The Data Locality of Work Stealing

Authors: Umut A. Acar, Guy E. Blelloch, Robert D. Blumofe
Presented by: Omar Obeya

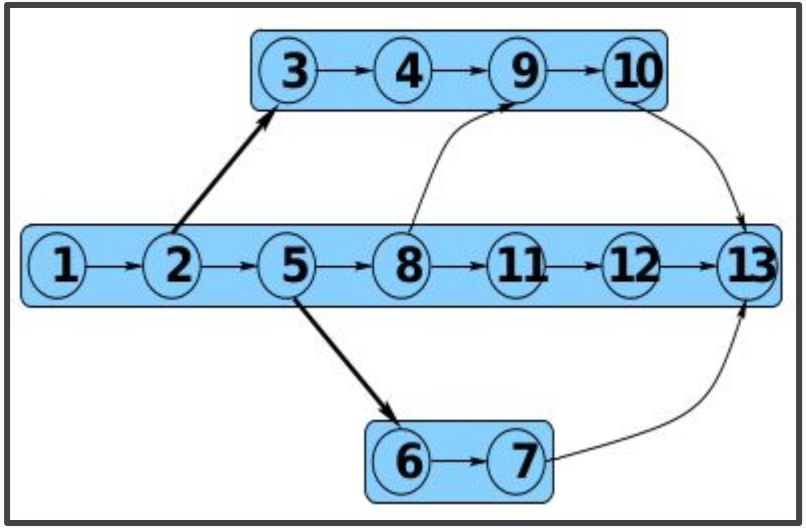
Goal

1. Studying the effect of races on cache misses.
2. Studying the effect work steals have on cache misses.
3. Designing and implementing efficient tools to improve data locality while allowing work stealing.

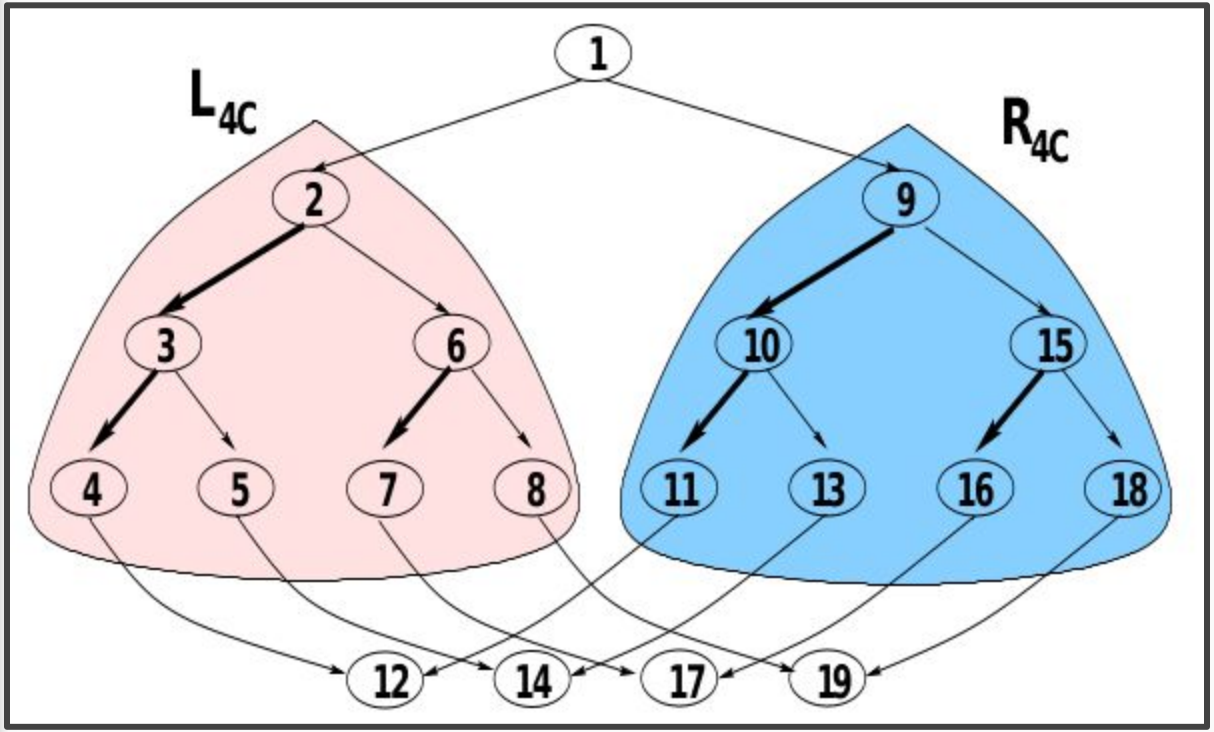
Model

- Represent Graphs using DAG
  - Series-Parallel Computation
  - Nested-Parallel Computation

- Simple Cache Replacement Policy
  - Deterministic
  - Cache replacement of a cache line is only a function of information after last access to line.



