CS260 Homework 1 Solutions

February 6, 2020

1 Solve Recurrences (15 pts)

Applying Master Theorem we can get

(a) $T(n) = \Theta(n \log^2 n)$

(b) $T(n) = \Theta(\log n)$

(c) $T(n) = \Theta(\log \log n)$

(d) $T(n) = \Theta(n^{\log_5 10})$

(e) $T(n) = \Theta(n^2)$

(f) (bonus) The answer is $\Theta(\sqrt{n})$

2 Medians (10pts)

Questions:

(a)

$$W(n) = W(n/5) + W(7n/10) + \Theta(n)$$
$$D(n) = D(n/5) + D(7n/10) + \Theta(\log n)$$

(b) (4pts) Yes. Assume the cost $W(n) \leq cn$ for some constant c. Suppose the constant hidden in Θ is c'. Plug in the recurrence we know:

$$W(n) \leq cn/5 + c \times 7n/10 + c'n \leq (9c/10 + c')n$$

Let $c > 10c'$ proves the cost.

(c) (bonus) The answer is $\Theta(n^{0.84})$.

3 N-ary Forking vs. Binary-forking (10 pts)

(a) N-ary fork-join:

Algorithm 1:

$$W(n) = 2W(n/2) + \Theta(1)$$
$$D(n) = D(n/2) + \Theta(1)$$

Algorithm 2:

$$W(n) = W(n/2) + \Theta(n)$$
$$D(n) = D(n/2) + \Theta(1)$$

Binary fork-join:

Algorithm 1:

$$W(n) = 2W(n/2) + \Theta(1)$$
$$D(n) = D(n/2) + \Theta(1)$$

Algorithm 2:

$$W(n) = W(n/2) + \Theta(n)$$
$$D(n) = D(n/2) + \Theta(\log n)$$

(b) They have the same work bound but not the same recurrence. The first algorithm divide the original algorithm into two problems of half sizes. The second algorithm reduce the algorithm size into a half in $\Theta(n)$ time.

(c) (3pts) They have the same depth bound under n-ary fork-join, but different depth bounds under binary fork-join. The reason is that the second algorithm uses a parallel for loop, which costs $O(1)$ depth in n-ary fork-join but $O(\log n)$ depth in binary fork-join.

4 Fibonacci Numbers (15 pts)

(a) $v_{i+1} = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix} v_i$.

(b) $v_n = \left(\begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix} \right)^{n-1} v_1$.

(c) Let a matrix $A = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}$. Redefine the associative operation in the prefix sum algorithm as matrix multiplication. For a sequence of n matrices:

$$A, A, A, A, ...A$$

we can use the prefix sum algorithm to compute its prefix sum (prefix product) as:

$$A^0, A^1, A^2, A^3, ...A^{n-1}$$

Then using each of them to multiply v_1 gives you the full list of $F_1, F_2, ...F_n$. The cost directly follows the cost of a prefix sum algorithm.

Remark: Many of you only computed the last element F_n, using a similar algorithm as the reduce algorithm. Note that to get F_n we do not need linear work, but only $O(\log n)$ work.

5 Programming (30 pts)

5.1 Answer the Following Questions

1. The tested machine has 12 cores and 24 hyperthreads are available. 32K L1 data cache, 32K L1 instruction cache. 256K L2 cache. 30M L3 cache.
2. (just performance testing)
3. Usually you need to first adjust the threshold coarsely to find the right order of magnitude, e.g., test 1, 10, 100, 1000, 10000, ... Another way is to use a binary search, e.g., first test using 1, $n/2$ and n, and then find the half that shows better performance and continue. Then you can adjust it in a more fine-grained way. For algorithms like reduce or scan, this number can be larger than several thousands, so you need to test a large range of threshold to find a good parameter.
4. As long as you tried to implement it and optimize it, it will be good.