Goal for Parallel Algorithm Design

1 Algorithm Performance Measurement

1.1 Runtime

Runtime is the best way to measure an algorithm's performance.

1. **Sequential runtime**: $T(n)$, where n is the problem size. The sequential runtime is just the same as normal algorithm runtime, has upper, lower and average bound.
2. **Parallel runtime**: $T(n, p)$, where p is the number of processors.

1.2 Speedup

The speedup of an algorithm is defined as:

$$\text{Speedup} = \frac{\text{Runtime of the best sequential algorithm}}{\text{Runtime of the parallel algorithm}} \tag{1}$$

In math notation:

$$S(p) = \frac{T(n,1)}{T(n,p)} \tag{2}$$

One of the important **Lemma** is that:

$$S(p) \leq p \tag{3}$$

This can be proven by contradiction method. Suppose:

$$S(p) > p \tag{4}$$

Then:

$$\frac{T(n,1)}{T(n,p)} > p \tag{5}$$

$$T(n,1) > p \cdot T(n,p) \tag{6}$$

This contradicts the **work law**:

$$T(n,1) \leq p \cdot T(n,p) \tag{7}$$

The work of a computation executed by p processors is the total number of primitive operations that the processors perform. Ignoring communication overhead from synchronizing the processors, this is equal to the time used to run the computation on a single processor, which is $T(n, 1)$.

1.3 Parallel Efficiency

The parallel efficiency is defined as:

$$\text{Efficiency} = \frac{\text{Work done by the best sequential alg}}{\text{Work done by the parallel alg}} \tag{8}$$

In math notation:

$$E(p) = \frac{T(n,1)}{p \cdot T(n,p)} = \frac{S(p)}{p} \leq 1 \tag{9}$$

2 Goal for Design

What should be the aim when designing a parallel algorithm? The answer is either speedup or efficiency.

2.1 Case Study

Suppose we have a **3-loop naive sequential algorithm**, the runtime is:

$$T(n, 1) = O(n^3) \tag{10}$$

Now we have two parallel algorithms:

1. $T(n, n^3) = O(\log n)$
2. $T(n, n^2) = O(n)$

For the algorithm 1, the speedup will be:

$$S(n^3) = \frac{T(n,1)}{T(n,n^3)} = \frac{O(n^3)}{O(\log n)} = O(\frac{n^3}{\log n}) \tag{11}$$

And the efficiency will be:

$$E(n^3) = \frac{T(n,1)}{n^3 \cdot T(n,n^3)} = \frac{O(n^3)}{n^3 \cdot O(\log n)} = O(\frac{1}{\log n}) \tag{12}$$

For the algorithm 2, the speedup will be:

$$S(n^2) = \frac{T(n,1)}{T(n,n^2)} = \frac{O(n^3)}{O(n)} = O(n^2) \tag{13}$$

And the efficiency will be:

$$E(n^2) = \frac{T(n,1)}{n^2 \cdot T(n,n^2)} = \frac{O(n^3)}{n^2 \cdot O(n)} = O(1) \tag{14}$$

In a summary:

Algorithm	Speedup	Efficiency
Algorithm 1	$O(\frac{n^3}{\log n})$	$O(\frac{1}{\log n})$
Algorithm 2	$O(n^2)$	$O(1)$

Table 1: Algorithm Comparison

Notice here the O notation for speedup and efficiency does not represent the time complexity anymore, it represents and magnitude. Therefore, the larger O is, the more speedup/efficiency the algorithm is.

Recall the Complexity Comparison:

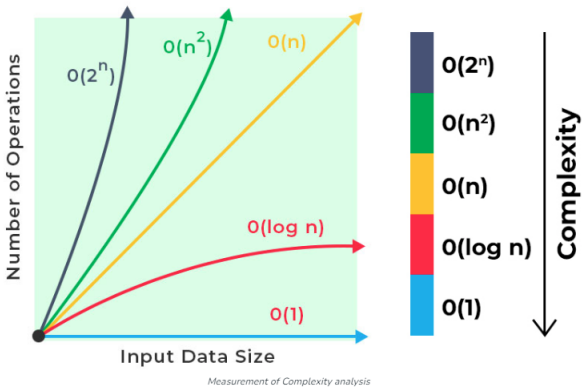

Figure 1: Complexity Comparison

So we know that, if we want to maximize **speedup**, we should choose **algorithm 1**. If we want to maximize **efficiency**, we should choose **algorithm 2**.

2.2 Brent's Lemma

Let $T(n, p_1)$ be the run-time of a parallel algorithm designed to run on p_1 processors. The same algorithm can be run on $p_2 < p_1$ processors **without loss of efficiency** ($E(p_2) \geq E(p_1)$), which is called **efficiency scaling**.