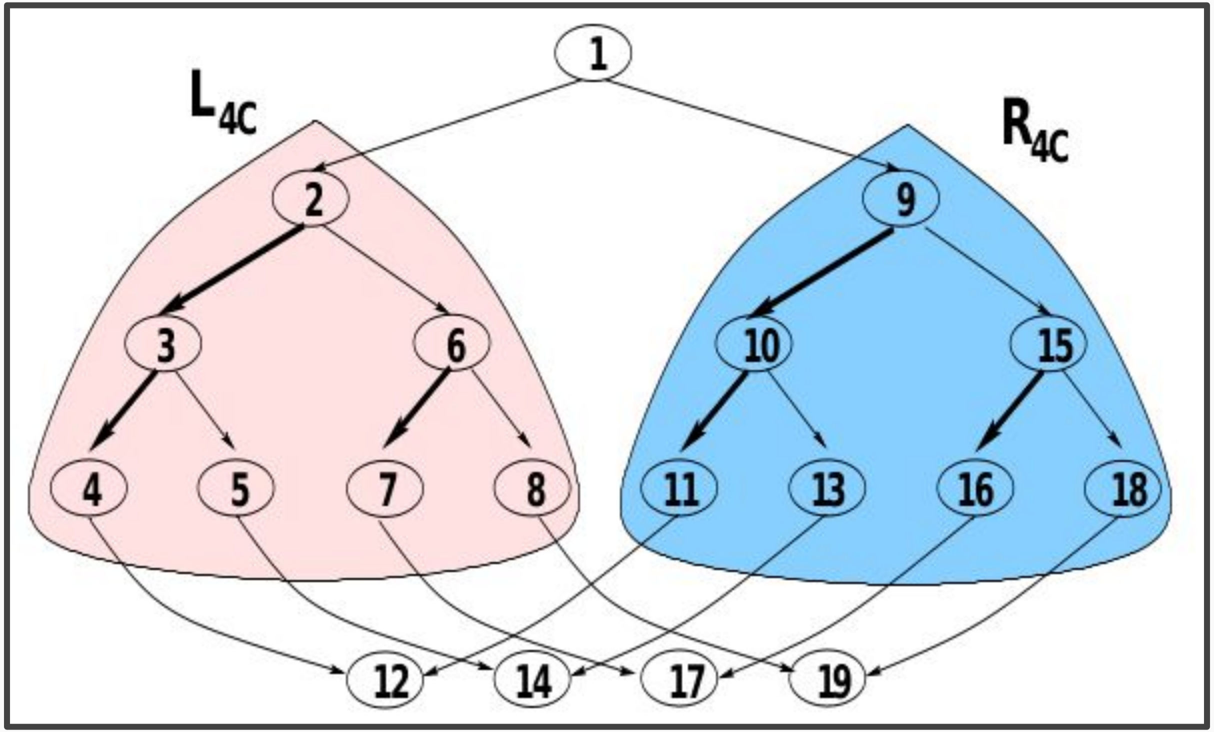
General Computation Example



General Computation Example

- Graph G_{4C}
  - Root
  - L_{4C} in red
  - R_{4C} in blue
  - 4 merge nodes.

