Model for Parallel Computation

1 Finding the Sum

1.1 Serial Algorithm

Given n numbers, computer their sum. If we are using serial algorithm, we need to add numbers one by one, so the runtime will be:

$$T(n, 1) = O(n) \approx c_1 \cdot n \tag{1}$$

1.2 Parallel Algorithm

Same question, recall the divide and conquer algorithm, which usually needs k divisions:

$$k = \log_2 n \tag{2}$$

So the runtime will be:

$$T(n, n) = O(\log n) \approx c_2 \cdot \log n \tag{3}$$

1.3 Speedup

The speedup could be expressed as:

$$S(n) = \frac{c_1}{c_2} \cdot \frac{n}{\log n} \tag{4}$$

By practice, we have $c_1 \leq c_2$, so:

$$S(n) \leq \frac{n}{\log n} \tag{5}$$

2 Interconnection Network

2.1 Bus

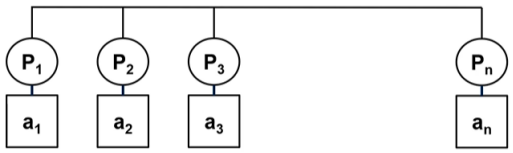

Figure 1: Bus Interconnection

A bus is a shared communication link, a set of parallel wires carrying information from one part of a computer to another. The characteristics include:

1. Bus network has a single communication line, which is called a bus, that all the nodes are connected to.
2. Each node is directly connected to this bus and can communicate with any other node by sending a message over the bus.
3. The bus itself is a shared communication medium, and typically only one node can successfully transmit data at a time, necessitating some form of arbitration to control access to the bus.
4. It's a cost-effective networking approach for small networks but can become a bottleneck as network size and traffic increase.

The fastest way to calculate the sum for a single bus is shown below (notice that all the process need to be done **one by one**):

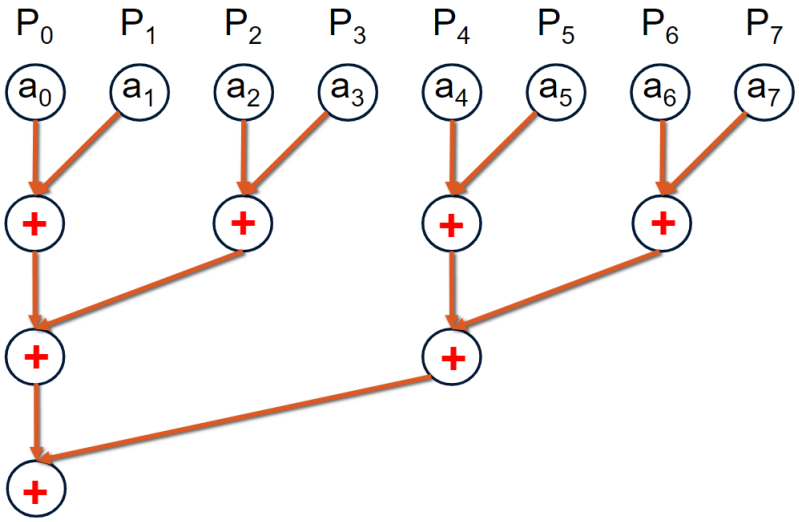

Figure 2: Bus Calculation

Therefore, if n is some 2^k (if not, it will be the approximation), then we have:

$$T(n,n) = c_2 \cdot [\frac{n}{2} + \frac{n}{4} + ... + 1] = c_2 \cdot (n-1) \tag{6}$$

Then the speedup will be:

$$S(n) \approx \frac{c_1 \cdot n}{c_2 \cdot n} = \frac{c_1}{c_2} < 1 \tag{7}$$

In the worst case, we have $c_2 = \tau$, where τ is the **latency**. Latency refers to the time delay between the initiation and the execution of an operation. It usually includes:

1. Communication Latency: This is the time it takes for a message to travel from its source to its destination. In a parallel computer, this could be the time it takes for a data packet to travel across the network from one processor to another.
2. **Memory Latency**: This is the delay between a request for a byte or word in memory and the delivery of the requested byte or word. For instance, when a processor requests data from the main memory, there is a delay before the data is available to the processor due to the time required to access the memory cells.
3. **Processing Latency**: The delay from when an instruction is issued by a processor to when it is completed. This includes the time taken for instruction decoding, execution, and writing back results.

2.2 All Connected Network

The characteristics of a fully connected network:

1. Each node is directly connected to every other node.
2. This type of network requires a large number of connections.
3. Fully connected networks are incredibly robust (there are multiple paths for communication between any two nodes), but they are highly impractical for large systems due to the massive number of connections required.

For the all connected network, we can use the divide and conquer algorithm. Similarly, we have:

$$T(n,1) \approx n \tag{8}$$

$$T(n,n) \approx \tau \cdot \log n \tag{9}$$

If we use $p << n$ processors instead, then we assign $\frac{n}{p}$ numbers per processors. Then the parallel runtime is calculated as:

$$T(n,p) \approx \frac{n}{p} + \tau \cdot \log p \tag{10}$$

Then the speed up will be:

$$S(p) \approx \frac{n}{\frac{n}{p} + \tau \cdot \log p} \tag{11}$$

If the condition satisfies:

$$\tau p \log p \leq n \tag{12}$$

Then we have:

$$S(p) \geq \frac{p}{2} \tag{13}$$

3 Model Communication

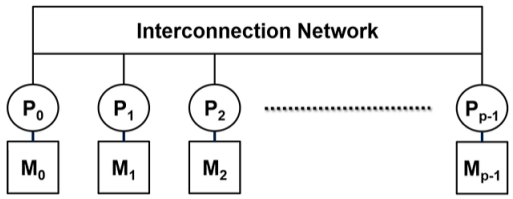

Figure 3: Model Example

Suppose the processor P_i is sending message to P_j of size m, then the transfer time could be calculates as:

$$t_c = \tau + \mu \cdot m \tag{14}$$

Here:

1. τ: Latency, time taken to initiate the transfer, not depending on the size of the message.
2. μ: Time taken to send each unit of data.
3. $\frac{1}{\mu}$: Bandwidth, or the number of data units that be sent per unit of time.

