Parallel and Distributed Computing

01: How can you define Parallelism? Describe briefly different types of parallelism with example?
02: Why use parallelism? Define the ways of measuring speed of computers. How parallelism is used in super-computing?
03: Give the classification of parallel machine. Define different types of parallel machine(block diagram, example).
04: What is SIMD machine? Define with example.
05: What do you mean by inter process communication in parallel machine? Explain the shared memory conflicts in inter process communication. How it can be solved?
06: Give the classification of shared memory computer with definition. What are the limitation of shared memory?
07. Define inter process communication with message passing. Differentiate between shared and Distributed memory.
08. Summing M numbers with N processors using shared and distribute memory.
09. Suppose, there are N processors to search a list S={L1, L2....Lm}. $1<N\leq m$. Show how the different Sub-classes of shared memory behave to solve this problem.
10. Name different inter connection Networks and the metrics used to evaluate the networks.
11. Define different inter connection with networks with diagram.
12. How pipeline algorithm works?
13. Define the factors that limit the speed-up of parallel algorithms.
14. Calculate the speed-up and efficiency of a problem of summing m numbers on squire mash of N processors.
15. Suppose and iterative algorithm for solving non-liner equations f(x)=0, Where _____ .Explain the asynchronous parallel mode of the algorithm with example, if t=2, t=3 and t=1. Ci indicates processor is using x in its calculation. Show upto 10time unit.
16. Name and define the scalability of parallel algorithms.
17. State Amdahl law with derivation and example.
18. Write the implications of Amdahl law.
19. Explain the semantics of message sends and receives.

01: How can you define Parallelism? Describe briefly different types of parallelism with example?

Ans: Two/More processor(Core) executes Two/More process/events at the same time(simultaneously) is called Parallelism.

- An event typically means one of the following:
  - An arithmetical operation
  - A logical operation
  - Accessing memory
  - Performing input or output (I/O)

Types of Parallelism

- Parallelism can be examined at several levels.
  - **Job level**: several independent jobs simultaneously run on the same computer system.
  - **Program level**: several tasks are performed simultaneously to solve a single common problem.
  - **Instruction level**: the processing of an instruction, such as adding two numbers, can be divided into sub-instructions. If several similar instructions are to be performed their sub-instructions may be overlapped using a technique called *pipelining*.
  - **Bit level**: when the bits in a word are handled one after the other this is called a *bit-serial* operation. If the bits are acted on in parallel the operation is *bit-parallel.*

In this parallel processing course we shall be mostly concerned with parallelism at the program level. *Concurrent processing* is the same as parallel processing.

=======

02: Why use parallelism? Define the ways of measuring speed of computers. How parallelism is used in super-computing?

Ans:

- Better utilisation of resources. Want to keep hardware busy.
- Want to run programs faster by spreading work over several processors.
- Ways to measure speed:
  - Floating point operations per second. 1 Mflop/s is one million floating point operations per second.
    - High performance workstation » 10-20 Gflop/s
    - Current best supercomputer » 1 Pflop/s
  - Transactions per second. 1 Tps is one transaction per second.
  - Instructions per second. 1 Mips is one million instructions per second.

How parallelism is used in super-computing?

- **Weather forecasting**. Currently forecasts are usually accurate up to about 5 days. This should be extended to 8 to 10 days over the next few years. Researchers would like to better model local nonlinear phenomena such as thunderstorms and tornadoes.
- **Climate modelling**. Studies of long-range behaviour of global climate. This is relevant to investigating global warming.
- **Engineering**. Simulation of car crashes to aid in design of cars. Design of aircraft in "numerical wind tunnels.'"
- **Material science**. Understanding high temperature superconductors. Simulation of semiconductor devices. Design of lightweight, strong materials for construction.
- **Drug design**. Prediction of effectiveness of drug by simulation. Need to know configuration and properties of large molecules.

==========

03: Give the classification of parallel machine. Define different types of parallel machine(block diagram, example).

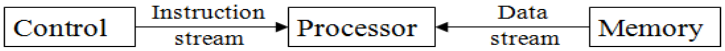
Ans:

- Depending on whether there is one or several of these streams we have 4 classes of computers.
  - Single Instruction Stream, Single Data Stream: SISD
  - Multiple Instruction Stream, Single Data Stream: MISD
  - Single Instruction Stream, Multiple Data Stream: SIMD
  - Multiple Instruction Stream, Multiple Data Stream: MIMD

SISD Computers

This is the standard sequential computer.