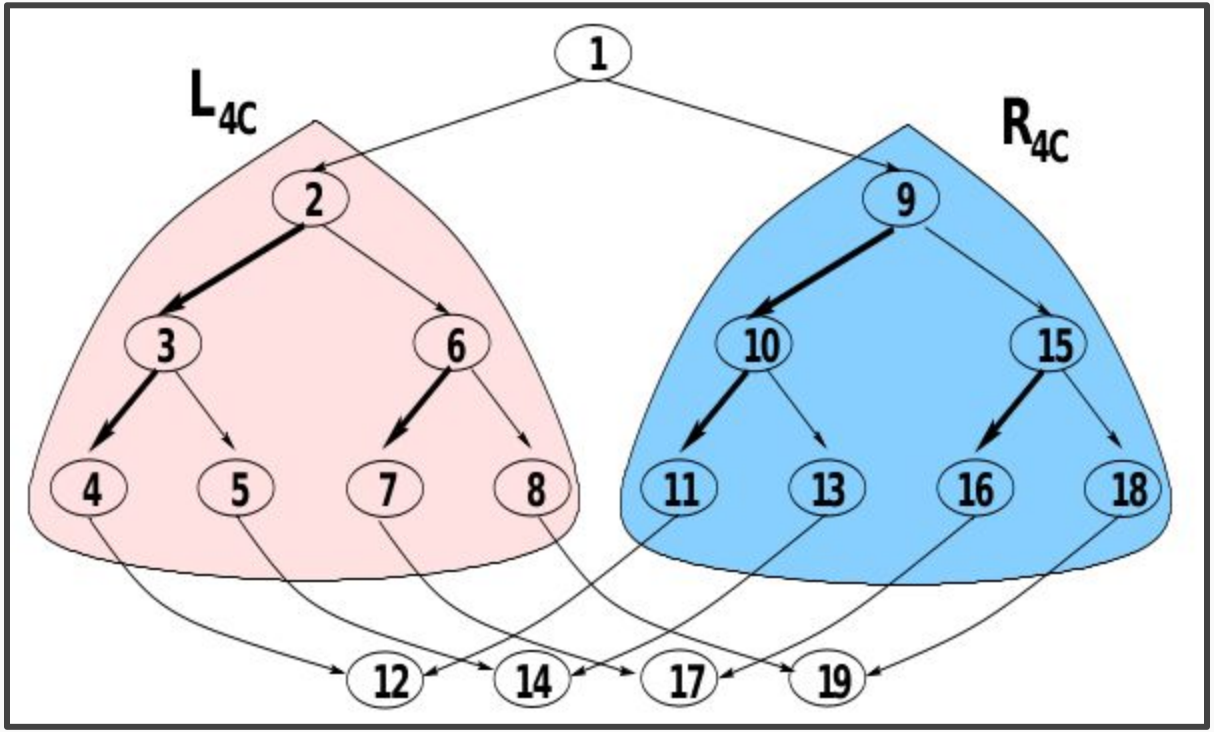
- Cache access
  - C cache access per node.
  - Three groups of cache
    - Root
    - L_{4C}
    - R_{4C} + 4 nodes



General Computation Example

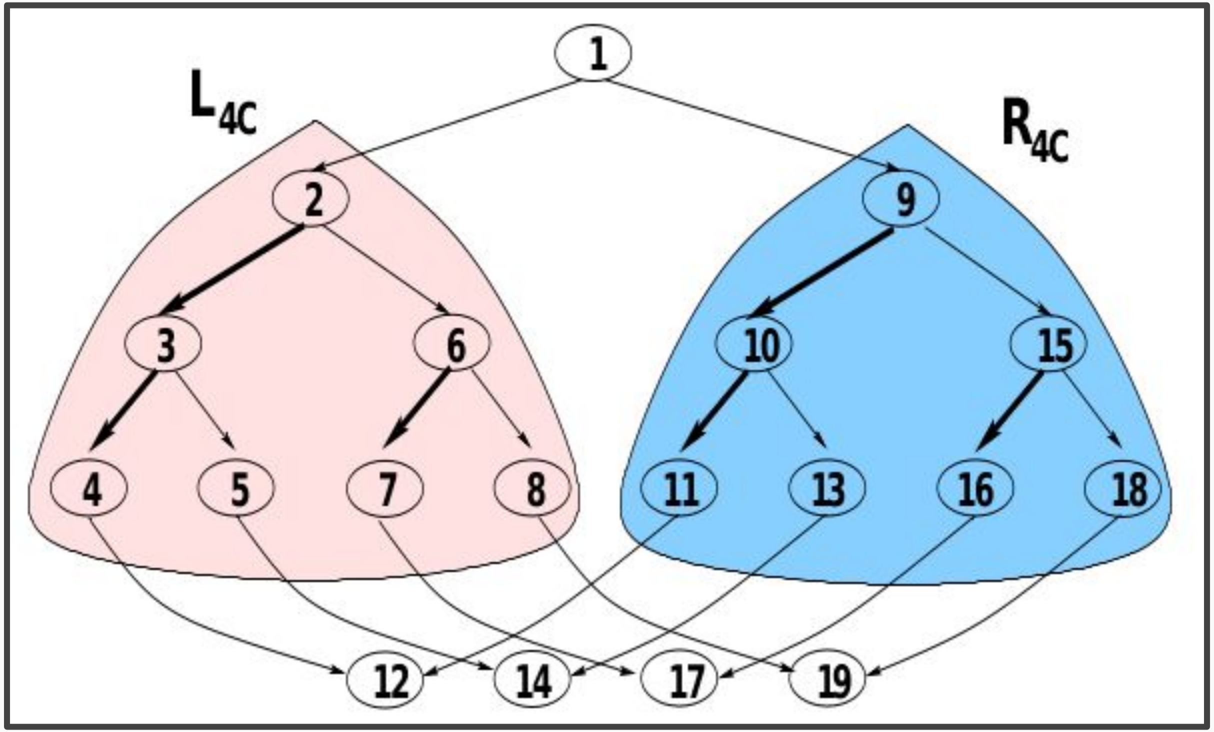
- Serial Execution
  - Root
  - L_{4C}
  - R_{4C} + merger nodes

- Cache misses
  - Root: C
  - L_{4C}: C
  - R_{4C}+ merger nodes: C
  - Total: 3C cache misses!