In a parallel communication step, two rules are obeyed:

1. Each processor can send at most 1 message
2. Each processor can receive at most 1 message

4 Permutation Network Model

A permutation network is a specific type of network that is designed to connect a set of inputs to a set of outputs in a pattern that can be rearranged (permuted) so that each input is connected to any of the outputs. This is particularly useful in parallel computing environments where different computational elements need to rapidly change the way they are interconnected, depending on the task at hand.

4.1 Shift Permutation

Shift permutation refers to a specific type of data routing where the elements in a sequence are shifted by a fixed offset. For example, if we have a sequence of four elements:

$$S = [A, B, C, D] \tag{15}$$

Then the 1-position shift permutation to the right would result in:

$$S' = [D, A, B, C] \tag{16}$$

The 2-position shift permutation to the right would result in:

$$S' = [C, D, A, B] \tag{17}$$

Then suppose the total number of processors as p and we want to move i position, then for the left shift:

$$i \to (i - 1 + p) \bmod p \tag{18}$$

And the right shift will be:

$$i \to (i + 1) \bmod p \tag{19}$$

4.2 Hypercubic Permutations

Hypercubic permutations refer to the patterns of data routing and communication that are used in parallel computing architectures based on a hypercube topology. A hypercube, also known as an n-dimensional cube or n-cube, is a generalized concept of a three-dimensional cube to an arbitrary number of dimensions (n). The characteristics of hypercube topology include:

1. Each node in the network is represented as a point in an n-dimensional space.
2. Each node is uniquely identified by an n-bit binary string.
3. A node is directly connected to other nodes whose binary labels differ by exactly one bit. This means that in an n-dimensional hypercube, each node has n neighbors.

Suppose we have:

$$p = 2^d \tag{20}$$

Where d is an integer, and we want the hypercubic permutation on fixed position j. For example we have $p = 8$ and $d = 3$, then:

j = 0	j = 1	j = 2
0=000 ↔ 001=1	0 ↔ 2	0 ↔ 4
2=010 ↔ 011=3	1 ↔ 3	1 ↔ 5
4=100 ↔ 101=5	4 ↔ 6	2 ↔ 6
6=110 ↔ 111=7	5 ↔ 7	3 ↔ 7

Figure 4: Hypercubic permutation

In this case, $j = 0$ refers to change 1 bit in the 0 position, which is the last bit of the string. Similar with $j = 1$ and $j = 2$ cases.

4.3 Hypercubic Permutations Example

The example below is using hypercubic permutations to find the sum of n numbers.

```
Algorithm (for P_i)
sum ← add local n/p numbers
for j=0 to d-1 do

     if ((rank AND 2^j) ≠ 0)
         { send sum to (rank XOR 2^j); exit; }
     else
          receive sum' from (rank XOR 2^j)
          sum = sum + sum'
endfor
if (rank=0)
     print sum
```

Figure 5: Algorithm

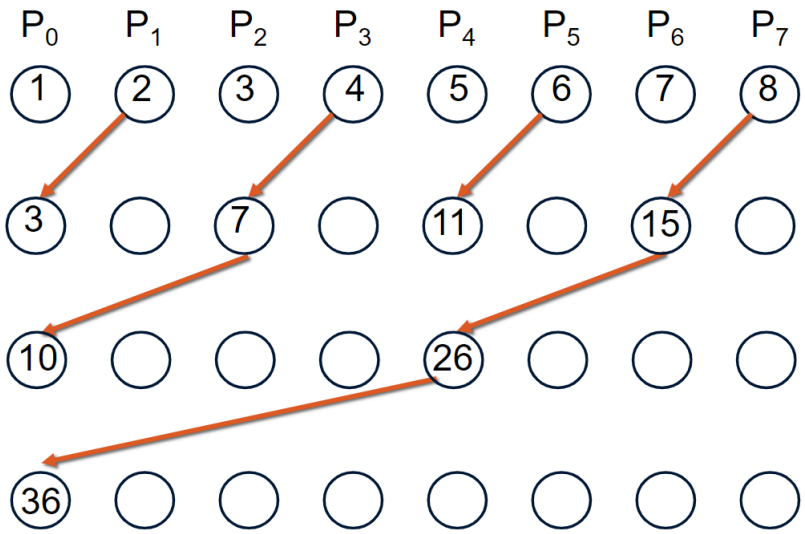

Figure 6: Result

For the AND operation, here is a quick recap:

A (Input)	B (Input)	A AND B (Output)
0	0	0
0	1	0
1	0	0
1	1	1

Figure 7: AND Operation

In the algorithm, AND operation is used determine whether the j^{th} position in the processor bit string has the bit as 1.

For the XOR operation, here is a quick recap:

A (Input)	B (Input)	A XOR B (Output)
0	0	0
0	1	1
1	0	1
1	1	0

Figure 8: XOR Operation

In the algorithm, XOR is used for the value transfer between the processors. For example P_1 to P_0, P_3 to P_2, P_4 to P_0 and so on.