Performance and Scalability of Parallel Systems

Parallel Systems

- Sequential Algorithm: Execution time expressed as a function of the size of its input.
- Parallel Algorithms: Depends on :
  - input size
  - architecture of the parallel computer on which it is implemented
  - Number of processors
- Parallel System: Parallel Algorithm + Parallel Architecture

Objective

- Study various metrics for evaluating performance of parallel systems
- **Scalability** of a parallel algorithm on an architecture is a measure of its ability to achieve performance proportional to the number of processors.

Performance metrics for parallel Systems

- Run Time
- Speedup
- Efficiency
- Cost

Run Time

- Serial Runtime: time elapsed between beginning and end of its execution on a sequential computer (T_s)
- Parallel Runtime: the moment that a parallel computation starts to the moment that the last processor finishes execution (T_P)

Speedup

- In evaluating a parallel system, interested in knowing how much performance gain is achieved by parallelizing a given application over a sequential implementation.
- Speedup (S): A measure that captures the relative benefit of solving a problem in parallel.

Speedup

- Serial Algorithm:
  - More than one to sequential algorithm
  - Not all suitable for parallelization
  - Serial computer:use the algorithm that solves the problem in the least amount of time.
  - Given parallel algorithm, fair to judge its performance w.r.t the fastest sequential algorithm on a single proc.
  - Best sequential algorithm = fastest sequential algorithm
- Definition: Ratio of the serial time taken of the best sequential algorithm to solve a problem on a single processor to the time required to solve the same problem on a parallel computer with p processors.

Speedup

- Absolute Speedup : $S = \frac{T_s}{T_P}$
- Execution time of best sequential algorithm/Execution time on p processors
- Relative Speedup: $S = \frac{T_1}{T_P}$
- Execution time on 1 processor/Execution on p processors

Theoretical analysis

- Speedup = number of computational steps on 1 processor/number of computational steps with p processors.

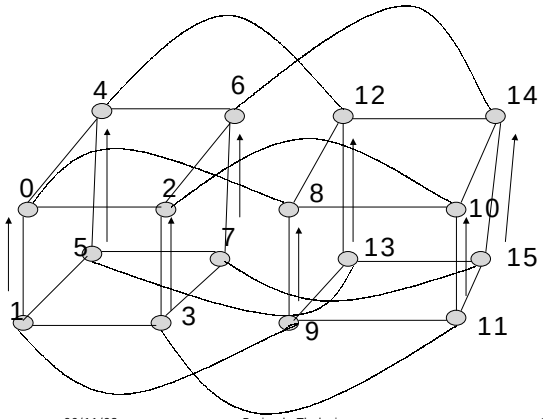
Example

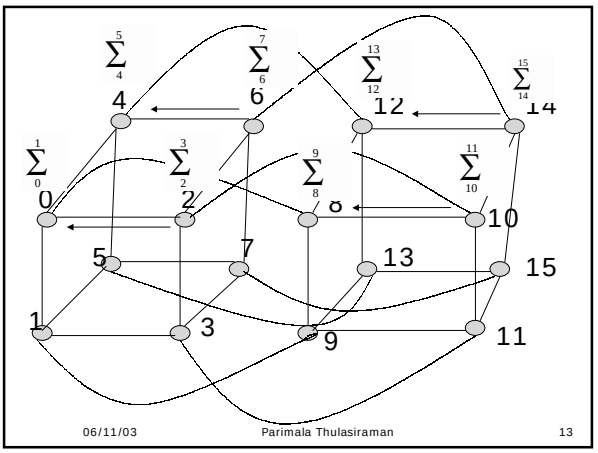
- Adding *n* numbers on an *n* processor hypercube
  - Assume *n* numbers, *n* processors
  - At the end of the computation one of the processor holds the sum
  - *n* is a power of 2.

