Performance and Scalability of Parallel Systems

Parallel Systems

- Sequential Algorithm: Execution time expressed as a function of the size of its input.
-

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.

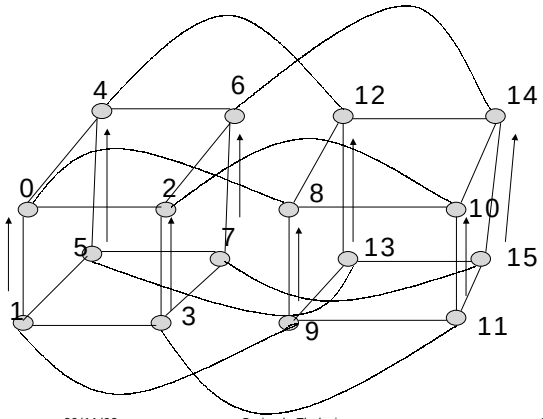
Example

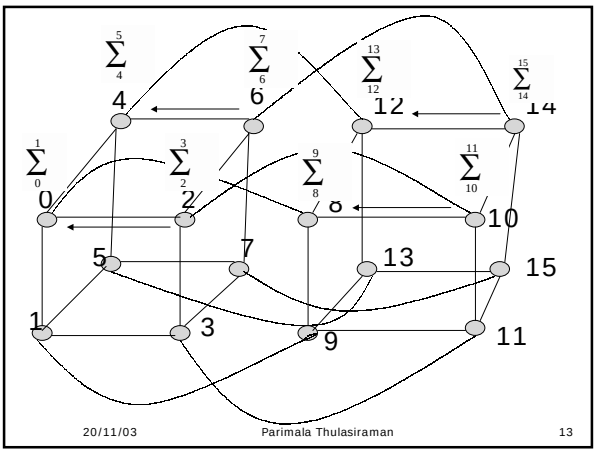
- Adding *n* numbers on an *n* processor hypercube
  - Assume *n* numbers, *n* processors
  - At the end of the computation one of the processor holds the sum
  - *n* is a power of 2.

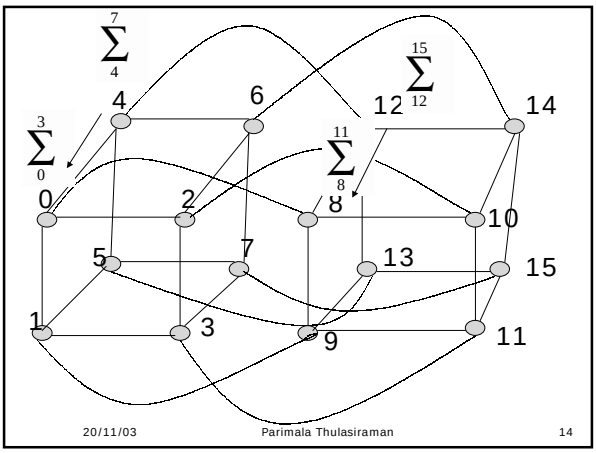
Example

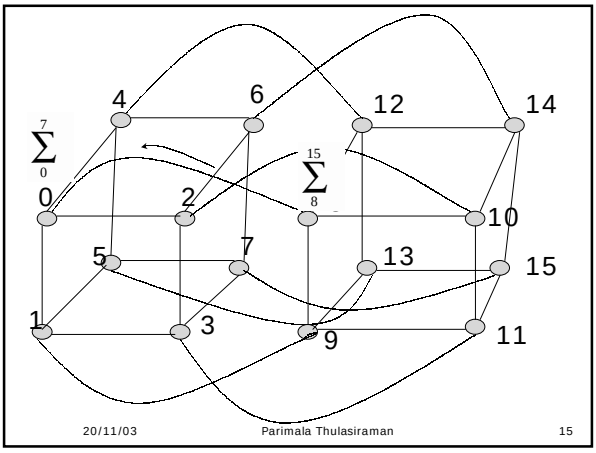
- Consists of 1 addition & 1 communication
- Processors that communicate are directly connected with each other in a hypercube; labels differ in one bit position
- Addition and communication constant amount of time