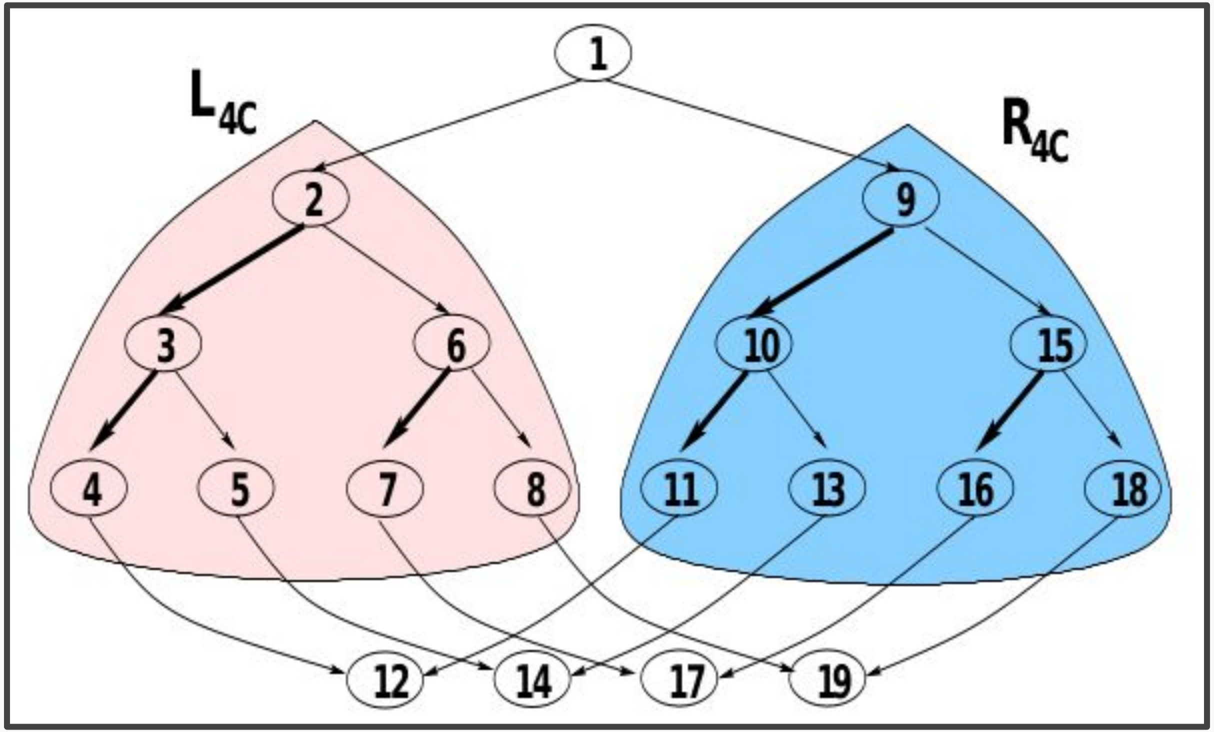
General Computation Example

- 2-Core Execution
  - Root (Core 0)
  - L_{4C} (Core 0) and R_{4C} (Core 1)
  - Merger nodes (Core 0)
- Problem
  - R_{4C} and merger nodes are accessing same data but executed by different cores.

