Lecture 17-18: Analysis of the work-stealing algorithm

In these lectures we analyze the Cilk work-stealing scheduler. Let the dag G describe the multithreaded computation. Nodes represent instructions and edges represent dependencies or spawns (see Figure 1 of Lecture 7). Let T_∞ denote the critical path length of G and let T_1 denote the total work of G.

We use an accounting argument to bound the running time. At all times in the Cilk execution, a processor is either *executing instructions* or *attempting to steal*. For out argument, we postulate two buckets, a WORK bucket and a STEAL bucket. We assume that during each time step

- if the processor is working on a thread, it puts a dollar in the WORK bucket;
- if a processor is attempting to steal, it puts a dollar in the STEAL bucket.

The *total work* executed during the computation is the sum of the dollars in the two buckets. In each time step, P dollars are put in the buckets. Thus, the *execution time* is the total number of dollars divided by P.

Lemma 1 *The total number of dollars in the* WORK *bucket is exactly T_1.*

We now bound the total number of dollars in the STEAL bucket. We create an *augmented dag G'* by adding additional edges to G as follows. For every set of instructions u, v, and w such that (u, v) is a spawn edge and (u, w) is a continue edge, the *deque* edge (w, v) is placed in G'. (See Figure 1.) The augmented dag G' captures the structures of the ready deques

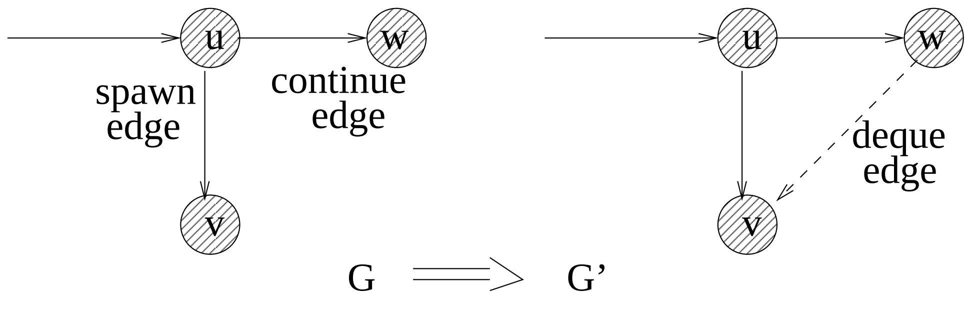

Figure 1: The augmented dag G' constructed from G.

in the following sense. If u has no unexecuted predecessors in G', then it is at the top of a deque.

Note 1 *The depth of G' (i.e., the length of the longest chain) is no more than $2T_\infty$.*

Note 2 *Deque edges are not dependency edges. That is, if (w, v) is a deque edge, v could be executed before w.*

Definition 1 *An instruction is* critical *if every instruction that precedes (directly or indirectly) it in G' has been executed.*

A critical instruction must be ready, since G' contains every edge of G, but a ready instruction may or may not be critical. The structural properties of a ready deque guarantee that if a thread is not on top of a ready deque, then it cannot be critical, because the predecessor of the thread's current instruction across the deque edge has not yet been executed.

We now define the notion of a *round*.

Definition 2 *A* round *of work steal attempts is a set of at least $2P$ but fewer than $3P$ consecutive steal attempts. (We assume that steal attempts always complete in one unit of time.)*

We divide the execution into rounds inductively. Let t_1 be the earliest time such that at least $2P$ steal attempts have occurred. Since at the most P steals can happen in one time step, the number of steals is at most $3P - 1$. The interval $(0, t_1)$ is the first round. The i-th round is the interval (t_{i-1}, t_i), where t_i is the earliest time such that at least $2P$ steal attempts have occurred since t_{i-1}. Figure 2 illustrates this idea.

