CSE 6220 Homework 1

Karl Hiner, @khiner6

1

Let S be an array containing unordered elements and you are to develop an algorithm to find the location of a given $x \in S$. Measure the run-time in terms of the number of comparisons required. Average case analysis should be done rigorously by computing the average over all possible inputs, taking each input to be equally likely.

Since the elements are unordered, the best we can do is to look through each element $S_i \in S$, comparing each element with x. That is, we simply compare each element $S_i \in S$ with x, in ascending order of i, and return the first index i such that $S_i = x$. Note that we are told our query element x is in S, so we do not need to consider the case in which x is not found.

(a) Compute the best case, worst case, and average case run-times of the serial algorithm.

The best case run-time of the serial algorithm occurs when x is the first element in S, in which case only one comparison is required.

$$T(n,1)_{\text{best}} = 1$$

In the worst case, x is the last element in S, in which case n comparisons are required.

$$T(n,1)_{\text{worst}} = n$$

For the average case, we are finding the expected value of the number of comparisons $E(C)$ across all possible cases.

Let $P(c)$ be the probability of finding the element x in c comparisons. Then we can find the expected value with:

$$\begin{aligned} E(C) &= \sum_{c=1}^{n} c \cdot P(c) && \text{(def. of expected value)} \\ &= \sum_{c=1}^{n} c \cdot \frac{1}{n} = \frac{1}{n} \sum_{c=1}^{n} c && \text{(assuming all inputs are equally likely)} \\ &= \frac{1}{n} \cdot \frac{n(n+1)}{2} = \frac{n+1}{2} && \text{(sum of arithmetic series)} \end{aligned}$$

Thus, assuming each element in S has an equal probability of being x, the average number of comparisons will be

$$T(n,1)_{\text{avg}} = \frac{n+1}{2}.$$

(b) To solve the problem in parallel, S is evenly distributed on p processors. For simplicity, only count the number of comparisons required as a measure of the run-time, and assume p divides n. Compute the best case, worst case, and average case run-times of the parallel algorithm.

Since S is evenly distributed on p processors, and p evenly divides n, each processor will be assigned $\frac{n}{p}$ elements, and each parallel comparison step will perform p element comparisons. The algorithm returns as soon as one processor has found the element.

The best case occurs when only one comparison step is needed, which is the same as the serial case:

$$T(n,p)_{\text{best}} = 1$$

In the worst case, x is not found until the last comparison step:

$$T(n,p)_{\text{worst}} = \frac{n}{p}$$

To find the average case, observe that the situation in terms of comparison steps is very similar to the serial case - the probability of finding x will be the same for each timestep. The difference is only that the maximum number of comparison steps is now $\frac{n}{p}$ instead of n.

Specifically, the probability of finding x during any single comparison step c will be

$$P(c) = \frac{1}{\left(\frac{n}{p}\right)} = \frac{p}{n}.$$

This means we can use the same result from our serial analysis, substituting $\frac{n}{p}$ for n:

$$T(n,p)_{\text{avg}} = \frac{\frac{n}{p}+1}{2} = \frac{n+p}{2p}$$

(c) Compute the best case speedup, worst case speedup, and average case speedup. State your observations.

$$S(p)_{\text{best}} = \frac{T(n,1)_{\text{best}}}{T(n,p)_{\text{best}}} = \frac{1}{1} = 1$$

$$S(p)_{\text{worst}} = \frac{T(n,1)_{\text{worst}}}{T(n,p)_{\text{worst}}} = \frac{n}{\left(\frac{n}{p}\right)} = p$$

$$S(p)_{\text{avg}} = \frac{T(n,1)_{\text{avg}}}{T(n,p)_{\text{avg}}} = \frac{\left(\frac{n+1}{2}\right)}{\left(\frac{n+p}{2p}\right)} = \frac{(n+1)p}{n+p}$$

Thus, in the best case, the parallel algorithm provides no speedup, and in the worst case, the parallel algorithm is p times faster than the serial algorithm.

Finally, in the average case, the expression for the speedup has n and p in both the numerator and denominator. However, observe that in the numerator, p is multplied by

a factor of n, while in the denominator, n is added. Thus, the term in the numerator dominates the speedup asymptotically,

$$\lim_{n\to\infty} \frac{(n+1)p}{n+p} = p,$$

and we also have a speedup of p for the average case, asymptotically with respect to the problem size.