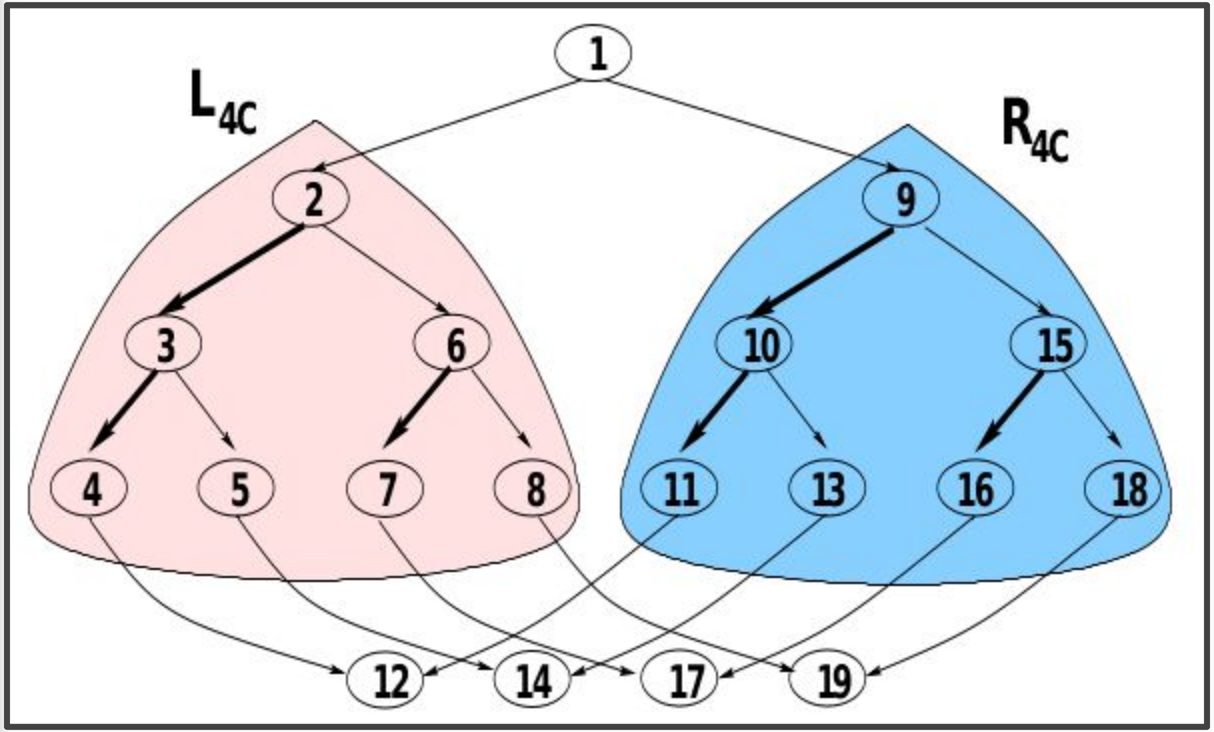


General Computation Example

- 2-Core Cache misses
  - Root: C
  - L_{4C}: C
  - R_{4C}: C
  - Merger nodes: 4C!!!
  - Total: 7C cache misses!
  - Overhead: 4C
  - Overhead is independent from serial cache misses.



Nested-Parallel Computations



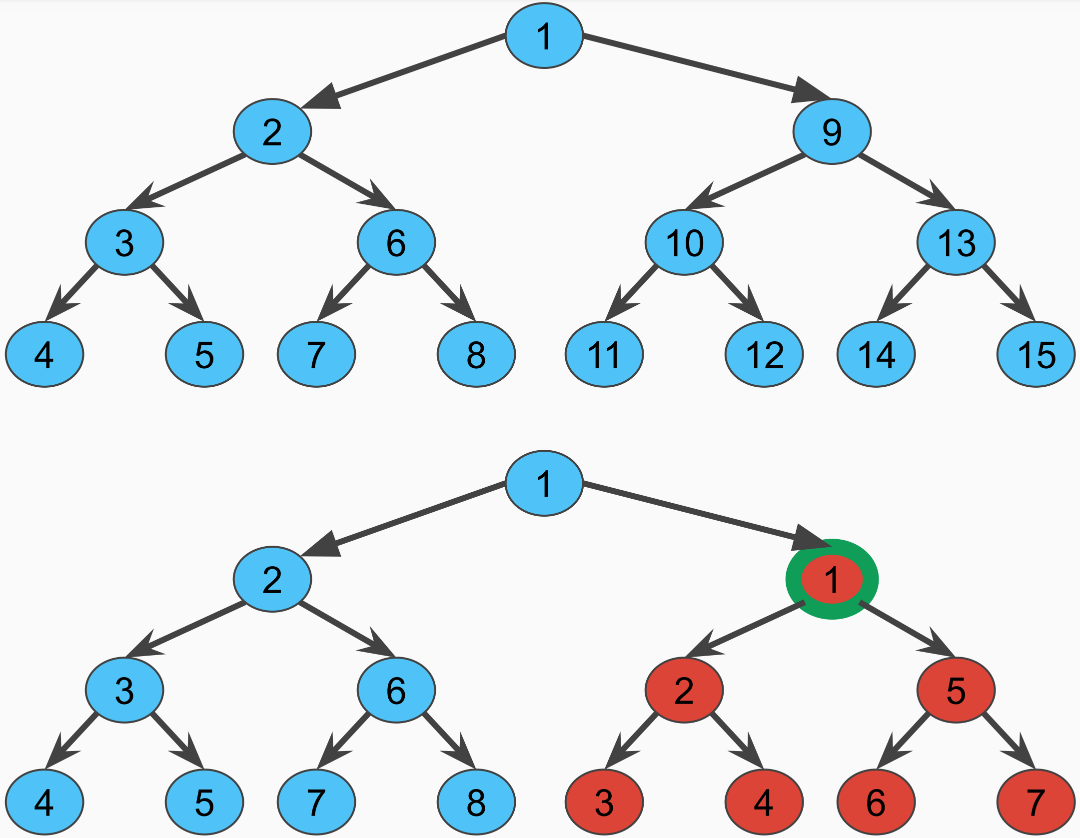
Race and Caches

- Races
    - Write-Write dependency: can cause the situation in the previous slide.
    - Write-Read dependency: a thread write will invalidate the other thread's cache.

