Analyzing Irregular Mutual Exclusion in Parallel Programs

Diego Novillo, Ron Unrau, Jonathan Schaeffer

Computing Science Department
University of Alberta

GCC Engineering
Red Hat, Inc.

31 August 2000

Statement of Problem

- Given a statement *s* and a lock variable *L*, does *s* execute under the protection of lock *L*?
  - **always** → *s* is protected by *L*
  - **never** or **sometimes** → *s* is **not** protected by *L*
- Existing analysis techniques are based on structural definition of mutex regions.
- We propose a dataflow based definition that can identify irregularly shaped regions.

Structure-based Mutex Region Recognition

- `lock/unlock` statements act like `begin/end` block delimiters.
- Mutex regions are single-entry, single-exit blocks.
- Dominance information is used to determine extent of mutex region.
- There is one mutex region `M(L)` spanning statements `2-9`.

```
1: lock(L);
2:     s1;
3:     while (expr) {
4:         s2;
5:         s3;
6:         s4;
7:     }
8:     s5;
9: unlock(L);
```

Problems with Structural Definition

- A structural analyzer will not discover the mutex regions in this case.
- The lock is released briefly to execute `s3`.
- There is 1 mutex region with multiple entry and exit points:

$M(L_1, L_2)$ = {2, 3, 4, 5, 8, 9, 10, 11}

```
 1: lock(L1);
 2:     s1;
 3:     while (expr) {
 4:         s2;
 5:         unlock(L);
 6:         s3;
 7:         lock(L2);
 8:         s4;
 9:     }
10:     s5;
11: unlock(L);
```

Dataflow-based Mutex Region Recognition

- Problem is reduced to that of computing reaching definitions for each lock variable *L*
- Synopsis
  1. `lock(L)/unlock(L)` operations contain a definition for *L*.
  2. Every other node in the flowgraph contains a use of *L*.
  3. A node is protected by *L* if and only if all reaching definitions for *L* come from `lock` nodes.
- Every `lock(L)` defines a mutex region with all the nodes reached by its definition of *L*.
- Regions with common nodes are merged.
- Mutex regions may have multiple entry and exit points.

Lock Picking

- Mutex analysis is part of the CSSAME form.
- Allows removal of superfluous conflicts that cannot occur because of synchronization.
- Lock picking examines every lock node in the program.
- If every entry node of a mutex region contains no conflicts for its lock variable, the region locks can be removed.

Lock Picking

```
parloop (p, 1, N) {
  for (i = 0; i < M; i++) {
    lock(R);
    for (j = 0; j < N; j++) {
      sum_reduction(a[i][j]);
    }
    unlock(R);
  }
}

sum_reduction(double x)
{
  lock(S);
  SUM = SUM + x;
  unlock(S);
}
```

GOAL

Remove unnecessary lock operations

Always protected by lock R → nested lock

Conclusions

- We presented a new analysis to identify and validate irregular mutex synchronization patterns.
- Using this analysis together with the CSSAME form, it is possible to detect and remove unnecessary lock operations.