CSE 332: Data Structures and Parallelism

Parallelism Solutions

Work it Out [the Span]

Consider the following sequential code which returns the average value in a perfect binary tree:

```
int findAverage (Node n) {
   if(n != null){
      if (n.left == null && n.right == null) {
         return n.value;
      } else {
         return (n.value + (findAverage(n.left) + findAverage(n.right)) *
       (2 ^ (n.height − 1) − 1)) / 2 ^ n.height − 1;
      }
    return 0;
   }
}
```

(a) What's the asymptotic big-$\mathcal{O}$ runtime of this function?

Solution:

We can express the recurrence as:

$$T(n) = \begin{cases} c_0 & \text{if n is null or a leaf} \\ 2T(n/2) + c_1 & \text{otherwise} \end{cases}$$

The recurrence tree modelling this recurrence has height $\log(n)$, each node has some constant amount of work (we don't have to worry about splitting this one up into base case nodes and branch nodes because we're looking for asymptotic runtime, so we'll just use "some constant" and call it c_2 until we can find a big-$\mathcal{O}$), and there are 2^i nodes at the ith level. Putting this together, we get:

$$\begin{aligned} \sum_{i=0}^{\log(n)} c_2 2^i &= c_2 \sum_{i=0}^{\log(n)} 2^i \\ &= c_2 \frac{1 - 2^{\log(n)}}{1-2} && \text{applying one of our closed forms} \\ &= c_2(2^{\log(n)} - 1) \\ &= c_2(n-1) \end{aligned}$$

From here we see the runtime of this function is $\mathcal{O}(n)$.

(b) Now suppose we parallelized this function. If instead of just making recursive calls, we forked a new thread for each call, what would the work and span of this new function be?

Solution:

Work is defined as the amount done if we ran the program with only one thread. In this example, that would be the exact same as the original function, whose runtime we computed to be $\mathcal{O}(n)$. So, the work of this function is $\mathcal{O}(n)$.

Span is defined as the largest amount done by any thread if we ran the program with ∞ threads. If we had that many threads, both recursive calls would happen at the same time, so our new recurrence would look like:

$$T_\infty(n) = \begin{cases} c_0 & \text{if n is null or a leaf node} \\ T_\infty(n/2) + c_1 & \text{otherwise} \end{cases}$$

The new recurrence tree for this recurrence would have $\log(n)$ height, "some constant" amount of work at each node again (call it c_2 because it really doesn't matter here), and one node per level. So to get the closed form of this, we would get:

$$\begin{aligned}\sum_{i=0}^{\log(n)} c_2 &= c_2 \sum_{i=0}^{\log(n)} 1 \\ &= c_2(\log(n) + 1)\end{aligned}$$

We see here that our span is $\mathcal{O}(\log(n))$.

(c) What's the maximum possible speedup of this function when we implement it in parallel?

Solution:

Speedup is defined as the amount of work done when the program is run with only one Processor (i.e. sequentially) divided by the maximum amount of work done by one thread when the program is run with P processors. The maximum possible speedup, then, is the work done with only one processor divided by the work done with infinite processors. So it's work divided by span. This is $n/\log(n)$.