

## Chapter 4 Performance and Scalability of Parallel Systems

**Definition:** A parallel system consists of an algorithm and the parallel architecture that the algorithm is implemented.

Note that an algorithm may have different performance on different parallel architecture.

**4.1.2 Speedup:** The **speedup** is defined as the ratio of the serial run time of the best sequential algorithm for solving a problem to the time taken by the parallel algorithm to solve the same problem on  $p$  processors.

$$S = \frac{T_S}{T_P}$$

**Example 4.1** Adding  $n$  numbers on an  $n$ -processor hypercube

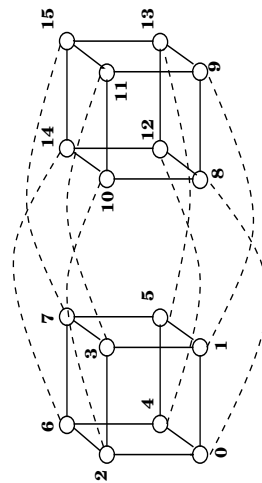
$$T_S = \Theta(n), \quad T_P = \Theta(\log n), \quad S = \Theta\left(\frac{n}{\log n}\right)$$

## 4.1 Performance Metrics for Parallel Systems

**4.1.1 Run Time:** The **parallel run time** is defined as the time that elapses from the moment that a parallel computation starts to the moment that the last processor finishes execution.

**Notation:** Serial run time  $T_S$ , parallel run time  $T_P$ .

**Example 4.1.** Computing the sum of 16 numbers on a 16-processor hypercube



## 4.2 Granularity and Performance

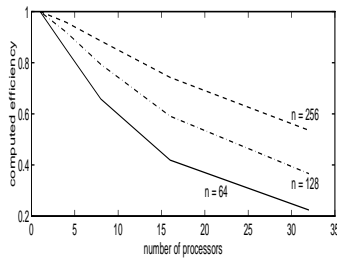
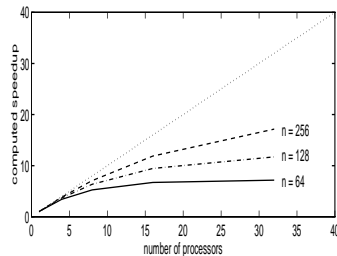
Use less than the maximum number of processors

Increase performance by increasing granularity of computation in each processor

**E.g. 4.5** Adding  $n$  numbers cost-optimally on a hypercube

Use  $p$  processors, each holds  $n/p$  numbers. First add the  $n/p$  numbers locally. Then the situation becomes adding  $p$  numbers on a  $p$  processor hypercube. Parallel run time and cost:

$$\Theta(n/p + \log p), \quad \Theta(n + p \log p)$$



### 4.2.1 Scalability

Scalability is a measure of a parallel system's capacity to increase speedup in proportion to the processor number.

**E.g. 4.6** Adding  $n$  numbers cost-optimally

$$T_p = \frac{n}{p} + 2 \log p$$

$$S = \frac{np}{n + 2p \log p}$$

$$E = \frac{S}{p} = \frac{n}{n + 2p \log p}$$

Well-known **Amdahl's law** dictates the achievable speedup and efficiency.

### 4.2.1 Scalability (cont.)

Increase processors → decrease efficiency

Increase problem size → increase efficiency

Can a parallel system keep efficiency by increasing processor number and problem size simultaneously???

Yes: → **scalable** parallel system

No: → **non-scalable** parallel system

A scalable parallel system can always be made cost-optimal by adjusting the processor number and problem size.

#### 4.2.1 Scalability (cont.)

**E.g. 4.7** Adding  $n$  numbers on a  $n$  processor hypercube, the efficiency is

$$E = \Theta\left(\frac{1}{\log n}\right)$$

No way to fix  $E$  (make the system scalable)

Adding  $n$  numbers on a  $p$  processor hypercube optimally, the efficiency is

$$E = \frac{n}{n + 2p \log p}$$

Choosing  $n = \Omega(p \log p)$  makes  $E$  a constant

#### 4.4.1 Isoefficiency Function

$$T_P = \frac{W + T_O(W, p)}{p}$$

$$E = \frac{1}{1 + T_O(W, p)/W}$$

Solve the above equation for  $W$

$$W = \frac{E}{1 - E} T_O(W, p) = K T_O(W, p)$$

The isoefficiency function determines the growth rate of  $W$  required to keep the efficiency fixed as  $p$  increases

Highly scalable systems have small isoefficiency function.