Drifted Nodes

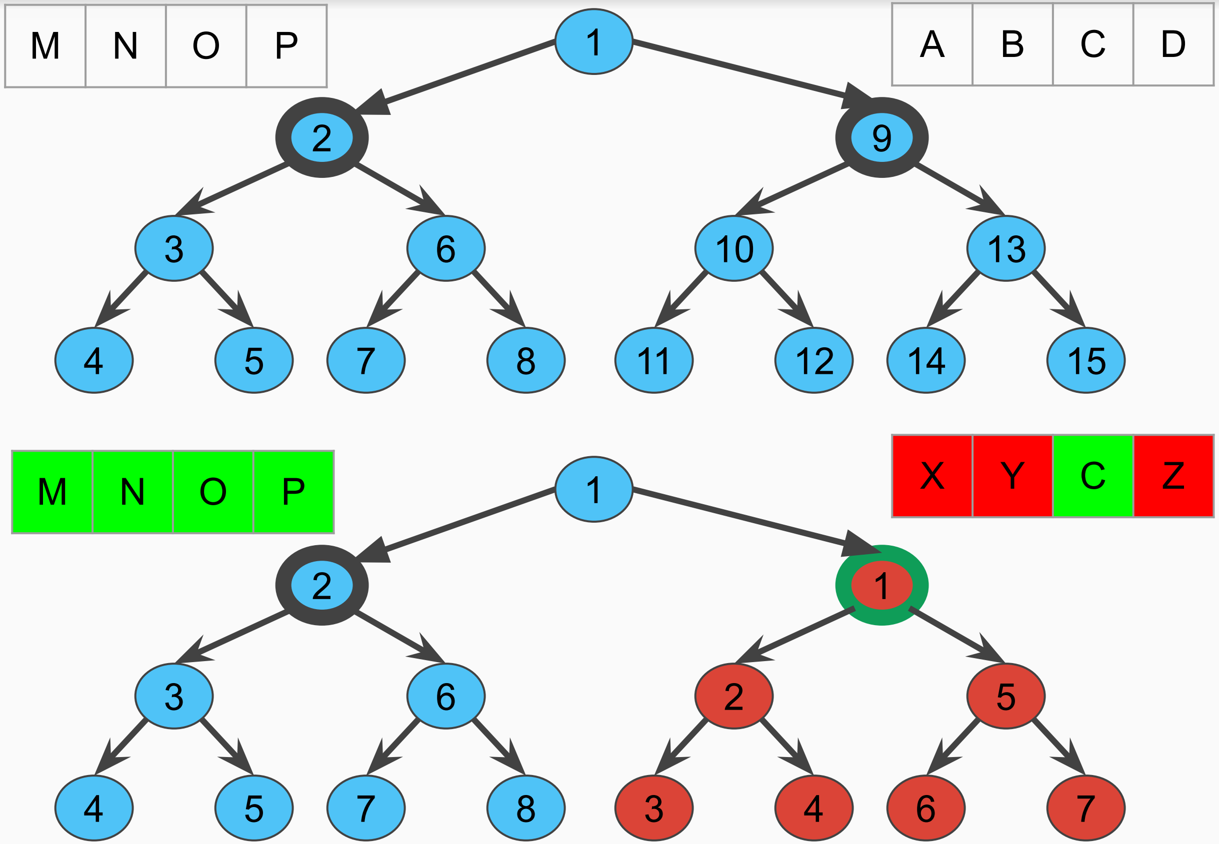
- Drifted Node

A node that has a different predecessor in the parallel execution than the serial execution.



Drifted Nodes and cache

- Simple cache policy is function of cache state and cache access.
- If two execution start at the same node and perform same access, then, they can differ by at most C cache misses.