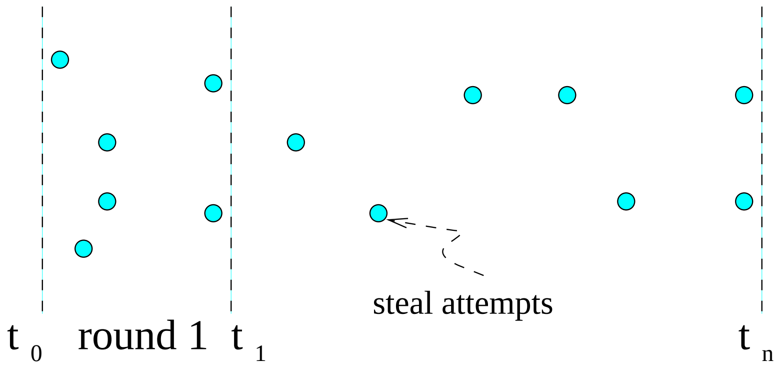

Figure 2: Rounds.

Idea: It is unlikely that an instruction remains critical for many rounds. We use this fact to bound the number of rounds, which in turn bounds the number of steals. More precisely, we use the *delay sequence argument* to bound the number of rounds.

We now formalize our definition of a delay sequence.

Definition 3 *A* delay sequence *is a 3-tuple (U, R, Π) satisfying the following conditions:*

- *$U = (u_1, u_2, \ldots, u_L)$ is a maximal directed path in G'. Specifically, for $i = 1, 2, \ldots, L-1$, the edge (u_i, u_{i+1}) belongs to G', instruction u_1 has no incoming edges in G' (instruction u_1 must be the first instruction of the root thread), and instruction u_L has no outgoing edges in G' (instruction u_L must be the last instruction of the root thread).*

- *R is a positive integer.*
- *$\Pi = (\pi_1, \pi_2, \ldots, \pi_L)$ is a partition of the integer R.*

The delay sequence (U, R, Π) is said to occur *during an execution if for each $i = 1, 2, \ldots, L$, at least π_i steal-attempt rounds occur while instruction u_i is critical.*

Lemma 2 *Consider the execution of a fully strict multithreaded computation with critical path length T_∞ by the Work-Stealing Algorithm on a computer with P processors. If at least $3P(2T_\infty + R)$ steal attempts occur, then some (U, R, Π) delay sequence occurs.*

Proof. For a given execution in which at least $3P(2T_\infty + R)$ steal attempts take place, we construct a delay sequence (U, R, Π) and show that it occurs. With at least $3P(2T_\infty + R)$ steal attempts, there must be at least $2T_\infty + R$ rounds of steal attempts. We construct the delay sequence by identifying a set of instructions on a directed path in G' such that for every time step during the execution, one of these instructions is critical. At most $2T_\infty$ instructions are on the delay sequence, so there are at most $2T_\infty$ steal-attempt rounds, where one of these instructions gets executed. Therefore, there must be at least R steal-attempt rounds that occur while an instruction from the delay sequence is critical. To finish the proof, we produce a directed path $U = (u_1, u_2, \ldots, u_L)$ such that for every time step during the execution, one of the u_i is critical. The partition $\Pi = (\pi_1, \pi_2, \ldots, \pi_L)$ can be derived by simply letting π_i equal the number of steal-attempt rounds that occur while u_i is critical.

We construct a delay sequence backwards. Let u_L be the last instruction of the root thread. Let $u_\ell, u_{\ell+1}, \ldots, u_L$ be a path in G', where u_ℓ is the last instruction executed that precedes u_L. We emphasize that u_ℓ is not necessarily an immediate predecessor of u_L, since execution order follows G, not G'. Thus, just before it is executed, u_ℓ is critical; after it is executed, u_L is critical. We repeat this process to find a path $u_{\ell'}, u_{\ell'+1}, \ldots, u_\ell$, such that $u_{\ell'}$ is the last instruction executed that precedes u_ℓ. We continue until reaching u_1, which is always the first instruction of the root thread. $\square$

Lemma 3 *Consider the execution of any fully-strict multithreaded computation by the Work-Stealing Algorithm on a parallel computer with $P \geq 2$ processors. The probability that an instruction stays critical for r rounds, in the augmented graph G', is less than e^{-2r}.*

Proof. A critical instruction must be topmost in some deque. Since the number of steal attempts in r rounds is at least $2rP$, the probability that none of these attempts steals the instruction is at most

$$\left(1 - \frac{1}{P}\right)^{2Pr} \leq e^{-2r} \ .$$

(This follows from the very very very very important inequality, $\forall x,\ 1 + x \leq e^x$.) $\square$