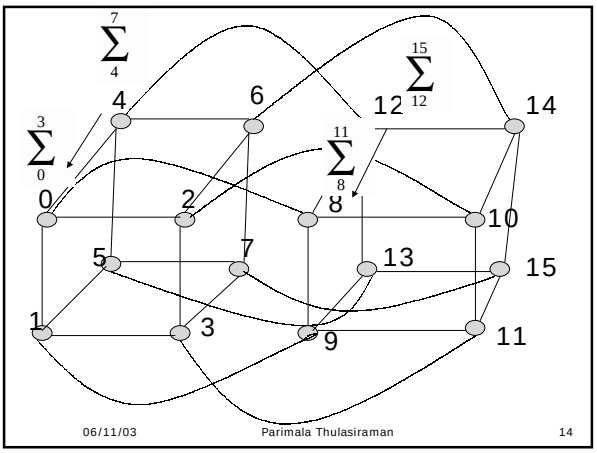
Example

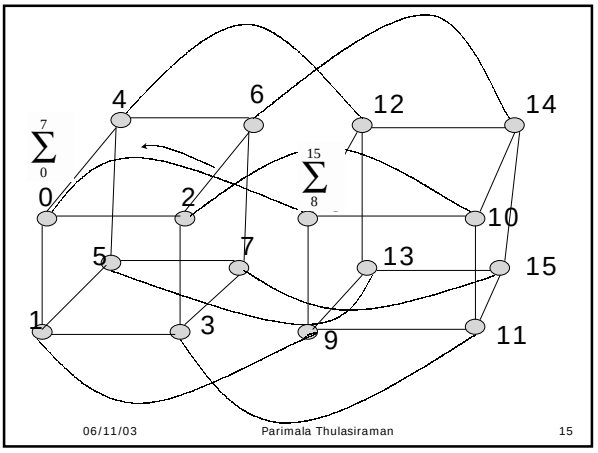
- Consists of 1 addition & 1 communication
- Processors that communicate are directly connected with each other in a hypercube; labels differ in one bit position
- Addition and communication constant amount of time

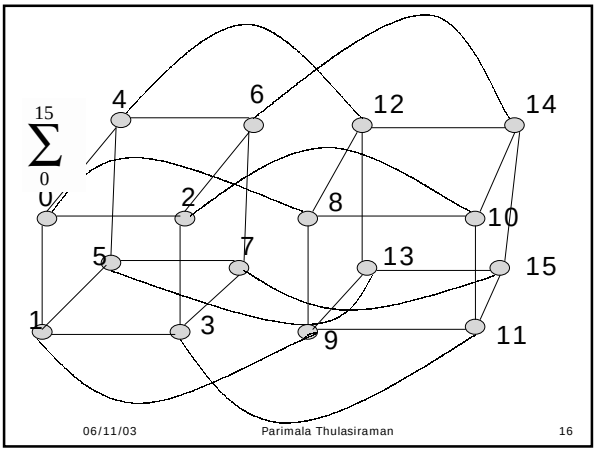
$$T_p = \Theta(\log n)$$




0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
06/11/03 Parimala Thulasiraman 13


0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
06/11/03 Parimala Thulasiraman 14


0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
06/11/03 Parimala Thulasiraman 15


0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
06/11/03 Parimala Thulasiraman 16

Example

- Addition and communication constant amount of time

$$T_p = \Theta(\log n)$$

Example

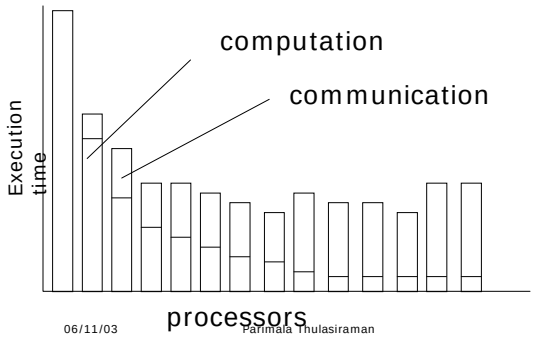
- Single processor : $T_s = \Theta(n)$
- $$S = \frac{T_s}{T_P} = \Theta(\frac{n}{\log n})$$

Theoretically, speedup cannot exceed the number of processors P. T_s Units of time to solve a sequential algorithm. A speedup of P can be obtained on
P processors if none of the processors spends more than T_s / P time.