Conclusions about parallel nested computation

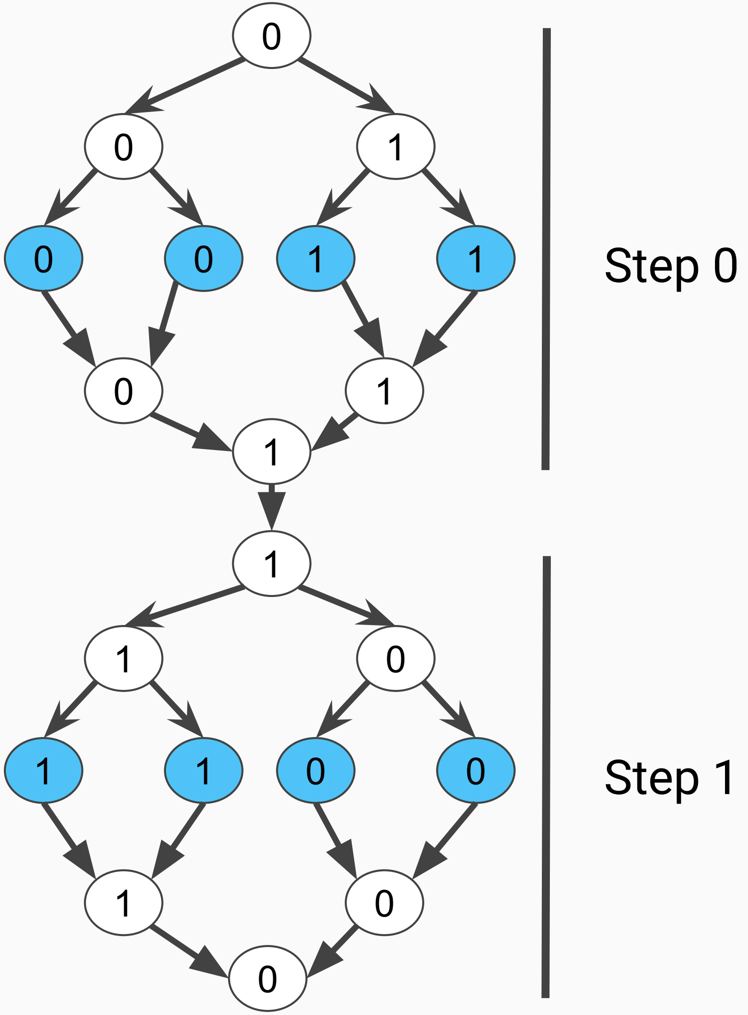
- Total no. of cache misses overhead of a nested parallel algorithm is:
  C * no. of drifted nodes.

- Total no. of drifted nodes is upper bounded by twice no. of steals.

- Expected No. of overhead cache misses on P processors is $O(\lceil m/s \rceil * C * P * \text{span})$, where m is the execution time of an instruction incurring a cache miss and s is the steal time.

Iterative Data-Parallel Application

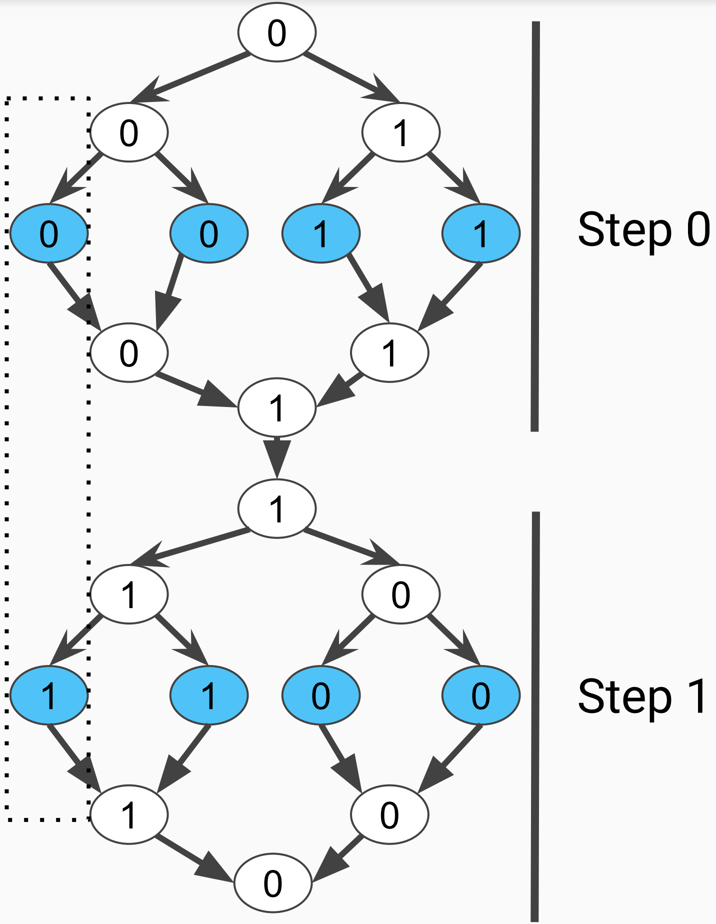
```
for(int step = 0; step < 2; step ++) {
  Parallel_for(int i =1; i < n-1; i++) {
      A[i] = (A[i-1] + A[i] + A[i+1])/3;
  }
}
```



Iterative Data-Parallel Application

Problem

Same data accessed by different processors in different steps.



