6.895 Theory of Parallel Systems Seth Gilbert

Problems 1–3

Problem 1.

Theorem 1

$$T_P \leq \frac{T_1 - T_\infty}{P} + T_\infty$$

Proof As in the proof of Brent-Graham, we consider the complete and incomplete steps that the scheduler takes. Let T_C be the number of complete steps, and T_I be the number of incomplete steps. Let W_C be the amount of work done during complete steps. We know, then, that:

$$T_P \leq T_C + T_I \,. \tag{1}$$

Notice that, as in the proof of Brent-Graham, there are at most T_∞ incomplete steps; that is:

$$T_I \leq T_\infty \,. \tag{2}$$

This follows from the argument that in each incomplete step, at least 1 unit of work on the critical path is accomplished.

Now we consider the number of complete steps. In Brent-Graham, we noticed that at most T_1 work can be accomplished during complete steps; that is $W_C \leq T_1$. Here, we realize that we can subtract the work accomplished during incomplete steps, and thus get the tighter bound:

$$W_C \leq T_1 - T_I \,. \tag{3}$$

Since there are p processors doing work in each complete step, we get the following bound:

$$T_C \leq \frac{T_1 - T_I}{p} \,. \tag{4}$$

We now start with Inequality (1), and substitute:

$$\begin{aligned}
T_P &\leq T_C + T_I && && (5)\\
&\leq \frac{T_1 - T_I}{p} + T_I && \text{(by Inequality (4))} && (6)\\
&\leq \frac{T_1}{p} + T_I \cdot \left(1 - \frac{1}{p}\right) && && (7)\\
&\leq \frac{T_1}{p} + T_\infty \cdot \left(1 - \frac{1}{p}\right) && \text{(by Inequality (2))} && (8)\\
&\leq \frac{T_1 - T_\infty}{p} + T_\infty && && (9)
\end{aligned}$$

□

Problem 2. Our goal is to determine the fastest that the program can run on 10 processors. That is, we want to determine a lower bound on T_{10}. (Note that we get an easy lower bound of 10 seconds from the fact that the program runs in 10 seconds on 64 processors.)

Theorem 2 $T_{10} \geq 29$

Proof First, we determine an upper bound on T_∞. Recall that:

$$T_P \geq T_\infty\,. \tag{10}$$

We know that for $P = 64$, $T_P = 10$. This implies that:

$$T_\infty \leq 10\,. \tag{11}$$

Next, recall from Problem 1:

$$T_P \leq \frac{T_1 - T_\infty}{P} + T_\infty\,. \tag{12}$$

Therefore:

$$\begin{aligned}
T_4 &\leq \frac{T_1 - T_\infty}{4} + T_\infty && (13)\\
80 &\leq \frac{T_1 - T_\infty}{4} + T_\infty && (14)\\
&\leq \frac{T_1}{4} + \frac{3}{4} \cdot T_\infty && (15)\\
&\leq \frac{T_1}{4} + \frac{3}{4} \cdot 10 && (16)\\
320 &\leq T_1 + 30 && (17)\\
290 &\leq T_1 && (18)
\end{aligned}$$

Finally, recall that:

$$T_P \geq \frac{T_1}{P} \tag{19}$$

Substituting the results of Inequality (18), we see that:

$$T_{10} \geq \frac{290}{10} \geq 29 \tag{20}$$

$\square$

How do we know that this is the best bound possible? Consider the graph in Figure 1 of the equations determined by Inequalities (12), (10), and (19), when applied to the given data for four and 64 processors. (Graph courtesy of Jim Sukkha.) It is clear, then, that this is the best point in the feasible region.

