A Randomized Concurrent Algorithm for Disjoint Set Union

SIDDHARTHA V. JAYANTI
ROBERT E. TARJAN
PRINCETON UNIVERSITY
Set Union Data Type

Maintain \( n \) nodes in **disjoint sets**

**Unite**\((x, y)\) – update operation

**SameSet**\((x, y)\) – query operation

*Initially nodes are in singleton sets.*
Applications

- FORTRAN compilers: COMMON and EQUIVALENCE statements
- Incremental **connected components**
- Finding dominators in flow graphs
- **Spanning tree / forest**
- Percolation
- Strongly Connected Components
- Model Checking

[Diagram of spanning trees and finding strongly connected components]
Road Map

• Review famous sequential algorithm

• Previous Attempt at Concurrent Data Structure

• Our Concurrent Data Structure
Implementation
[Galler and Fischer]

Set = Rooted Tree with parent pointers

Primitives:

\textbf{Link}(b, x): make \ b\.parent = x
\textbf{or} \ x\.parent = b

\textbf{Find}(c): return root of c’s tree
\textbf{(returns $x$ in this case)}
Implementation

**SameSet**($x, y$):

\[
\begin{align*}
u &= \text{Find}(x) \\
v &= \text{Find}(y) \\
\text{return} \ (u = v)
\end{align*}
\]

**Unite**($x, y$):

\[
\begin{align*}
u &= \text{Find}(x) \\
v &= \text{Find}(y) \\
\text{if} \ (u \neq v) \ \text{Link}(u, v)
\end{align*}
\]
Linking by Rank

Link(b, c)
Link (a, b)
Link (e, f)
Link(d, e)
Link(b, e)
Find

**Find**(a)

*return e*

Find Sequence
Find with Splitting

**SplittingFind(a)**

*return e*
Ackermann’s Function

• $A_k(n)$ – highly super-exponential function

• $A_4(2)$ more than number of particles in observable universe

• $\alpha(n, d) = \min\{k > 0 \mid A_k(d) > n\}$

• $\alpha(n, d)$ is practically bounded by 4
<table>
<thead>
<tr>
<th>Find with Splitting</th>
<th>Linking by Rank</th>
<th>Amortized Time per Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
</tbody>
</table>

$m$ – number of operations, $n$ – number of nodes
Computational Model

Asynchronous Shared Memory Machine

Shared Memory

\( \mathbf{p}_1 \) \hspace{1cm} \mathbf{p}_2 \hspace{1cm} \mathbf{p}_3 \hspace{1cm} \ldots \hspace{1cm} \mathbf{p}_i \hspace{1cm} \ldots \hspace{1cm} \mathbf{p}_{k-1} \hspace{1cm} \mathbf{p}_k \\
Local Memory \hspace{2cm} Local Memory \hspace{2cm} Local Memory \hspace{1cm} Local Memory \hspace{1cm} Local Memory \hspace{1cm} Local Memory
Compare and Swap

procedure CAS(x, x₀, x₁)
2:   if x = x₀ then
3:     x ← x₁
4:     return true
5: else
6:     return false
Correctness Criteria

**Linearizability** [Herlihy, Wing 1990]:

Each \( p_i \) should complete operation in a bounded number of its steps.

**Wait-Freedom** [Herlihy 1991]:

Each \( p_i \) should complete operation in a bounded number of its steps.
Work

• $W_j = \text{number instructions executed by } p_j$

• Total work $\quad W = \sum_{j=1}^{k} W_j$

• For sequential algorithm, work = time
Previous Algorithm
[Anderson and Woll, 1991]

Extends linking by rank algorithm

\( n \) nodes
\( m \) operations
\( p \) processes

Amortized work per operation \( \Theta(\alpha(m, 1) + p) \)?

Hard to maintain both rank and parent

Per operation work is linear in \( p \)
Linking by ID
[Goel, Khanna, Larkin, Tarjan 2014]

• The nodes are given IDs 1, 2, \ldots, n uniformly at random

• Link winner determined by **fixed ID** rather than **changing rank**.

```
Link(4, 2)
Link(3, 4)
Link(6, 1)
Link(5, 6)
Link(4, 6)
```
### Time Complexity

[Goel, Khanna, Larkin, Tarjan 2014]