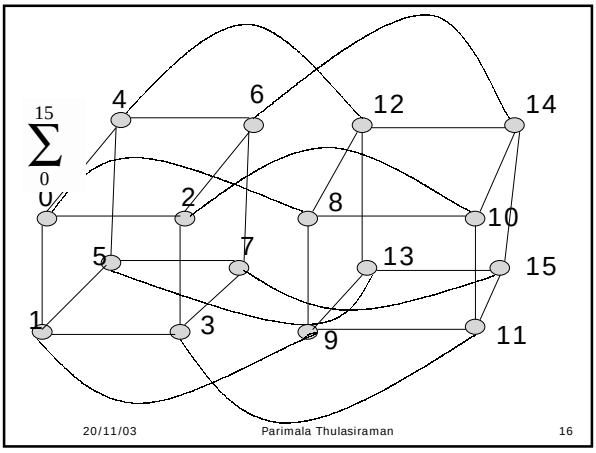
$$T_p = \Theta(\log n)$$




20/11/03
Parimala Thulasiraman
13


20/11/03
Parimala Thulasiraman
14


20/11/03
Parimala Thulasiraman
15


20/11/03
Parimala Thulasiraman
16

Example

- Addition and communication constant amount of time

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

Example

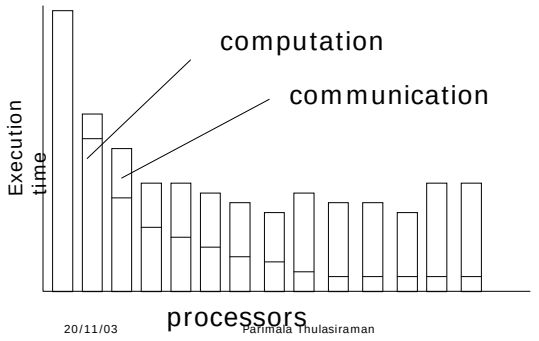
- Single processor : $T_s = \Theta(n)$
- $$S = \frac{T_s}{T_P} = \Theta(\frac{n}{\log n})$$

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

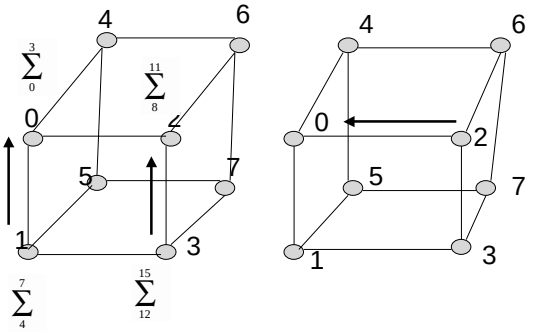
- Adding n numbers on p processors hypercube
- 1 unit time to add and 1 unit time to communicate
- n/P elements per processor
- Addition : n/P-1 time locally
- P partial sums added in log P steps, one addition, one communication

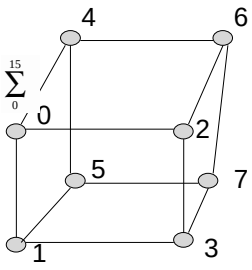
Input = 16, P =4

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

Overhead Function

- Causes for performance loss are termed **overhead (communication,idling,contention,synchronization)**
- Total overhead or overhead function: cost that is not incurred by the fastest know serial algorithm on a sequential computer $T_o(W,P)$
- Total time spent by all processors = PT_P

$$T_o = PT_P - T_1$$

Time for a seq. alg. To execute on 1 processor : T_1

- Problem size :
  - Total number of basic operations inherent in the problem.
  - is a function of the input size.
- W : problem size

$$T_o = PT_P - T_1$$

$$T_o = PT_P - Wt_c$$

Isoefficiency Function

$$T_P = \frac{Wt_c + T_0}{P}$$

Function of problem size, Overhead function, # of processors

$$S = \frac{Wt_c}{T_P} = \frac{Wt_c P}{Wt_c + T_o}$$

$$E = \frac{S}{P} = \frac{Wt_c}{Wt_c + T_o} = \frac{1}{1 + \frac{T_0}{Wt_c}}$$

Observation

$$E = \frac{S}{P} = \frac{1}{1 + \frac{PT_P - Wt_c}{Wt_c}}$$

$$= \frac{1}{1 + \frac{PT_P}{Wt_c} - 1}$$

$$= \frac{Wt_c}{PT_P}$$

- W is kept constant and P is increased, E decreases because total overhead increases with P.
- If W is increases, fixed P, E increases