#### 4.4 Isoefficiency Metric of Scalability

**Degree of scalability:** The rate to increase problem size to maintain efficiency as processor number changes

**Problem size:** Number of basic computation steps in best sequential algorithm to solve the problem on a single processor ( $W = T_S$ )

**Overhead function:** Part of the parallel system cost (processor-time product) that is not incurred by the fastest known serial algorithm on a serial computer

$$T_O = pT_P - W$$

#### 4.5 Sources of Parallel Overhead

Interprocessor communication: increase data locality to minimize communication

Load imbalance: distribution of work (load) is not uniform. Inherent parallelism of the algorithm is not sufficient

Extra computation: modify best sequential algorithm may result in extra computation. Extra computation may be done to avoid communication

#### 4.6 Minimum Execution Time

$$T_P = \frac{n}{p} + 2 \log p$$

**4.1.3 Efficiency:** The **efficiency** is defined as the ratio of speedup to the number of processors. Efficiency measures the fraction of time for which a processor is usefully utilized.

$$E = \frac{S}{p} = \frac{T_S}{pT_P}$$

**Example 4.2** Efficiency of adding  $n$  numbers on an  $n$ -processor hypercube

$$E = \Theta\left(\frac{n}{\log n} \cdot \frac{1}{n}\right) = \Theta\left(\frac{1}{\log n}\right)$$

**4.1.4 Cost:** The **cost** of solving a problem on a parallel system is defined as the product of run time and the number of processors.

A **cost-optimal** parallel system solves a problem with a cost proportional to the execution time of the fastest known sequential algorithm on a single processor.

**Example 4.3** Adding  $n$  numbers on an  $n$ -processor hypercube

Cost is  $\Theta(n \log n)$  for the parallel system and  $\Theta(n)$  for sequential algorithm. The system is **not** cost-optimal.

**A Few Examples:**

E.g. 4.8 Overhead Function for Adding  $n$  Numbers on a  $p$  processor Hypercube

Parallel run time is:  $T_P = \frac{n}{p} + 2 \log p$

Parallel system cost is:  $pT_P = n + 2p \log p$

Serial cost (and problem size) is:  $T_S = n$

The overhead function is:

$$\begin{aligned} T_O &= pT_P - W \\ &= (n + 2p \log p) - n \\ &= 2p \log p \end{aligned}$$

What can we know here?

E.g. 4.10 Isoefficiency Function with a Complex Overhead Function

Suppose overhead function is:

$$T_O = p^{3/2} + p^{3/4}W^{3/4}$$

For the first term:

$$W = Kp^{3/2}$$

For the second term:

$$W = Kp^{3/4}W^{3/4}$$

Solving it yields:

$$W = K^4p^3$$

The second term dominates, so the overall asymptotic isoefficiency function is  $\Theta(p^3)$

E.g. 4.9 Isoefficiency Function for Adding  $n$  Numbers on a  $p$  processor Hypercube

The overhead function is:

$$T_O = 2p \log p$$

Hence, isoefficiency function is

$$W = 2K p \log p$$

Increasing processor number from  $p_0$  to  $p_1$ , the problem size has to be increased by a factor of

$$p_1 \log p_1 / (p_0 \log p_0)$$

If  $p = 4$ ,  $n = 64$ ,  $E = 0.8$ . If  $p = 16$ , for  $E = 0.8$ , we have to have  $n = 512$ .

E.g. 4.14 Minimum Cost-Optimal Execution Time for Adding  $n$  Numbers

Minimum execution time is:

$$p = \frac{n}{2}, \quad T_P^{\min} = 2 \log n$$

If, for cost-optimal

$$W = n = f(p) = p \log p \quad (1)$$

then

$$\log n = \log p - \log \log p \approx \log p$$

Now solve (1) for  $p$ , we have

$$p = f^{-1}(n) = n / \log p \approx n / \log n$$

$$T_P^{\text{Cost-Optimal}} = 3 \log n - 2 \log \log n$$