Lecture 16: Cilk Scheduler I

In this and the following lectures we shall describe the Cilk scheduler in detail, and prove that, on P processors, it achieves execution time

$$T_P = T_1/P + O(T_\infty) ,$$

with high probability. In this lecture we give an overview of the proof; the details will be filled in the following lectures.

1 Scheduling Algorithm

- Each processor has a ready deque (double-ended queue). Threads are inserted on bottom, removed from either end.
- When a processor begins work stealing, it operates as follows:
  - It chooses a victim uniformly at random.
  - If the victim's deque is empty, it tries again.
  - Otherwise, it steals the top thread and begins working on it.

A processor works on a thread α until α ...

- spawns β – puts α on bottom of ready deque, and starts work on β
- returns
  - if deque nonempty, begins to work on the bottom thread
  - if deque empty, tries to steal α's parent
  - if α's parent busy, work steal
- sync – if there exists outstanding children, work steal (Note: deque is empty)

Initially, one processor has the root thread; the other processors start work stealing.

The following structural lemma is of fundamental importance. It says that the right threads to steal are those at the top of the ready deque, because they are shallower in the dag.

Lemma 1 *Consider any processor p at any time t. Let α_0 be the thread that p is working on. Let $\alpha_1, \alpha_2, \ldots, \alpha_k$ denote threads in p's deque from bottom to top. Then*

- *α_{i-1} is a child of α_i for $i = 1, \ldots, k$ and is an only child unless its parent is a_k.*
- *P processors $\Rightarrow$ the activation tree has $\leq P$ leaves.*

Proof. (Sketch) The proof can be done by induction on actions. Each action of the scheduler preserves the inductive hypothesis.

1. steal
2. spawn
3. return: steal parent (busy leaves property)
4. sync: only stalls at top level

Corollary 2 *Assume spawn causes child to have dag depth 1 greater than the continuation. Then dag depth increases with depth in deque.*

Thus, the shallowest thread at the top of some deque.
Recall Brent's theorem:

- P processors busy $\Rightarrow \leq T_1/P$ time
- $\leq P$ processors busy $\Rightarrow T_\infty$ time

Idea: When stealing, we want to have good probability of making progress on the critical path. Here we assume that stealing conflicts can be resolved in constant time—we will worry about this assumption later.

2 Accounting Argument

Imagine we have two buckets: *work* and *steal*

On each step, each processor puts \$1 into one of two buckets depending on its action on that step:

- executing instruction – puts \$1 into *work* bucket
- work stealing – puts \$1 into *steal* bucket

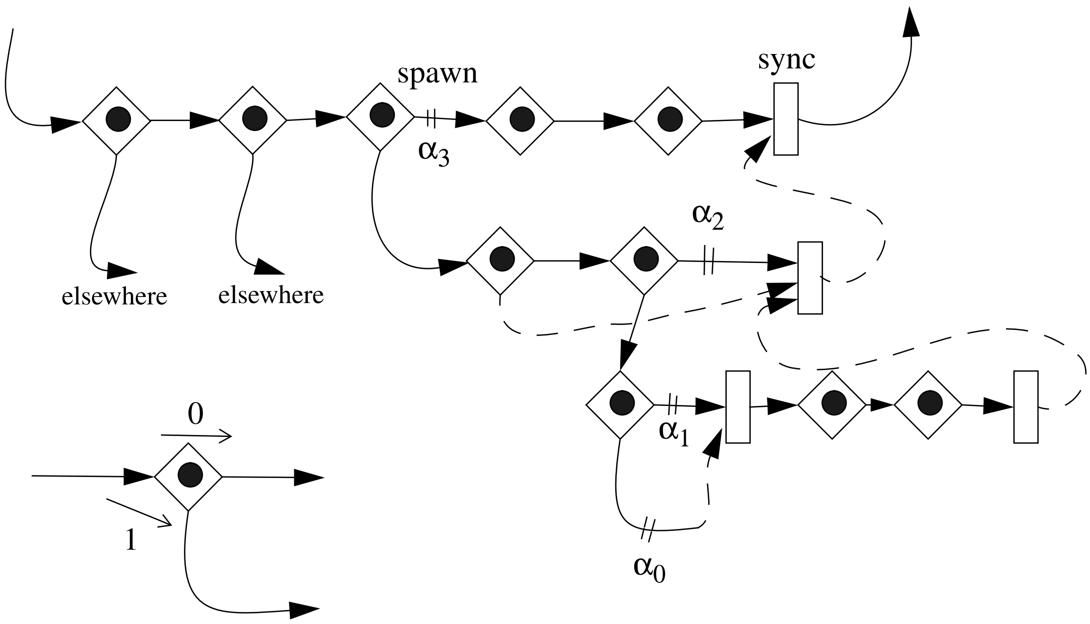
At the end, there are T_1 dollars in the *work* bucket. We will show that, with high probability, there are $O(PT_\infty)$ dollars in *steal* bucket.

$$\text{Total time} = \frac{T_1 + O(PT_\infty)}{P} = \frac{T_1}{P} + O(T_\infty)$$

Proof. Next lecture.

Idea: After $\Theta(P)$ steals, constant expected progress is made on critical path. The probability that the critical processor is not hit:

$$\left(1 - \frac{1}{P}\right)^{\Theta(P)} \leq e^{-\Theta(1)}$$


spawn
sync
α_3
α_2
α_1
α_0
elsewhere
elsewhere
0
1