Suppose we run the previous algorithm using p_2 processors, let each processor simulate $\lceil \frac{p_1}{p_2} \rceil$ processors to ensure the number of processors is an integer. Here $\lceil \frac{p_1}{p_2} \rceil$ is the ceiling notation, for example $\lceil 0.5 \rceil = 1$. Therefore:

$$T(n, p_2) = \lceil \frac{p_1}{p_2} \rceil T(n, p_1) \tag{15}$$

$$E(p_2) = \frac{T(n,1)}{p_2 \cdot \lceil \frac{p_1}{p_2} \rceil T(n, p_1)} \tag{16}$$

Based on the definition of ceiling:

$$\lceil \frac{p_1}{p_2} \rceil \leq \frac{p_1}{p_2} + 1 \tag{17}$$

Therefore:

$$\text{RHS} \geq \frac{T(n,1)}{p^2 \cdot (\frac{p_1}{p_2} + 1) T(n,p_1)} = \frac{T(n,1)}{(p_1 + p_2) T(n,p_1)} \tag{18}$$

Recall that:

$$E(p_1) = \frac{T(n,1)}{p_1 \cdot T(n,p_1)} \tag{19}$$

So we can prove that:

$$E(p_2) \geq E(p_1) \tag{20}$$

2.3 Goals of PAD

There are two main goals of PAD, including:

1. **Speed Max:** Design algorithm to **minimize** $T(n,p)$ using the **smallest** possible value p. In other words, to reach a certain speed, we need at least p number of processors.
2. **Efficiency Max:** Design algorithm to **maximize** $E(p)$ where p is the **largest** possible number of processors. In other words, to reach a certain efficiency, we could not have more than p number of processors.

Assume two algorithms:

1. A_1: designed for speed and uses p_1 processors
2. A_2: designed for efficiency and uses a maximum of p_2 processors

The conclusion is that:

$$p_1 > p_2 \tag{21}$$

This could be proven using contradiction method. Suppose $p_1 < p_2$, as shown below:

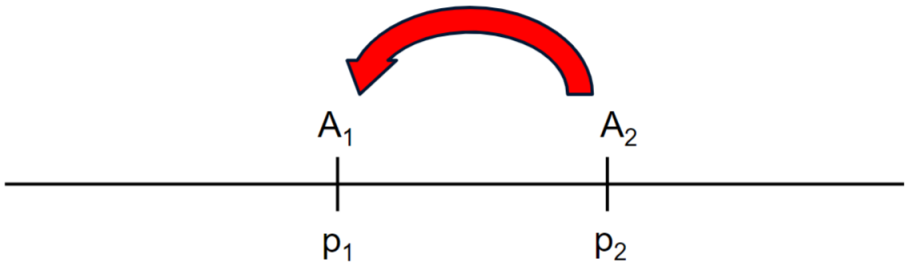

Figure 2: Contradiction Hypothesis

Notice that using Brent's Lemma, now A_1 could be as efficient as A_2 because now $p_1 < p_2$. With the same efficiency, also because $p_1 < p_2$, A_2 will run faster than A_1. This **contradicts** A_1 is designed for speed and A_2 is designed for efficiency. Therefore the conclusion is proven.

In summary, A_1 will be less efficient than A_2, but will be faster than A_2.

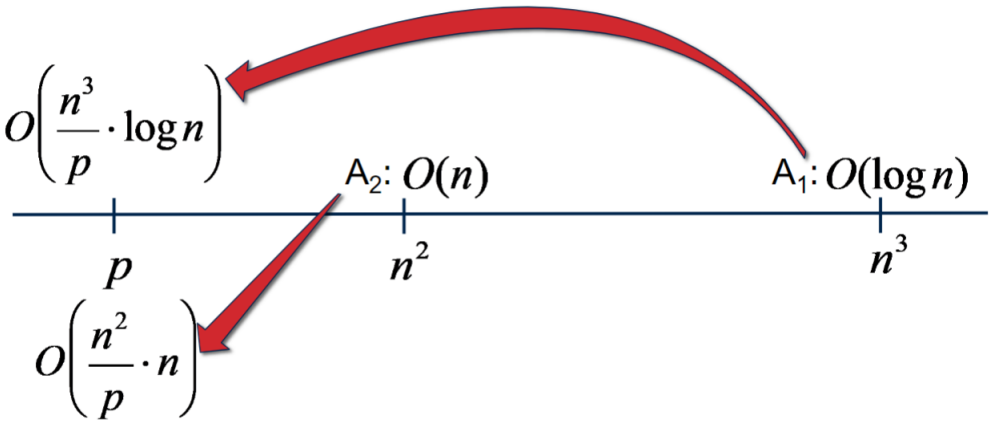

Figure 3: Less processors case

If there are less processors, like $p < p_2$, because A_2 is more efficient than A_1, based on Brent's lemma, p scaled from p_2 will be more efficient than p scaled from p_1, so at the fixed processors A_2 will be **faster** for less processors.

