Helper Locks for Fork-Join Parallel Programming

Kunal Agrawal
kunal@cse.wustl.edu
Washington University
at St. Louis

Charles E. Leiserson
cel@mit.edu
MIT Computer Science and
Artificial Intelligence Laboratory

Jim Sukha
sukhaj@mit.edu
MIT Computer Science and
Artificial Intelligence Laboratory

Locks and Nested Parallelism

Fork-join languages such as Cilk [2] do not efficiently exploit parallelism nested inside a locked critical section. For example, consider the following code which repeatedly performs insertions into a resizable hash table **H**. This code uses a reader/writer lock to prevent inserts from interfering with a resize. Although the resize itself (which rebuilds the table) can be parallelized, Cilk may still execute the resize serially, since processors waiting to insert block instead of helping to complete the resize.

```
void hash_insert(HashTable* H, int key) {
  acquire(H->resize_lock, READER);
  int b = hashcode(key);
  locked_bucket_insert(H->buckets[b], key);
  release(H->resize_lock);
  if (overflow(H))
    resize_table(H);
}
```

```
void resize_table(HashTable* H) {
  acquire(H->resize_lock, WRITER);
  Buckets* newB = create_buckets(2*H->size);
  parallel_for b in H->buckets:
    rehash_bucket(H->buckets[b]);
  H->buckets = newB;  H->size = 2*H->size;
  release(H->resize_lock);
}
```

```
void parallel_inserts(HashTable* H, int n) {
  parallel_for k in 1:n
    hash_insert(H, rand());
}
```

Helper Locks

A **helper lock L** behaves like an ordinary lock, except that it is connected to a **parallel region R**. A processor that fails to acquire a helper lock L tries to help complete the parallel region connected to **L** [1].

To use helper locks for a resizable hash table:

- Specify a table resize function as a parallel region **R**.
- Call a resize by starting a region, protected by a helper lock **L**.
- An insert that fails to acquire **L** tries to help complete a resize region **R** if there is an **R** currently holding **L**.

To support helper locks, we add two constructs to Cilk, the `start_region` and `help_region` constructs.