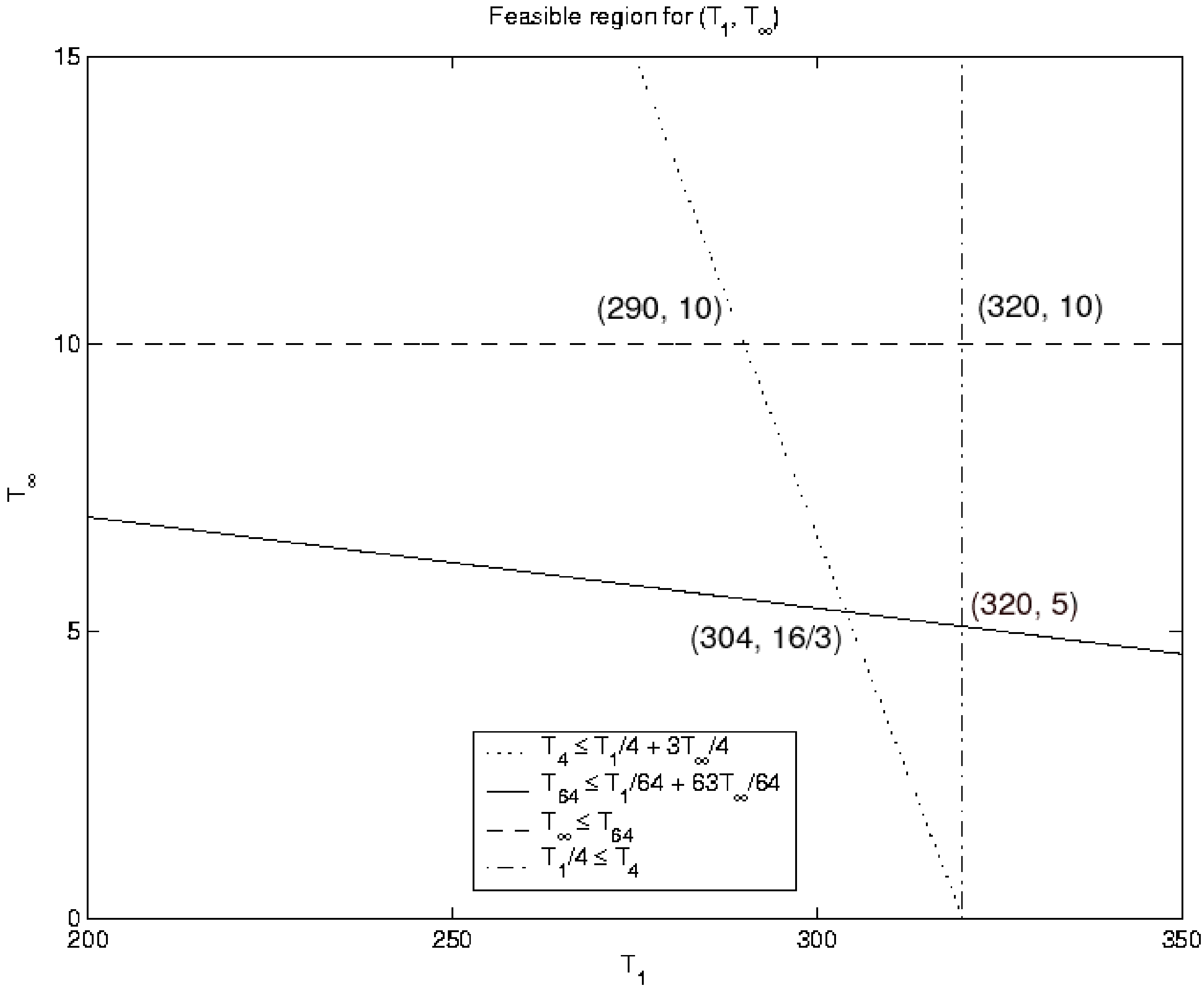

Figure 1: Graph demonstrating bounds on work and critical path.

Problem 3.

Problem 3.a. An arbitrary greedy schedule may continually utilize slow processors rather than fast processors. For example, consider a two processor system where one processor is very fast (π_1) and the other is very slow (π_2). Consider a simple, single-threaded application. The greedy scheduler can schedule the entire computation on either processor 1 or processor 2. The computation is arbitrarily faster if scheduled on the fast processor rather than the slow processor.

Problem 3.b. We need a scheduler with the following property: at all times the fastest processors are working, and the slower processors are idle. We therefore maintain the following invariant: if there are exactly i tasks being executed at some time t, if $i < p$ then the fastest i processors are executing these tasks.

First, whenever a task is ready to begin, the fastest available processor is chosen to begin it. (As in the greedy scheduler, if more tasks are ready to begin than processors are available, all available processors are used.) If a processor completes a task, and no other task is available, it checks if there is a slower processor that is busy. If so, it preempts the slower processor and completes the work of the slower processor.

Problem 3.c.

Theorem 3

$$T_P \le \frac{W_1}{p \cdot \pi_{ave}} + \left(\frac{p-1}{p}\right) \cdot \frac{W_\infty}{\pi_{ave}}$$

Proof In order to analyze the performance of the scheduler, we assume that each processor is doing work at every step, even when it is *idle*. We describe the work done on the program as *real* work, W_R. We describe the work done while the processor is idle as *fake* work, W_F.

This addition of fake work is only for the purposes of analysis; it has no effect on the real execution of the system. In particular, notice that the execution completes as soon as the real work is completed. This book-keeping tool, however, allows us to provide a simple bound on the execution time of the system:

$$\begin{aligned} T_P &\le \frac{W_R + W_F}{\pi_1 + \pi_2 + \cdots + \pi_p} && (21) \\ &\le \frac{W_R + W_F}{p \cdot \pi_{ave}} && (22) \end{aligned}$$

Since every processor is busy at every time step – either doing real or fake work – it is easy to see that by dividing the total work by the total work accomplished each time step, we get a bound on the total time required.

We already know the exact amount of real work accomplished during an execution: $W_R = W_1$. Our goal, then, is to bound the amount of fake work.

