3</u>: $B = 1$, and so processors P_7 and P_8 are eliminated.

<u>After stage 4</u>: $B = 1$, and so processor P_6 is eliminated.

Concurrent Writes where the Leftmost Writer Wins

Solution: Use binary search.



After stage 1: $B = 1$, and so processors $P_9, \dots, P_{16}$ are eliminated.

After stage 2: $B = 0$, and so processors $P_1, \dots, P_4$ are eliminated.

After stage 3: $B = 1$, and so processors P_7 and P_8 are eliminated.

After stage 4: $B = 1$, and so processor P_6 is eliminated.

So processor P_5 is the leftmost writer.

Eliminating Priority Concurrent Writes from MST

Input: n is the number of vertices and E is the set of edges.

Output: For $1 \le u \le n$, $R[\ u\ ]$ is set to the smallest index i such that $E[\ i\ ].u = u$.

if $B[\ u\ ] = 1$ then the next active segment of u is set to the left half of its current active segment, otherwise it is set to the right half

```
Par-Simulate-Priority-CW-using-Binary-Search ( n, E, R )
array B[ 1 : n ], l[ 1 : n ], h[ 1 : n ], lo[ 1 : n ], hi[ 1 : n ], md[ 1 : n ],
parallel for u ← 1 to n do l[ u ] ← 1, h[ u ] ← |E|
for k ← 1 to 1 + log |E| do
    parallel for u ← 1 to n do B[ u ] ← 0, lo[ u ] ← l[ u ], hi[ u ] ← h[ u ]
    parallel for i ← 1 to |E| do
        u ← E[ i ].u, md[ u ] ← ⌊ ( lo[ u ] + hi[ u ] ) / 2 ⌋
        if i ≥ lo[ u ] and i ≤ md[ u ] then B[ u ] ← 1
    parallel for i ← 1 to |E| do
        u ← E[ i ].u, md[ u ] ← ⌊ ( lo[ u ] + hi[ u ] ) / 2 ⌋
        if B[ u ] = 1 and i ≥ lo[ u ] and i ≤ md[ u ] then h[ u ] ← md[ u ]
        elif B[ u ] = 0 and i > md[ u ] and i ≤ hi[ u ] then l[ u ] ← md[ u ] + 1
parallel for i ← 1 to |E| do
    u ← E[ i ].u
    if i = l[ u ] then R[ u ] ← i
```

for each $u \in [\ 1, n\ ]$, and each edge i with $E[\ i\ ].u = u$ and i in the left half of the current active segment for u, $B[\ u\ ]$ is set to 1

the leftmost edge i with $E[\ i\ ].u = u$ writes its index i to $R[\ u\ ]$

Eliminating Priority Concurrent Writes from MST

```
Par-Simulate-Priority-CW-using-Binary-Search ( n, E, R )
1.  array B[ 1 : n ], l[ 1 : n ], h[ 1 : n ],
          lo[ 1 : n ], hi[ 1 : n ], md[ 1 : n ]
2.  parallel for u ← 1 to n do l[ u ] ← 1, h[ u ] ← |E|
3.  for k ← 1 to 1 + log |E| do
4.      parallel for u ← 1 to n do
            B[ u ] ← 0, lo[ u ] ← l[ u ], hi[ u ] ← h[ u ]
5.      parallel for i ← 1 to |E| do
6.          u ← E[ i ].u, md[ u ] ← ⌊ ( lo[ u ] + hi[ u ] ) / 2 ⌋
7.          if i ≥ lo[ u ] and i ≤ md[ u ] then B[ u ] ← 1
8.      parallel for i ← 1 to |E| do
9.          u ← E[ i ].u, md[ u ] ← ⌊ ( lo[ u ] + hi[ u ] ) / 2 ⌋
10.         if B[ u ] = 1 and  i ≥ lo[ u ] and i ≤ md[ i ] then
                h[ u ] ← md[ u ]
11.         elif B[ u ] = 0 and  i > md[ i ] and i ≤ hi[ u ] then
                l[ u ] ← md[ u ] + 1
12. parallel for i ← 1 to |E| do
13.     u ← E[ i ].u
14.     if i = l[ u ] then R[ u ] ← i
```

The *parallel for loops* in lines 2 and 12 perform $O(m+n)$ work and have $\Theta(\log n)$ depth.

The *serial for loop* in line 3 iterates $\Theta(\log n)$ times with each iteration performing $\Theta(m+n)$ work in $\Theta(\log n)$ depth.