2

The following information is given on a parallel algorithm:

$$T(n,1) = n^2$$
$$T(n,p) = \frac{n^2}{p} + n, \quad p \leq n^2$$

Write the expression for speedup.

$$S(p) = \frac{T(n,1)}{T(n,p)} = \frac{n^2}{\frac{n^2}{p} + n} = \frac{np}{n+p}$$

What happens to the speedup if n is fixed and p is continually increased?

If we treat n as a constant, then we can express this statement as a limit and evaluate it:

$$\lim_{p\to\infty} \frac{np}{n+p} = n \cdot \lim_{p\to\infty} \frac{p}{n+p} = n \cdot 1 = n$$

Thus, if n is fixed and p is continually increased, the speedup grows linearly with n. In other words, if we implement this parallel algorithm using a very large number of processors, the algorithm will run approximately n times faster than its serial version on a problem of size n.

What happens to the speedup if the problem size n is kept proportional to p, the number of processors (i.e., $n = kp$ for some constant k)?

If $n = kp$ for some constant k, we can substitute kp for n in our function for speedup $S(p)$:

$$S(p) = \frac{np}{n+p} = \frac{kp^2}{p(k+1)} = \frac{kp}{k+1} = \frac{k}{k+1}p$$

Thus, if we linearly increase the problem size with the number of processors (or vice-versa), we will see a corresponding linear speedup. However, we are told that the given speed $T(n,p)$ is only valid for $p \leq n^2$. To verify our result holds for $n = kp$, we can substitute for n:

$$p \leq n^2 \implies p \leq k^2p^2$$

This clearly holds for all positive k and p, and so we conclude that if the problem size is kept proportional to the number of processors with constant k, the speed of this parallel algorithm will improve at a rate of $\frac{k}{k+1}$ as $p \to \infty$.

3

We are given two parallel algorithms for solving a problem of size n.

$$T_1(n, n^2) = \sqrt{n}$$
$$T_2(n, n) = n$$

Which algorithm will run faster on a machine with p processors? Do not make any particular assumption on the value of p. Instead, consider all possible values of p.

From the information provided, it is impossible to say which will run faster if we fix p, since we do not know the relationship between the problem size and the number of processors.

To show that we cannot make any comparisons without more information, we can show two potential $T(n, p)$ equations for each T_n that are consistent with the given instantiations of p in terms of n, but with one set giving $T_1 > T_2$ and one set giving $T_2 > T_1$.

3.1 Case 1: $T_1(n, p) > T_2(n, p)$:

Let $T_1(n, p) = \dfrac{n^2\sqrt{n}}{p}$ and $T_2(n, p) = \dfrac{n^2}{p}$. Then,

$$T_1(n, n^2) = \frac{n^2\sqrt{n}}{n^2} = \sqrt{n}, \quad \text{and}$$
$$T_2(n, n) = \frac{n^2}{n} = n.$$

So the performance of both algorithms is consistent with the given instantiations of p. If we keep p constant and compare the performance of the two algorithms, then T_1 grows at a rate of $n^2\sqrt{n}$, while T_2 grows at n^2. So T_1 is slower than T_2, since $T_1 > T_2$ for any $p > 0$ as $n \to \infty$. (In this example, this holds for all $n > 1$, but we are only concerned with arbitrarily large n for the purposes of this proof.)

3.2 Case 2: $T_2(n, p) > T_1(n, p)$:

Let $T_1(n, p) = \dfrac{n\sqrt{n}}{\sqrt{p}}$ and $T_2(n, p) = \dfrac{n^2\sqrt{n}}{p\sqrt{p}}$. Then,

$$T_1(n, n^2) = \frac{n\sqrt{n}}{n} = \sqrt{n}, \quad \text{and}$$
$$T_2(n, n) = \frac{n^2\sqrt{n}}{n\sqrt{n}} = n.$$

Again, the performance of both algorithms is consistent with the given instantiations of p. If we keep p constant and compare the performance of the two algorithms, then T_1 grows at a rate of $n\sqrt{n}$, while T_2 grows at $n^2\sqrt{n}$. So T_2 is slower than T_1, since $T_2 > T_1$ for any $p > 0$ as $n \to \infty$.

We have shown that both $T_1(n, p) > T_2(n, p)$ and $T_2(n, p) > T_1(n, p)$ for $p > 0, n \to \infty$. Thus, we do not have enough information to say which algorithm is faster.