Superlinear Speedup

- In some cases, a speedup greater than P is observed.
  - Due to non-optimal sequential algorithm
  - H/W characteristics that put the sequential algorithm at a disadvantage.
    - Data too big to fit in memory-use of secondary storage degrades performance
    - Partition data among processor, much easier to fit into local memory

For a fixed problem size there is an optimum
Number of processors that minimizes
Overall execution time



Efficiency

- Only an ideal parallel system with P processors can deliver speedup = P.
- Processors cannot devote their entire time to computation (communication,synchronization, idling)
- Efficiency: A measure of the fraction of time for which the processor is usefully employed; Ratio of speedup to the number of processors

Efficiency

- Ideal: speedup = P, efficiency = 1
- Practice: speedup < P; 0 <E < 1

Efficiency

- Example:

$$\frac{S}{P} = \frac{n/\log n}{n} = \frac{1}{\log n}$$

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

Cost

- Cost = Parallel Runtime * # of processors

$$T_P \times P$$

- Reflects the sum of the time each processor spends solving the problem

$$E = S / P = \frac{T_s / T_P}{P} = \frac{T_s}{T_P \times P} = \frac{\text{time seq. alg.}}{\text{cost}}$$

Cost = T_s / E

Cost Optimal

- Cost of solving a problem on a ||'l computer is proportional to the execution time of the fastest sequential algorithm on a single processor– Cost Optimal
- Cost optimal system has an efficiency of $\Theta(1)$
- Cost is referred to as Work or processor time product
- Example not cost optimal $\therefore E = \Theta(\frac{1}{\log n})$

$$\cos t = PT_P = n \log n$$

$$T_s = n$$

Granularity

- In the previous example we used as many processors as there are inputs.
- This is too excessive.
- In practice, we assign larger pieces of input data to processors.

Scaling Down

- Using fewer than the maximum possible number of processors to execute a parallel algorithm is called scaling down
- $P < n$
- Mapping data appropriately to maintain cost optimality

Scalability

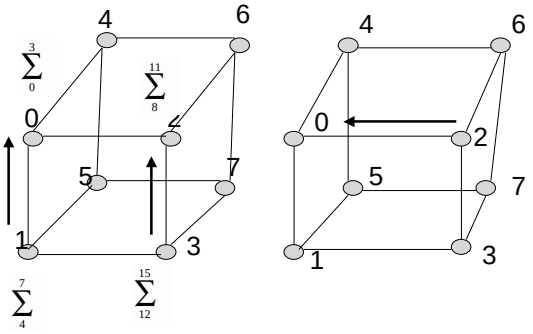
- Adding n numbers on p processors hypercube
- 1 unit time to add and 1 unit time to communicate
- n/P elements per processor
- Addition : n/P-1 time locally
- P partial sums added in log P steps, one addition, one communication

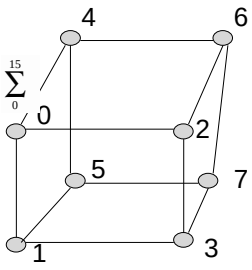
Input = 16, P =4

3	7	11	15
2	6	10	14
1	5	9	13
0	4	8	12
P0	P1	P2	P3

Each processor locally Adds its n/P elements in $\Theta(n/P)$ time.

Add P partial sums on P processors





Adding partial sums done In log(P) time.

$\Theta(\log P)$

1 communication + 1 addition = 2 log P

Compute time

T_P : $(n/P + \log P)$

Cost = T_P X P = $n + P \log P$

Serial time = n-1

S = n/(n/P+logP) = nP/(n+PlogP)

E = S/P = n/(n+PlogP)

(See paper)

The expressions can be used to calculate the speedup and efficiency of any pair of n and P.

Observation:

1. Speedup does not increase linearly as the number of processors increases
   Speedup becomes saturated, speedup curve flattens (Amdahl's law)
2. Efficiency drops with an increasing number of processors
3. Efficiency increases with *n* and same number of processors

Lots of Parallel systems exhibit this phenomena

Scalability

- Increasing number of processors reduces efficiency
- Increasing size of input, increases efficiency
- Increasing size of input and number of processors simultaneously keeps efficiency fixed.
- N=64, P=4, E = 0.80
- N=192, P=8, E = 0.80
- N=512, p=16, E=0.80
  - The ability to maintain efficiency at a fixed value by simultaneously increasing the number of processors and size of the problem is exhibited by many parallel systems. These are **scalable** parallel systems

Scalability

- Scalability: A measure of the capacity to increase speedup in proportion to number of processors
- It reflects a parallel systems ability to utilize increasing processing resources effectively.
- Isoefficiency Metric: Relate problem size to number of processors to maintain fixed efficiency.
- Question: At what rate should the problem size be increased with respect to number of processors to keep efficiency fixed? (page 5)
  - Degree of scalability