Work: $\Theta\big((n+m)\log n\big)$

Span: $\Theta(\log^2 n)$

Randomized Parallel MST with Priority CW

Input: *n* is the number of vertices, *E* is the set of edges, and *MST*[1: |*E*|] are flags with all of them initially set to 0. For every edge (*u*, *v*) both (*u*, *v*) and (*v*, *u*) are included in *E*.

Output: For all *i*, *MST*[*i*] is set to 1 if edge *E*[*i*] is included in the MST.

```
Par-Randomized-MST-Priority-CW ( n, E, MST )
array L[ 1 : n ], C[ 1 : n ], R[ 1 : n ]
sort the edges in E in non-decreasing order of edge weights
parallel for v ← 1 to n do L[ v ] ← v
F ← ( |E| > 0 ) ? True : False
while F = True do
    parallel for v ← 1 to n do C[ v ] ← RANDOM{ Head, Tail }
    parallel for i ← 1 to |E| do R[ E[ i ].u ] ← i ( priority: |E| − i )
    parallel for i ← 1 to |E| do
        u ← E[ i ].u, v ← E[ i ].v
        if C[ u ] = Tail and C[ v ] = Head and R[ u ] = i then
            L[ u ] ← v, MST[ i ] ← 1
    parallel for i ← 1 to |E| do E[ i ] ← ( L[ E[ i ].u ], L[ E[ i ].v ] )
    F ← False
    parallel for each ( u, v ) ∈ E do
        if u ≠ v then F ← True
```

Randomized Parallel MST w/o Priority CW

Input: n is the number of vertices, E is the set of edges, and MST[1: $|E|$] are flags with all of them initially set to 0. For every edge (u, v) both (u, v) and (v, u) are included in E.

Output: For all i, MST[i] is set to 1 if edge E[i] is included in the MST.

```
Par-Randomized-MST-Priority-CW ( n, E, MST )
array L[ 1 : n ], C[ 1 : n ], R[ 1 : n ]
sort the edges in E in non-decreasing order of edge weights
parallel for v ← 1 to n do L[ v ] ← v
F ← ( |E| > 0 ) ? True : False
while F = True do
    parallel for v ← 1 to n do C[ v ] ← RANDOM{ Head, Tail }
    Par-Simulate-Priority-CW-using-Binary-Search ( n, E, R )
    parallel for i ← 1 to |E| do
        u ← E[ i ].u, v ← E[ i ].v
        if C[ u ] = Tail and C[ v ] = Head and R[ u ] = i then
            L[ u ] ← v, MST[ i ] ← 1
    parallel for i ← 1 to |E| do E[ i ] ← ( L[ E[ i ].u ], L[ E[ i ].v ] )
    F ← False
    parallel for each ( u, v ) ∈ E do
        if u ≠ v then F ← True
```

Randomized Parallel MST w/o Priority CW

```
Par-Randomized-MST-Priority-CW ( n, E, MST )
 1.  array L[ 1 : n ], C[ 1 : n ], R[ 1 : n ]
 2.  sort the edges in E in non-decreasing
     order of edge weights
 3.  parallel for v ← 1 to n do L[ v ] ← v
 4.  F ← ( |E| > 0 ) ? True : False
 5.  while F = True do
 6.      parallel for v ← 1 to n do
             C[ v ] ← RANDOM{ Head, Tail }
 7.      Par-Simulate-Priority-CW-using-Binary-Search ( n, E, R )
 8.      parallel for i ← 1 to |E| do
 9.          u ← E[ i ].u, v ← E[ i ].v
10.          if C[ u ] = Tail and C[ v ] = Head and R[ u ] = i then
11.              L[ u ] ← v, MST[ i ] ← 1
12.      parallel for i ← 1 to |E| do
             E[ i ] ← ( L[ E[ i ].u ], L[ E[ i ].v ] )
13.      F ← False
14.      parallel for each ( u, v ) ∈ E do
15.          if u ≠ v then F ← True
```

Let n = #vertices, and m = #edges in original graph. Then $m \geq n - 1$ as graph is connected.

Expected number of contraction steps, $D = \mathrm{O}(\log n)$.

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

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

$$= \Theta(m \log^2 n)$$

Span:

$$T_\infty(n,m) = \Theta(\log^3 n + D\log^2 n)$$
$$= \Theta(\log^3 n)$$

Parallelism: $\frac{T_1(n,m)}{T_\infty(n,m)} = \Theta\left(\frac{m}{\log n}\right)$