Typical Work Stealing

Each process x maintains one deque (double-ended queue), such that:

- When x spawns a new independent task, it pushes it on the bottom of the deque.
- When x is done with its current task, it pops a task from the bottom of the deque.
- When a process is idle and has an empty deque, it steals a task from the top of another random process deque.

Prioritized Work Stealing: Mailbox

Each process x maintains a mailbox besides its regular deque such that:

- x’s mailbox is a FIFO queue containing threads with affinity to x.
- When x creates a thread, it pushes it to both the deque and the mailbox.
- When x is idle, it tries to pop a task from the mailbox first, if it failed, it tries the deque, if both fail, it tries stealing.
- Some mechanism is needed to maintain consistency between mailbox and deque.

Ropes

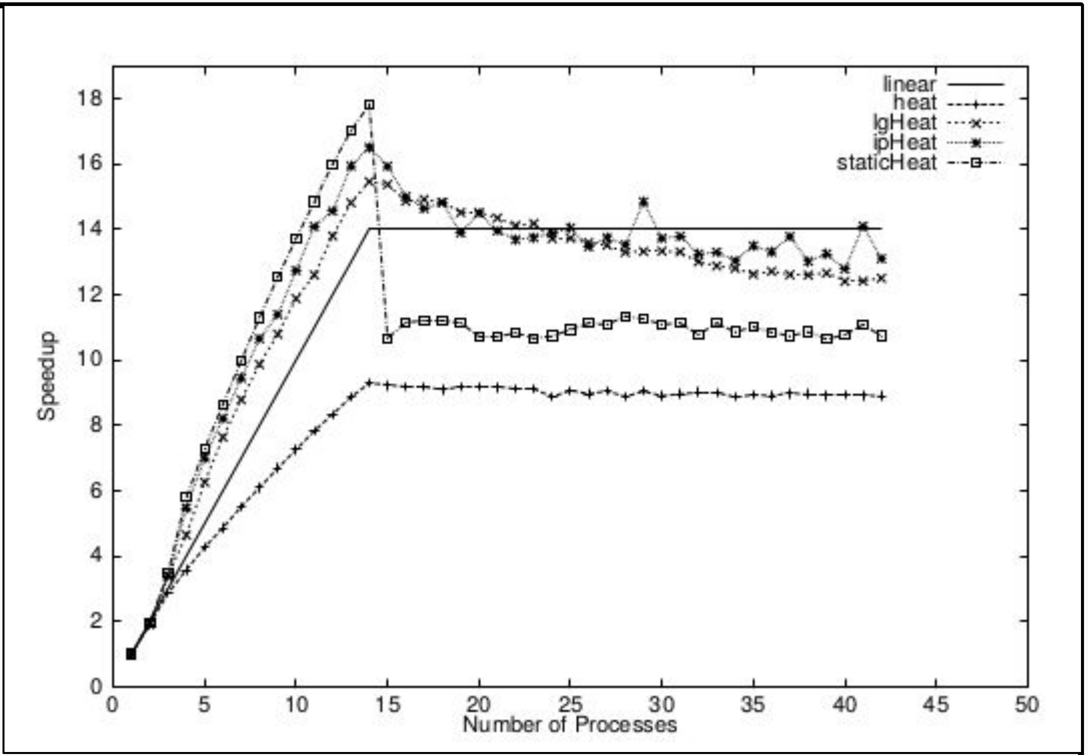
Each rope:

- corresponds to a subarray.
- Has an affinity to a process.
- Puts corresponding thread in the correct mailbox.
- If the corresponding thread got stolen, the robe is updated with a new process.

Take away: Ropes increase the likelihood that same data are accessed by the same process at each step in a dynamic fashion that does not harm load balance.

Implementations

- Static partitioning (static)
  - Bad load balancing.
  - Perfect locality.
- Work stealing (none)
  - Good load balancing.
  - Bad locality.
- Work stealing with ropes (lg)
  - Good load balancing
  - Good locality
- Work stealing with ropes with initial placements (ip)
  - Worse load balancing
  - Better locality



80%

Improvement over work regular work stealing

Conclusion

- Contributions
  - Theoretical
    - Lower bound on worst case cache overhead of general computation series-parallel parallelism.
    - Upper bound on worst case cache overhead of nested-parallel computations
  - Practical
    - Ropes and mailboxes to improve data locality of work stealing.

Thanks!
Questions?