4

The following information is given on a parallel algorithm:

$$\begin{aligned}\text{Serial execution time} &= n^2 \\ \text{Parallel execution time} &= \frac{n^2}{p} + pn \\ \text{Memory required per processor} &= \frac{n}{p}\end{aligned}$$

(a) Find the largest number of processors (as a function of n) that can be used while being optimally efficient.

Optimal efficiency implies $E(p) = \Theta(1)$:

$$\begin{aligned} E(p) = \frac{T(n,1)}{p \cdot T(n,p)} &= \Theta(1) && \text{(definition of efficiency)} \\ \implies \frac{n^2}{p \cdot \left(\frac{n^2}{p} + pn\right)} &= \Theta(1) && \text{(substitute given execution times)} \\ \frac{n}{n + p^2} &= \Theta(1) && \text{(simplify)} \\ \implies p^2 &= O(n) && \text{(satisfy } \Theta(1) \text{ condition above)} \\ p &= O(\sqrt{n}) && \text{(solve for } p\text{)} \end{aligned}$$

Thus, the number of processors must be no more than order of $\sqrt{n}$ for optimal efficiency. That is, $p = c + d\sqrt{n}$, where p is the number of processors, c and d are constants, and n is the problem size.

(b) Assuming that memory available per processor is fixed, can we scale the number of processors as a function of n according to the equation derived above?

Substituting our upper-bound for p in terms of n (found in (4b) above), we can find the required memory per processor, m:

$$m = O\left(\frac{n}{\sqrt{n}}\right) = O(\sqrt{n})$$

Thus, the memory required per processor increases with the problem size at the same rate of $\sqrt{n}$, which is the same rate as our upper bound on processor count for optimal efficiency. Since we have fixed memory per processor, but the required memory per processor is increasing at the same rate as the processor count as we increase the increasing problem size n, we conclude that we *can not* scale the number of processors as a function of n. That is, not while maintaining optimal efficiency as $n \to \infty$.

5

The complexity of the best sequential algorithm for solving a problem is $\Theta(n^2)$. A parallel algorithm designed for solving the same problem has a complexity given by

$$T(n,p) = \Theta\left(\frac{n^2}{p} + \frac{n}{\sqrt{p}}\log p\right) \quad p \leq n^2$$

Find the maximum number of processors as a function of n that can be used such that the algorithm runs with maximum possible efficiency.

Again, optimal efficiency implies $E(p) = \Theta(1)$.

$$\begin{aligned}
E(p) = \frac{T(n,1)}{p \cdot T(n,p)} &= \Theta(1) && \text{(definition of efficiency)} \\
\implies \frac{\Theta(n^2)}{p \cdot \Theta\left(\frac{n^2}{p} + \frac{n}{\sqrt{p}}\log p\right)} &= \Theta(1) && \text{(substitute given execution times)} \\
\frac{n^2}{n^2 + n\sqrt{p}\log p} &= \Theta(1) && \text{(simplify)} \\
\implies \sqrt{p}\log p &= O(n) && \text{(satisfy } \Theta(1) \text{ condition above)}
\end{aligned}$$

We need to find a value for p in terms of n such that this equality holds, keeping in mind that we can neglect terms with small degrees of n, since the equality must only hold asymptotically with respect to n (as $n \to \infty$).

Since $\log p$ is difficult to manipulate algebraically, one approach is to guess at a function f such that if we set $p = O(f(n))$, then $O\left(\sqrt{f(n)}\log\left(f(n)\right)\right) = O(n)$ holds.

Guess: $f(n) = \dfrac{n^2}{\log n^2}$. Then,

$$\begin{aligned}
\sqrt{p}\log p &= O(n) \\
p(\log p)^2 &= O(n^2) && \text{(squaring both sides)} \\
O\left(\frac{n^2}{\log n^2}\log\left(\frac{n^2}{\log n^2}\right)\right) &= O(n^2) && \text{(substitute } p = O(f(n))\text{)} \\
\frac{n^2}{\log n^2}\left(\log n^2 - \log\log n^2\right) &= n^2 && \text{(simplify)} \\
\frac{n^2}{\log n^2}\log n^2 &= n^2 && \text{(removing low-order } \log\log n^2 \text{ term)} \\
n &= n
\end{aligned}$$

Thus, an upper bound for the number of processors in terms of n that we can use with this algorithm while maintaining optimal efficiency is:

$$p = O\left(\frac{n^2}{\log n^2}\right)$$