A single processing unit receives a single stream of instructions that operate on a single stream of data



Example:

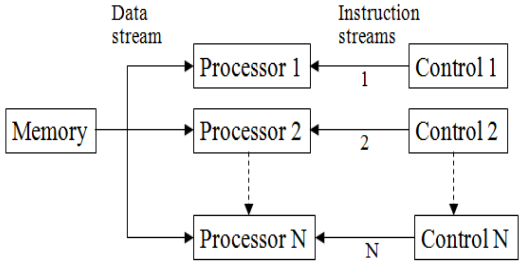
To compute the sum of N numbers $a_1, a_2, \ldots, a_N$ the processor needs to gain access to memory N consecutive times. Also N-1 additions are executed in sequence. Therefore the computation takes O(N) operations

Algorithms for SISD computers do not contain any process parallelism since there is only one processor.

26

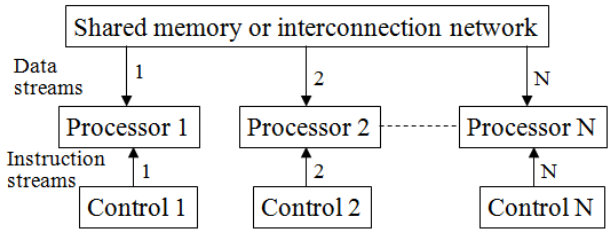
MISD Computers

N processors, each with its own control unit, share a common memory.



MIMD Computers

This is the most general and most powerful of our classification. We have N processors, N streams of instructions, and N streams of data.



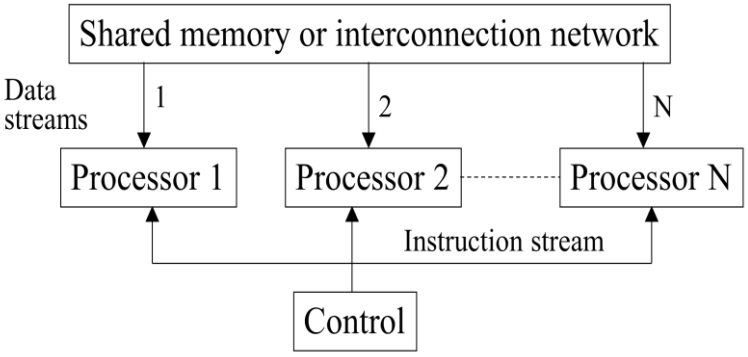
=======

04: What is SIMD machine? Define with example.

Ans:

- All N identical processors operate under the control of a single instruction stream issued by a central control unit.
- There are N data streams, one per processor, so different data can be used in each processor.

SIMD Example



Problem: add two 2×2 matrices on 4 processors.

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} + \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} = \begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{bmatrix}$$

The same instruction is issued to all 4 processors (add two numbers), and all processors execute the instructions simultaneously. It takes one step to add the matrices, compared with 4 steps on a SISD machine.

Example:

- In this example the instruction is simple, but in general it could be more complex such as merging two lists of numbers.
- The data may be simple (one number) or complex (several numbers).
- Sometimes it may be necessary to have only a subset of the processors execute an instruction, i.e., only some data needs to be operated on for that instruction. This information can be encoded in the instruction itself indicating whether
  - the processor is active (execute the instruction)
  - the processor is inactive (wait for the next instruction)

=========

05: What do you mean by inter processor communication in parallel machine? Explain the shared memory conflicts in inter processor communication. How it can be solved?

Ans:

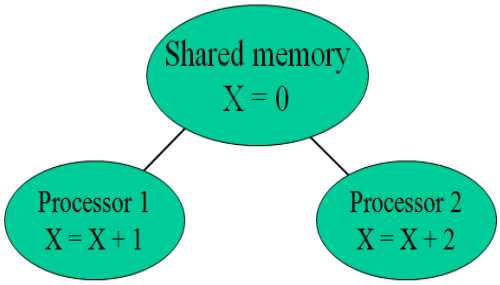
- Usually a parallel program needs to have some means of sharing data and results processed by different processors. There are two main ways of doing this
  1. Shared Memory
  2. Message passing
- Shared memory consists of a global address space. All processors can read from and write into this global address space.