Isoefficiency Function

- For different parallel systems, W must be increased at different rates w.r.t P in order to maintain fixed efficiency
- If W increases linearly w.r.t P, then scalable
- If W increases exponentially as P increases, not scalable: to get good speedup for large number of processors, problem size has to be enormous

Isoefficiency Function

- For scalable parallel systems, efficiency can be maintained at a fixed rate (between 0 and 1) if the ratio T_o/W is maintained at const. value

$$E = \frac{1}{1 + \frac{T_o}{Wt_c}}$$

$$\frac{T_o}{Wt_c} = \frac{1 - E}{E}$$

$$W = \frac{E}{1 - E} T_o t_c$$

Isoefficiency Function

- K= E/1-E, $W=KT_o$
- Growth rate of W required to keep efficiency fixed as P increases : **isoefficiency function**
- Determines the ease with which a parallel system can maintain a constant efficiency and hence achieve speedups increasing in proportion to the number of processors
- Small isoefficiency means that small increments in the problem size are sufficient for efficient utilization of an increasing number of processors.Parallel system is highly scalable.
- High isoefficiency, poorly scalable

Example

$$T_P = \frac{n}{P} - 1 + 2 \log P$$

$$T_P = T_1 + T_0$$

$$T_0 = 2 \log P$$

W = 2KPlogP
The asymptotic isoefficiency function Is $\Theta(P \log P)$
If P is increased to P' , the problem
Size, n, must be increased by a factor
of P'logP'/PlogP to increase speedup by a factor of P'/P
(to get the same efficiency as on P processors).

Complicated Expression

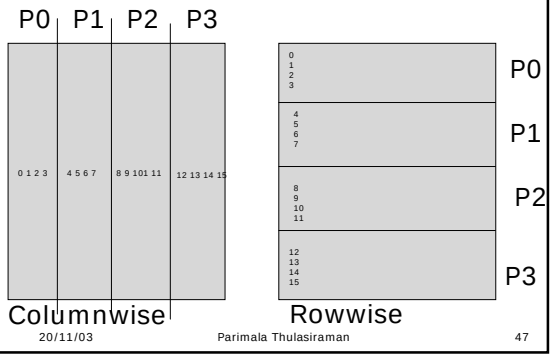
$$T_o = P^{3/2} + P^{3/4}W^{3/4}$$
$$W = KP^{3/2} = \Theta(P^{3/2})$$
$$W = KP^{3/4}W^{3/4}$$
$$= K^4P^3 = \Theta(P^3)$$

Take the higher of the 2 rates.

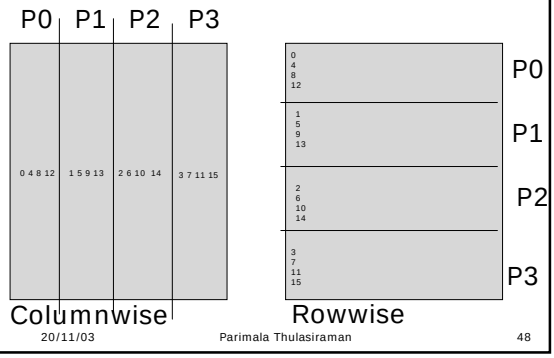
Isoefficiency Funtion

- Captures the characteristics of the parallel algorithm as well as the parallel architecture.
- Characterizes the amount of parallelism inherent in the algorithm
- Studies in h/w parameters: speed of processors and communication channels

Matrix-Vector Multiplication (Block-Stripping)



Matrix-Vector Multiplication (Cyclic-Stripping)



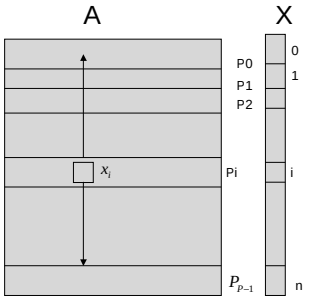
Matrix-Vector Multiplication

- Sequential Time :
  - Requires n^2 Additions and multiplications

$$W = n^2$$

$$T_1 = n^2 t_c = \Theta(n^2)$$

One row per processor –Simple case (Rowwise stripping)