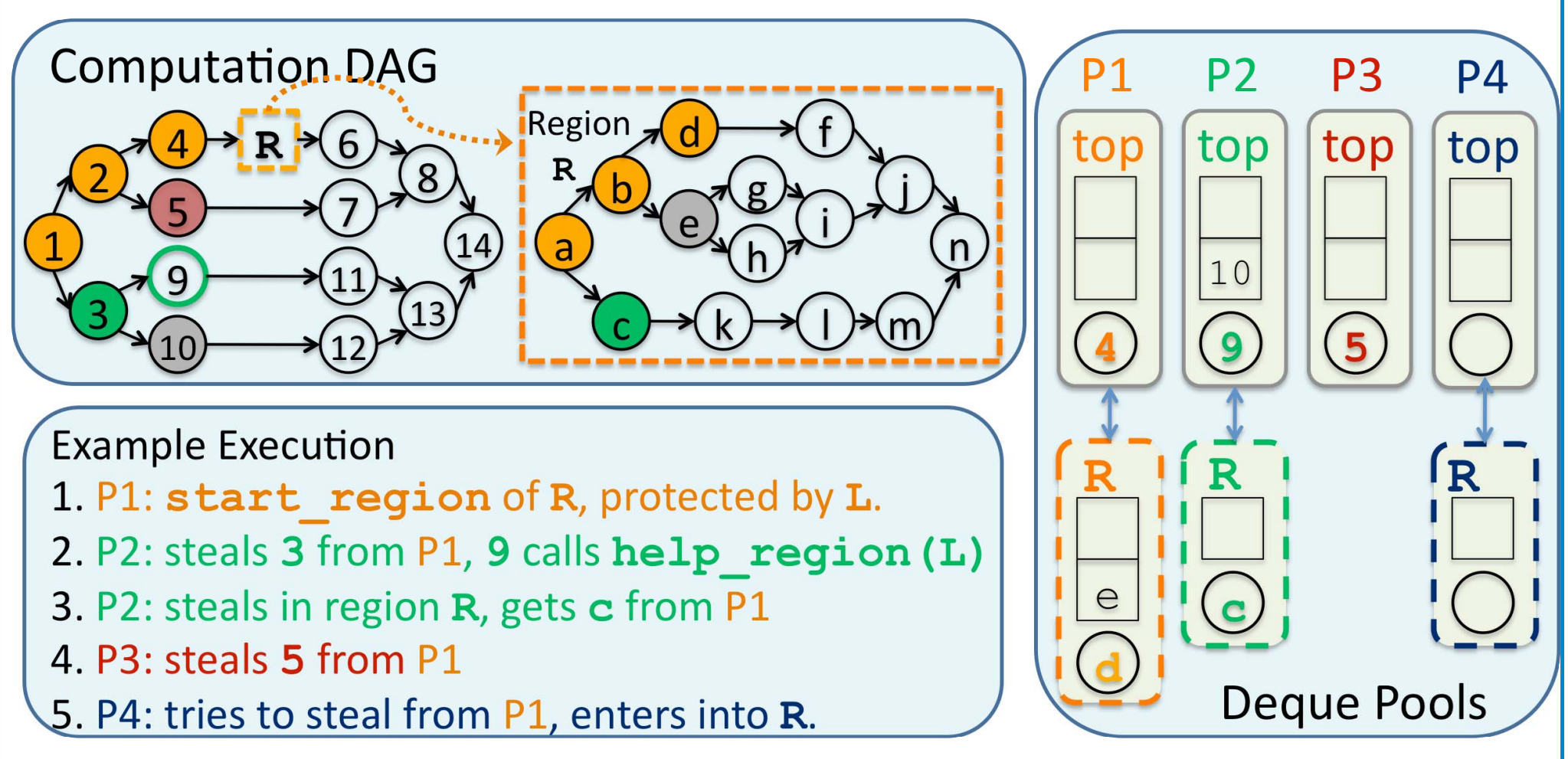
```
void hash_insert(HashTable* H, int key) {
  ...
  if (!acquired_read_lock(L)) {
     help_region(L);
  }
  ...
  if (overflow(H))
   start_region(resize_table)(L);
   sync;
}
```

Enter and help complete the region protected by **L**.

Starts `resize_table` as a parallel region, protected by the lock L.

HELPER Runtime

In [1] we present ***HELPER***, a runtime system that supports helper locks in Cilk. HELPER supports parallel regions by creating a separate **deque pool** for every region **R**. A deque pool contains up to P deques, one for each worker thread used to execute a program. A worker thread enters a region **R** and creates a deque for **R** in three ways, by (1) starting region **R**, (2) helping with **R**'s lock, or (3) through random work-stealing.



