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.

(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 is when the element x is found at the first position in the array, in which case only one comparison is required. The worst case run-time is when the element x is not found in the array, in which case n comparisons are required. To compute the average case run-time, we can assume that each element in the array has an equal probability of being the element x we are searching for. Therefore, the average case run-time is $\frac{n+1}{2}$ comparisons.

(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.

In the parallel algorithm, S is evenly distributed on p processors. Each processor will perform $\frac{n}{p}$ comparisons. The best case run-time is when the element x is found on the first processor, in which case only $\frac{n}{p}$ comparisons are required. The worst case run-time is when the element x is not found on any of the processors, in which case n comparisons are required. To compute the average case run-time, we can again assume that each element in the array has an equal probability of being the element x we are searching for. Therefore, the average case run-time is $\frac{n+1}{2p}$ comparisons.

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

To compute the best case speedup, we divide the best case run-time of the serial algorithm by the best case run-time of the parallel algorithm, which is $\frac{n}{1}$ divided by $\frac{n}{p} = p$. The worst case speedup is the worst case run-time of the serial algorithm divided by the worst case run-time of the parallel algorithm, which is n divided by $n = 1$. The average case speedup is the average case run-time of the serial algorithm divided by the average case run-time of the parallel algorithm, which is $\frac{n+1}{2}$ divided by $\frac{n+1}{2p} = p$.

Thus, in the best case, the parallel algorithm is p times faster than the serial algorithm. In the worst case, the parallel algorithm is equally as fast as the serial algorithm. In the average case, the parallel algorithm is p times faster than the serial algorithm.

2

The following information is given on a parallel algorithm:

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

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:

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

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 theoretically 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 $p \leq n^2 p$. To verify our result holds for $n = kp$, we can substitute for n:

$$p \leq n^2 p \implies p \leq k^2 p^3$$

This is clearly true 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 equations, but with one set giving $T_1 > T_2$ and one set giving $T_2 > T_1$.

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

Hi

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

Hi

$$T(n, p)$$

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

4

The following information is given on a parallel algorithm:

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

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

(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?

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.

Based on https://gatech.instructure.com/courses/295282/files/folder/Slides?preview=38054495 pages 30 and 33

What p can we use while maintaining $E(p) = \Theta(1)$?

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

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: $f(n) = \dfrac{n^{\frac{3}{2}}}{\log n}$. Then,

$$
\begin{aligned}
\sqrt{p} \log p &= O(n) \\
\sqrt{O\left(\frac{n^{\frac{3}{2}}}{\log n}\right)} \log O\left(\frac{n^{\frac{3}{2}}}{\log n}\right) &= O(n) && \text{(substitute } p = O(f(n))\text{)} \\
O\left(\frac{n^{\frac{3}{4}}}{\sqrt{\log n}}\left(\log n^{\frac{3}{2}} - \log\log n\right)\right) &= && \text{(simplify)} \\
O\left(\frac{n^{\frac{3}{4}}}{\sqrt{\log n}}\left(\log n^{\frac{3}{2}} - \log\log n\right)\right) &= && \text{(simplify)}
\end{aligned}
$$