- Each processor stores one complete row. Assume n processors
- nx1 vector is distributed- each processor stores one element
- Each processor requires entire vector to compute
  - All-to-all broadcast of vector– each processor broadcasts each of its elements
- Each processor computes:

$$P_i : y_i = \sum_{j=0}^{n-1} A[i, j]x[j]$$

Parallel Run time

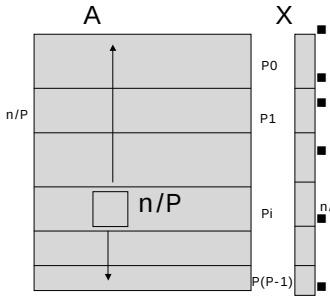
- Each processor broadcasts x : $\Theta(n)$
- Multiplication of a single row of A with X by each processor is $\Theta(n)$
- Addition of a single row of A with X by each processor is $\Theta(n)$

$$T_P = \Theta(n)$$

$$\text{cost} = P \times T_P = n \times n = n^2$$

Cost optimal: $T_1 = \Theta(n^2)$

P < n processors

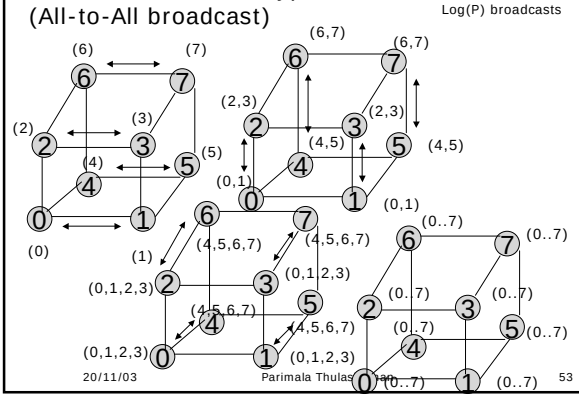


- Partitioning A : n/P rows per processor
- X partitioned: n/P
- Distribute n/P vector size
- All-to-all broadcast among P processors of size n/P
- Compute : n/P rows multiply by n/P vectors
- Result: n/P vectors

Parallel Runtime on hypercube (All-to-All broadcast)

Broadcast on All dimensions

Log(P) broadcasts



All-to-All broadcast

- Log P steps.
- Communication along every dimension
- Each step data size doubles and is transmitted
- Size of message exchanged at *ith* step is $2^{i-1}m$
- Time to send and receive messages $t_s + t_w 2^{i-1} m$
- To complete the entire iteration:

$$T_{total} = \sum_{i=1}^{\log P} (t_s + 2^{i-1} t_w m)$$
$$= \log P t_s + m(P-1)t_w$$
$$= \log P t_s + (\frac{n}{P})(P-1)t_w$$
$$= \log P t_s + n t_w, n >> P$$

t_s : *startup*

t_w : per word transfer time

Computation time

- An addition and multiplication takes times t_c
- Each processor: dot product of (n/P) elements and vector n.

$$\therefore \quad \frac{n^2}{P} t_c$$

$$T_P = t_c \frac{n^2}{P} + (t_s \log P + n t_w)$$

$$S = \frac{T_1}{T_P} = \frac{t_c n^2}{t_c \frac{n^2}{P} + (t_s \log P + n t_w)}$$

$$= \frac{P}{1 + P(t_s \log P + t_w n)/t_c n^2}$$

$$E = \frac{S}{P}$$

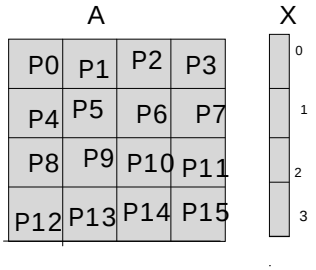
$$E = \frac{1}{1 + P(t_s \log P + t_w n)/t_c n^2}$$

Scalability Analysis (isoefficiency function)

- Rate at which problem size needs to increase with number of processors to maintain fixed E : $\Theta(P^2)$

$$T_o = PT_P - T_1 = n^2 t_c + t_s P \log P + t_w nP - n^2 t_c$$
$$= t_s P \log P + t_w nP$$
$$W = KT_o = Kt_s P \log P$$
$$W = Kt_w nP$$
$$= n^2 = Kt_w nP$$
$$n = Kt_w P \quad \text{(substitute into W)}$$
$$W = Kt_w{}^2 P^2$$