3 Scalability

3.1 Fixed-time scalability

One of the biggest concern is that we want to increase the problem size but retain the same runtime. Still use the matrix multiplication, the sequential runtime is:

$$T(n, 1) = O(n^3) \tag{22}$$

3.1.1 Efficient algorithm

And efficient algorithm is defined as an algorithm if we increase the problem size and corresponding scale of processors, the runtime will be the same. Assume we have the algorithm:

$$T(n, n^3) = O(1) \tag{23}$$

The scaling of this algorithm could be expressed as:

$$T(n, p) = O(\frac{n^3}{p}) \tag{24}$$

If we increase the problem size by 2, the work will increase by 8 (n^3), so if we use 8 processors:

$$T(2n, 8p) = O(\frac{8n^3}{8p}) = O(\frac{n^3}{p}) \tag{25}$$

3.1.2 Inefficient Algorithm

Not all the algorithm are that efficient. Assume we have an algorithm:

$$T(n, n^3) = O(\log n) \tag{26}$$

And the scaling will be:

$$T(n, p) = O(\frac{n^3 \log n}{p}) \tag{27}$$

Similarly, if we increase the size by 2:

$$T(2n, 8p) = O(\frac{8n^3 \log 2n}{8p}) = O(\frac{n^3 \log 2n}{p}) \tag{28}$$

Therefore, the runtime increases, so this algorithm is not efficient algorithm.

3.1.3 Selection from Efficient Algorithms

Assume we have two algorithms:

1. A_2: $T(n, n^2) = O(n), p \leq n^2$
2. A_3: $T(n, n) = O(n^2), p \leq n^2$

We want to choose the better algorithm between this. The scaling of A_2 will be:

$$T(n, p) = O(\frac{n^2}{p} \cdot n) \tag{29}$$

And the scaling of A_3 will be:

$$T(n, p) = O(\frac{n}{p} \cdot n^2) \tag{30}$$

For A_2, if we double the problem size n, we can **quadruple** the number of processors, or **double** the runtime.

For A_3, if we double the problem size n, we can **double** the number of processors, or **quadruple** the runtime.

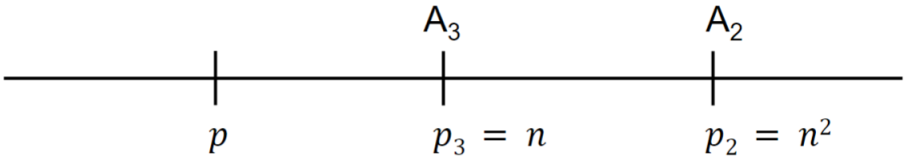

Figure 4: Comparison of Efficient Algorithms

Here p_3 and p_2 will be the limit for each algorithm. Assume the initial conditions:

$$n = 1000, p = 100, p_3 = 1000, p_2 = 1000000 \tag{31}$$

If we double the problem size, and use 8 times processors, then we have:

$$n = 2000, p = 800, p_3 = 2000, p_2 = 4000000 \tag{32}$$

Here it still satisfies:

$$p < p_3 < p_2 \tag{33}$$

Therefore both algorithms work. However, if we double again, we have:

$$n = 4000, p = 6400, p_3 = 4000, p_2 = 16000000 \tag{34}$$

Now the relations are:

$$p_3 < p < p_2 \tag{35}$$

Therefore we could only choose A_2 now.

3.2 Find the required number of processors

The other concern is how to calculate the required number of processors. Here are two examples:

3.2.1 Example 1

Assume we have a parallel algorithm:

$$T(n, p) = \Theta(\frac{n}{p} + \log p) \tag{36}$$

Now we want to **maximize the speed**, which means we want to find p to **minimize** $T(n, p)$, so we take the derivative:

$$\frac{d}{dp}(\frac{n}{p} + \log p) = 0 \tag{37}$$

$$-\frac{n}{p^2} + \frac{1}{p} = 0 \tag{38}$$

$$p = n \tag{39}$$

If we require the efficiency to be $\Theta(1)$, then:

$$E(p) = \frac{T(n, 1)}{pT(n, p)} = \frac{T(n, 1)}{p(\frac{n}{p} + \log p)} = \Theta(1) \tag{40}$$

$$\frac{n}{n + p \log p} = \Theta(1) \tag{41}$$

$$p \log p = O(n) \tag{42}$$

Take a guess:

$$p = O(\frac{n}{\log n}) \approx c\frac{n}{\log n} \tag{43}$$

Could be proven:

$$O(\frac{n}{\log n} \cdot \log \frac{n}{\log n}) = O(\frac{n}{\log n} \cdot (\log n - \log \log n)) \tag{44}$$

The magnitude of $\log \log n$ comparing with $\log n$ is pretty small and could be ignored. Therefore:

$$O(\frac{n}{\log n} \cdot \log n) = O(n) \tag{45}$$

3.2.2 Example 2

Now we assume the algorithm as:

$$T(n, 1) = \Theta(n^2) \tag{46}$$

and the scaling as:

$$T(n, p) = \Theta(\frac{n^2}{p} + \sqrt{n}), p \leq n^2 \tag{47}$$

For speed, we choose $p = n^2$ this time. Now if we want to maintain $\Theta(1)$ efficiency:

$$E(p) = \frac{\Theta(n^2)}{p \cdot (\frac{n^2}{p} + \sqrt{n})} = \Theta(1) \tag{48}$$

$$\frac{n^2}{n^2 + p\sqrt{n}} = \Theta(1) \tag{49}$$

$$p\sqrt{n} = O(n^2) \tag{50}$$

$$p = O(n\sqrt{n}) \tag{51}$$