Lemma 4 *For any $\epsilon > 0$, with probability at least $1-\epsilon$, the* STEAL *bucket contains at most $O(P(T_\infty + \lg(1/\epsilon)))$ dollars.*

Proof. We first compute the probability that a given (U, R, Π) delay sequence occurs.

$$\begin{aligned}
\Pr\{(U,R,\Pi) \text{ occurs}\} &= \prod_{1\le i\le L} \Pr\{\pi_i \text{ rounds occur while } u_i \text{ is critical}\} \\
&\le \prod_{1\le i\le L} e^{-2\pi_i} \\
&= e^{-2R}\,.
\end{aligned}$$

The probability that *any* (U, R, Π) delay sequence occurs is at most

$$(\text{number of } (U,R,\Pi) \text{ delay sequences}) \cdot e^{-2R}\,.$$

We compute an upper bound on the number of the delay sequences as follows. The outdegree of nodes of G' is at the most 2. The longest path in G' contains at most $2T_\infty$ nodes. Therefore, there are at most 2^{2T_∞} different paths in G'.

The delay sequence partitions the path into R pieces according to Π. The number of ways of choosing Π is $\binom{L+R-1}{R} \le \binom{2T_\infty+R}{R}$. (This quantity is the number of ways throwing R indistinguishable balls into L bins.) Therefore the probability that any (U, R, Π) delay sequence occurs is

$$2^{2T_\infty}\binom{2T_\infty+R}{R}e^{-2R} \le 2^{2T_\infty}\left(\frac{2T_\infty+R}{eR}\right)^R,$$

since $\binom{a}{b} \le \left(\frac{ea}{b}\right)^b$. Thus, if $R \ge cT_\infty$,

$$\begin{aligned}
\Pr\{\text{any } (U,R,\Pi) \text{ occurs}\} &\le 4^{T_\infty}\left(\frac{2+c}{ec}\right)^R \\
&\le \left[4^{1/c}\left(\frac{2+c}{ec}\right)\right]^R \\
&\le (0.8)^R \quad \text{for } c = 4 \\
&\le \epsilon \quad \text{for } R \ge 3.11\lg(1/e)\,.
\end{aligned}$$

□

Theorem 5 *With probability at least $1-\epsilon$, the execution time $T_P = T_1/P + O(T_\infty + \lg(1/\epsilon))$.*

Proof. We use the strategy outlined at the beginning of the notes for this lecture. The number of dollars in the WORK bucket is T_1. The number of dollars in the STEAL bucket is $O(P(T_\infty + \lg(1/\epsilon)))$ with probability at least $1-\epsilon$. In each time step, exactly P dollars are put in the buckets. Thus

$$PT_P = T_1 + O(P(T_\infty + \lg(1/\epsilon)))\,.$$

Dividing by P, we obtain the theorem. □

The following corollary was not presented in the lecture. We include it for completeness.

Corollary 6 *The expected execution time on P processors* $\mathrm{E}[T_P]$ *is bounded by* $T_1/P + O(T_\infty)$.

Proof.

$$
\begin{aligned}
\mathrm{E}[T_P - T_1/P - O(T_\infty)] &\le \Pr\{T_P - T_1/P - O(T_\infty) \le 0\} \cdot 0 \\
&\quad + \Pr\{0 < [T_P - T_1/P - O(T_\infty)] \le 1\} \cdot 1 \\
&\quad + \Pr\{1 < [T_P - T_1/P - O(T_\infty)] \le 2\} \cdot 2 \\
&\quad + \ldots \\
&\le \sum_{k=0}^{\infty} \Pr\{T_P - T_1/P - O(T_\infty) > k\} \cdot (k+1)\,.
\end{aligned}
$$

Recall that $\Pr\{T_P - T_1/P - O(T_\infty) > k\} \le \epsilon$, where $k = O(\lg(1/\epsilon))$, i.e., $\epsilon = 2^{-k/O(1)}$. We obtain

$$
\begin{aligned}
\mathrm{E}[T_P] &= T_1/P + O(T_\infty) + \sum_{k=0}^{\infty} \frac{k+1}{2^{k/O(1)}} \\
&= T_1/P + O(T_\infty) + O(1)\,.
\end{aligned}
$$

$\square$