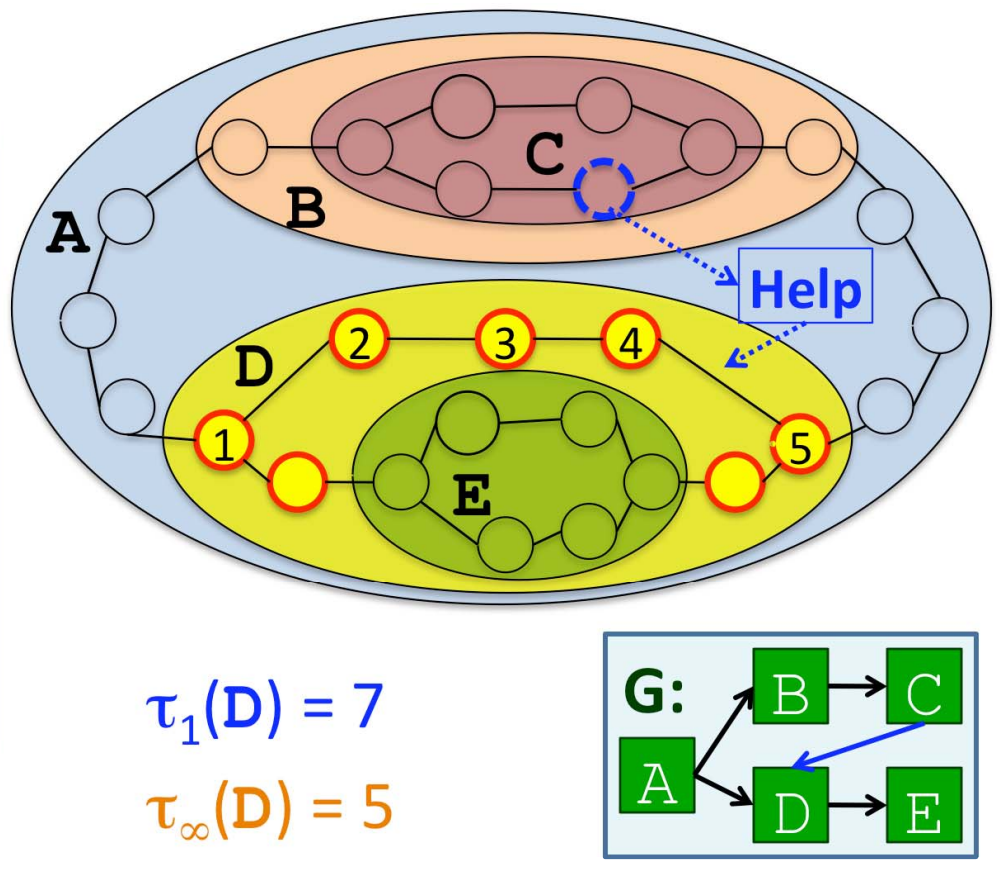
Example Execution
1. P1: `start_region` of **R**, protected by **L**.
2. P2: steals **3** from P1, **9** calls `help_region(L)`
3. P2: steals in region **R**, gets **c** from P1
4. P3: steals **5** from P1
5. P4: tries to steal from P1, enters into **R**.

Bounds on Time and Space

One can prove time and space bounds for programs using HELPER.

Definitions:

- $\tau_1(\mathbf{D})$: Region work of **D** (# of nodes, ignoring nested regions)
- $\tau_\infty(\mathbf{D})$: Region span of **D** (longest path ignoring nested regions)
- G = (V, E): Region Dag: Let (R_1, R_2) be an edge in G if region $\mathbf{R_1}$ starts $\mathbf{R_2}$ or helps for $\mathbf{R_2}$'s lock.
- $S_1(\mathbf{D})$: Stack space needed to execute **D** serially, ignoring nested regions.



$\tau_1(\mathbf{D}) = 7$

$\tau_\infty(\mathbf{D}) = 5$

Theorem 1: HELPER can execute programs in expected time:

$$O\left(\sum_{\text{all regions } \mathbf{R}} \left(\frac{\tau_1(\mathbf{R})}{P} + \tau_\infty(\mathbf{R})\right) + \underbrace{E \lg(1+PV/E)}\right)$$

Sufficient condition for linear speedup with HELPER:
Every region **R** has
$\tau_1(\mathbf{R}) / (\tau_\infty(\mathbf{R}) + P) \gg P$.

Best case: If E = V, term is O(V lg P).
(no `help_region` calls)
Worst case: If E = PV, term is O(PV).

Theorem 2: HELPER stack space usage is $O\left(P \sum_{\text{all regions } \mathbf{R}} S_1(\mathbf{R})\right)$.

Future Work

We have modified Cilk to implement a prototype of HELPER. We are interested in using this system in future research.

- Are there practical applications where using helper locks leads to simpler and/or more efficient parallel programs?
- How lightweight can an implementation of helper locks be?
- Is HELPER's runtime support for parallel regions useful for supporting other extensions to a Cilk-like language?

References

[1] K. Agrawal, C. E. Leiserson, and J. Sukha. Helper Locks for Fork-Join Parallel Programming. In *PPoPP* 2010.
[2] M. Frigo, C. E. Leiserson, and K. H. Randall. The Implementation of the Cilk-5 Multithreaded Language. In *PLDI* 1998.