Termination Detection Barriers

Peter Thiemann

January 16, 2011
Outline

1. Termination Detection Barriers
Termination Detection Barriers

- Barriers up to now
  - Computation organized in phases
  - Barrier used to synchronize phase transition
- Different kind of barrier: termination detection
  - Thread pool: terminate when all threads have run out of work
  - More complicated than bare counting
  - Threads must reach consensus that all of them are inactive
Termination Detection Barrier Interface

```java
public interface TDBarrier {
    void setActive (boolean state);
    boolean isTerminated ();
}
```

- `setActive(true)` is called before the thread starts looking for work
- `setActive(false)` is called when the thread is definitively out of work
- `isTerminated()` returns `true` when all threads are unemployed
public class SimpleTDBarrier implements TDBarrier {

    AtomicInteger count;
    int size;

    public SimpleTDBarrier (int n) {
        count = new AtomicInteger (n);
        size = n;
    }

    public void setActive (boolean active) {
        if (active)
            count.getAndDecrement();
        else
            count.getAndIncrement();
    }

    public boolean isTerminated () {
        return count.get() == size;
    }
}

Operation

- Counter initialized to number of participating threads
- Transitions of each thread modifies the counter:
  - inactive → active: decrements counter
  - active → inactive: increments counter
- If all threads are inactive, then the counter reverts to the number of threads: termination!
public void run () {
    int me = ThreadID.get();
tdBarrier.setActive (true);
    Runnable task = queue[me].popBottom();
    while (true) {
        while (task != null) {
            task.run();
            task = queue[me].popBottom();
        }
        tdBarrier.setActive (false);
        while (task == null) {
            int victim = random.nextInt () % queue.length;
            if (!queue[victim].isEmpty()) {
                tdBarrier.setActive (true);
                task = queue[victim].popTop();
                if (task == null)
                    tdBarrier.setActive (false);
            }
            if (tdBarrier.isTerminated())
                return;
        }
    }
}
A subtlety
- Tests whether queue is empty (line 13) before declaring activity.
- Otherwise, threads announce activity even if there is no chance of successfully stealing work.

Proof obligations
- Safety: if `isTerminated()` returns `true`, then the computation has indeed terminated.
- No active task may declare itself inactive. (Other way round ok)
- Liveness: if the computation terminates, then `isTerminated()` eventually returns `true`.
- See above subtlety.