<table>
<thead>
<tr>
<th>Find with Splitting</th>
<th>Linking by ID</th>
<th>Expected-amortized Time per Op</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>✓</td>
<td>( \Theta(n) )</td>
</tr>
<tr>
<td></td>
<td>✓ ✓</td>
<td>( \Theta(\log n) )</td>
</tr>
<tr>
<td>✓</td>
<td>✓ ✓</td>
<td>( \Theta\left(\log_1^{\frac{m}{n}} n\right) )</td>
</tr>
<tr>
<td>✓</td>
<td>✓ ✓</td>
<td>( \Theta\left(\alpha\left(n, \frac{m}{n}\right)\right) ) (optimal in cell-probe model)</td>
</tr>
</tbody>
</table>

The same results in expectation!
Concurrent Link($u, v$)

\begin{align*}
\text{Link}(u, v) & \quad \text{if } (v < u) \text{ swap}(u, v) \\
& \quad \text{return CAS}(u.\text{parent}, u, v)
\end{align*}

- CAS succeeds iff $u$ is a root
- $v$ is possibly not a root
Concurrent Find(x)

\[ \text{Find}(x) \]
\[
\begin{align*}
\text{u} &= \text{x} \\
\text{while (u not root)} &\ast \\
\text{v} &= \text{u.parent}, \text{w} = \text{v.parent} \\
\text{CAS(u.parent, v, w)} \\
\text{u} &= \text{v} \\
\text{return u}
\end{align*}
\]
Difficulty with Parallelization

Computation can be invalidated

\[ p_1 \]
SameSet(1, 2)
\[ p_2 \]
Unite(3, 7)

Root of 1 is 3
Root of 2 is 7
return false
Unite Implementation

Unite(x, y)
    \[ u = \text{Find}(x) \]
    \[ v = \text{Find}(y) \]

    if \( u = v \), return false

    if Link(u, v)*, return true

TRY AGAIN (occurs at most n times)
Unite Implementation

Links done by \textit{CAS}

- \textcolor{cyan}{p_1} \hspace{5mm} \text{Unite}(5, 6)
- \textcolor{magenta}{p_2} \hspace{5mm} \text{Unite}(4, 8)
SameSet Implementation

SameSet(x, y)

    u = Find(x)
    v = Find(y) *

    if (u = v), return true

    else if (u still root), return false

TRY AGAIN (occurs at most n times)
Problem Fixed

$p_1$ SameSet(1, 2)

$p_2$ Unite(3, 7)

Root of 1 is 3
Root of 2 is 7
3 not root!
return true
Main Theorem

$m$ operations, $n$ nodes, $p$ processors

Expected-amortized work per operation

$$\Theta\left(\alpha \left(\frac{m}{n} \log \left(\frac{np}{m} (p+1)\right)\right)\right)$$

*assuming ID order and linearization order are independent*
Main Theorem Part 2

$m$ operations, $n$ nodes, $p$ processors

Worst-case work per operation \textbf{whp} \hspace{1cm} \mathcal{O}(\log n)

*assuming ID order and linearization order are independent*
Current State-of-the-Art

- Randomized algorithm with same efficiency under **no assumption**

- **Deterministic algorithm** (only a loglog $p$ extra overhead!)

- We think work bound is **optimal**, we have shown a lower bound:

  $$\Omega \left( \alpha \left(n, \frac{m}{n}\right) + \log\log \left( \frac{np}{m} + 1 \right) \right)$$
Thanks!
Upper Bound Proof Idea

• Define $d = \frac{m}{np}$

• If $d > 1$, extend sequential analysis

• If $d < 1$

  - Use $d > 1$ argument

  - $O(\log p)$ height
Lower Bound Example

• Let us consider the case $\Theta(m) = \Theta(n) = \Theta(p)$

• Perform $SameSet(x, x)$ with each processor (random $x$)

• Per operation work = $\log p$
Illustration of our Solution
Correctness Criteria

**Linearizability** [Herlihy, Wing 1990]:
Each operation appears to take effect instantaneously at some point between its invocation and return.

**Wait-Freedom** [Herlihy 1991]:
Each process completes each operation in a finite number of its own steps.
Find with Compression

`CompressFind(a)`

*return e*

- 2 traversals
- not local
# Time Complexity

[Goel, Khanna, Larkin, Tarjan 2014]

<table>
<thead>
<tr>
<th>Find with Splitting</th>
<th>Linking by randomized ID</th>
<th>Expected Time per Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
</tbody>
</table>

The *same efficiency results carry over* in Expectation!
Linking by Rank

0 0 0 0 0 0
Correctness Criteria

**Linearizability** [Herlihy, Wing 1990]:

**Wait-Freedom**: Each $p_i$ should be able to complete its operation in a bounded number of its own steps.
Goal

Algorithm with work sub-linear in $p$. 
Approach

• Use linking by ID instead

• Only parent pointers change in this case