Observation 4 *Processor 1 does no fake work.*

Processor 1 is the fastest processor. Our scheduler has the property that at any given time if processor i is idle, then all processors $> i$ are also idle. If processor 1 is idle, that property implies that all processors are idle, and therefore the job is complete.

Consider processor $i > 1$. We attempt to bound the total amount of fake work accomplished by processor i during the entire execution. Notice that whenever processor i is idle (and some processor $< i$ is not idle, as the execution is not complete), there is progress being made on the critical path. This follows from the analogous argument in Brent-Graham: during incomplete steps, progress is being made on the critical path. Moreover, this progress on the critical path must be faster then π_i, since only processors faster then processor i are doing real work when processor i is idle. Therefore:

Observation 5 *Processor i can accomplish at most W_∞ fake work.*

More formally, consider the maximum amount of time that processor i can accomplish fake work:

$$\frac{W_\infty}{\pi_{i-1}}.$$

After this amount of time, any processors faster than i that are doing real work will have completed all the work on the critical path. Therefore, the most fake work that processor i can accomplish is:

$$W_\infty \cdot \frac{\pi_i}{\pi_{i-1}}.$$

Since $\pi_i \leq \pi_{i-1}$, the total amount of fake work accomplished by processor i is bounded by W_∞.

Since $p-1$ processors may be doing fake work, it is clear that the total amount of fake work is bounded:

$$W_F \leq (p-1) \cdot W_\infty\,. \tag{23}$$

Starting with Inequality (22), we see that:

$$\begin{aligned} T_P &\leq \frac{W_R + W_F}{p \cdot \pi_{ave}} && (24)\\ &\leq \frac{W_1 + (p-1) \cdot W_\infty}{p \cdot \pi_{ave}} && (25)\\ &\leq \frac{W_1}{p \cdot \pi_{ave}} + \left(\frac{p-1}{p}\right) \cdot \frac{W_\infty}{\pi_{ave}} && (26)\end{aligned}$$

□

Problem 3.d. The time to complete all tasks can no longer be shown to be within a factor of two of optimal.

First, notice that the proof from Lecture 1 – that the greedy scheduler for equivalent speed processors is within a factor of two of optimal – is no longer valid. In particular, it is no longer true that:

$$T_P \geq \frac{W_\infty}{\pi_{avg}}$$

as would be necessary to follow the proof from class.

This, however, only shows that the bound proved in Part 3.C is not strong enough to provide the desired performance bound. In order to show that the scheduler does not perform within a factor of two of optimal, we provide a counterexample.

Consider the computation graph in Figure 2. Assume there are 101 processors available; processor 1 accomplishes 10 tasks/second, and all other processor accomplish 1 task/second.

Observation 6 *The optimal scheduler completes this computation in no more than 504 seconds.*

First, processor 1 works on all 101 of the size 40 tasks. This takes 404 seconds. Then processors 2 through 101 work on the 100 tasks of size 100. This takes 100 seconds.

Observation 7 *The modified greedy scheduler of 3.B may take 1304 seconds on this computation.*

First processor 1 does the first task of size 40. Next, processor 1 works on the task of size 100, while processor 2 works on the task of size 40. When processor 1 completes the task of size 100 (in 10 seconds), it then finishes the remaining 30 units of the task of size 40 ins 3 seconds. Similarly, each level (containing one task of size 100 and one task of size 40) takes 13 seconds. Therefore, the total time for this schedule is 1304 seconds. Notice that processor 1 is busy the entire time, satisfying our requirement for the greedy scheduler.

Therefore, we conclude that the modified greedy scheduler is not within a factor of two of optimal.

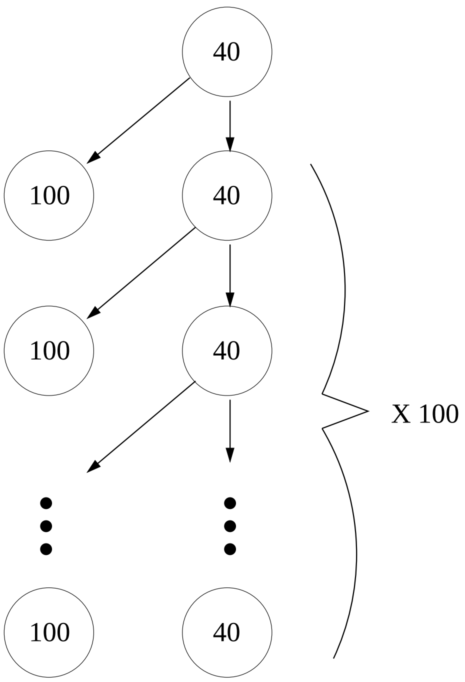

Figure 2: A potentially slow computation. A circle containing a number represents a chain of tasks of the specified size. The arrows represent dependencies in the computation.