Block CheckerBoard Partitioning

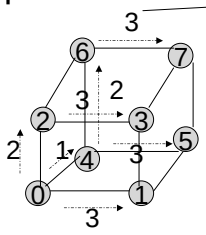


Assume A is a 4x4 Matrix
X is of size 4

One element per processor

- Assume n^2 processors
- Each processor stores a single element
- *ith* element of x is available at *ith* element of each row in matrix
- Distribute x to the last column.
- Align the x in the diagonal
- Copy the vector from each diagonal to the other processors—n simultaneous (one to all broadcast)
- Each processor multiplies each element with x element.
- Result : add each of the computed product—leave the sum in the last column of processors—single node accumulation

Parallel run time on a hypercube



If source is 0
Sends to all

$\log \sqrt{n^2}$ communication steps P= n^2

Each log n step takes $t_s + t_w m$ in each dimension –for all dimensions $\log n(t_s + t_w m)$

3 communication steps

- Align x along main diagonal : $\Theta(\log n)$
- Broadcast columnwise to selected processors: $\Theta(\log n)$
- Single node accumulation: $\Theta(\log n)$
- Single multiplication in constant time

$$\therefore T_P = \Theta(\log n)$$

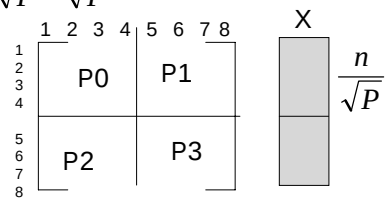
$$\text{cost} = PT_P = n^2 \log n$$

Not cost optimal

$$T_1 = n^2$$

P << n

$\frac{n}{\sqrt{P}} \times \frac{n}{\sqrt{P}}$ blocks per processor



Parallel run time

- Vector is aligned along diagonal
- One-to-all broadcast of the vector
- Each performs computation on its

$$\frac{n}{\sqrt{P}} \times \frac{n}{\sqrt{P}} = \frac{n^2}{P}$$

Data size vector

Node accumulation of the partial sums

Parallel run time

- Vector to main diagonal from last processor in each column:

$$t_s + t_w \frac{n}{\sqrt{P}} \log \sqrt{P}$$

- One-to-all broadcast:

$$t_s \log \sqrt{P} + t_w \frac{n}{\sqrt{P}} \log \sqrt{P}$$

- Single node accumulation:

$$t_s \log \sqrt{P} + t_w \frac{n}{\sqrt{P}} \log \sqrt{P}$$

Parallel run time

- computation: $t_c \frac{n^2}{P}$

$$T_P = t_c \frac{n^2}{P} + 2t_s \log\sqrt{P} + 3t_w \frac{n}{\sqrt{P}} \log\sqrt{P}$$

$$= t_c \frac{n^2}{P} + t_s \log P + \frac{3}{2} t_w \frac{n}{\sqrt{P}} \log P$$

Scalability analysis

$$T_P = t_c \frac{n^2}{P} + 2t_s \log\sqrt{P} + 3t_w \frac{n}{\sqrt{P}} \log\sqrt{P}$$

$$cost = T_P P = t_c n^2 + \underbrace{t_s P \log P + \frac{3}{2} t_w n \sqrt{P} \log P}_{T_O}$$

Isoefficiency function

- Determine the isoefficieny as we did last time for both startup and per word time parts. (see paper for details)

$$\Theta(P \log^2 P)$$

Parallel Overhead

- Major sources of overhead in a parallel system:
  - Interprocessor communication
  - Load imbalance
  - Extra computation
- Interprocessor communication: if P processor spend t_{comm} time performing communication, then interprocessor communication contributes t_{comm} x P to the overhead function

Load Imbalance

- If different processors have different work load, some processors may be idle during part of the time that others are working on the problem.
  - Statically partitioning the problem is difficult
  - Having equal workload is difficult
- Synchronization: If not all ready to synchronize, time wasted
- Idle time is an overhead
- In some applications there is a sequential part that only one processor can work on.
- W_s, work due to sequential component, W_P is the work due to parallelizable component. While one processor works on W_s the remaining (p-1) are idle.
- A serial component contributes (P-1) W_s to the overhead function of a P-processor parallel system.

Extra Computation

- Fastest sequential algorithm maybe difficult or impossible to parallelize, forcing us to use a parallel algorithm based on a poorer but easily parallelizable sequential one (with high degree of concurrency)
- Let X be the execution time of the fastest sequential alg. Let X' be the execution time of the poorer algorithm (easily parallelizable) for that problem
- X-X' = overhead, extra amount of work performed to solve the problem