Shared memory conflicts in inter processor communication.How it can be solved

Shared Memory Conflicts 1

- The shared memory approach is simple but can lead to problems when processors simultaneously access the same location in memory.
- *Example:*
- Suppose the shared memory initially holds a variable x with value 0. Processor 1 adds 1 to x and processor 2 adds 2 to x. What is the final value of x?
- *You should have met this problem before when studying locks and critical sections in the operating systems module.*

Shared Memory Conflicts 2



The following outcomes are possible

1. If P1 executes and completes x=x+1 before P2 reads the value of x from memory then x is 3. Similarly, if P2 executes and completes x=x+2 before P1 reads the value of x from memory then x is 3.
2. If P1 or P2 reads x from memory before the other has written back its result, then the final value of x depends on which finishes last.
   - if P1 finishes last the value of x is 1
   - if P2 finishes last the value of x is 2

Non-Determinancy

- Non-determinancy is caused by *race conditions*.
- A race condition occurs when two statements in concurrent tasks access the same memory location, at least one of which is a write, and there is no guaranteed execution ordering between accesses.
- The problem of non-determinancy can be solved by synchronising the use of shared data. That is if x=x+1 and x=x+2 were mutually exclusive then the final value of x would always be 3.
- Portions of a parallel program that require synchronisation to avoid non-determinancy are called *critical sections*.

Locks and Mutual Exclusion

In shared memory programs *locks* can be used to give mutually exclusive access.

```
Processor 1:
        LOCK (X)
                X = X + 1
        UNLOCK (X)
```

```
Processor 2:
        LOCK (X)
                X = X + 2
        UNLOCK (X)
```

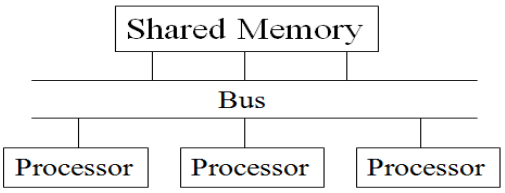
==========

06: Give the classification of shared memory computer with definition. What is the limitation of shared memory?

1. **Exclusive Read, Exclusive Write** (EREW)
   - Access to memory locations is exclusive, i.e., no 2 processors are allowed to simultaneously read from or write into the same location.
2. **Concurrent Read, Exclusive Write** (CREW)
   - Multiple processors are allowed to read from the same location, but write is still exclusive, i.e., no 2 processors are allowed to write into the same location simultaneously.
3. **Exclusive Read, Concurrent Write** (ERCW)
   - Multiple processors are allowed to write into the same location, but read access remains exclusive.
4. **Concurrent Read, Concurrent Write** (CRCW)
   - Both multiple read and write privileges are allowed.

Limitation of shared memory

- Shared memory computers are often implemented by incorporating a fast bus to connect processors to memory.



- However, because the bus has a finite bandwidth, i.e., it can carry only a certain maximum amount of data at any one time, then as the number of processors increase the *contention* for the bus becomes a problem. So it is feasible to build shared memory machines with up to only about 100 processors.

56

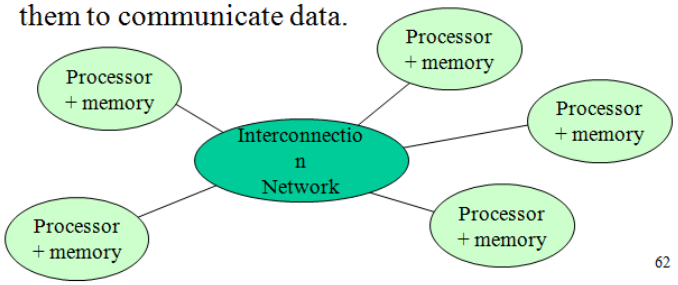
=======

07.Define inter process communication with message passing. Differentiate between shared and distributed memory.

Ans:

Interconnection Networks and Message Passing

In this case each processor has its own private (local) memory and there is no global, shared memory. The processors need to be connected in some way to allow them to communicate data.



62

Message Passing

- If a processor requires data contained on a different processor then it must be explicitly passed by using communication instructions, e.g., send and receive.

```
P1                      P2
receive (x, P2)     send (x, P1)
```

- The value x is explicitly passed from P2 to P1. This is known as *message passing*.

63

Comparison of Shared and Distributed Memory

Distributed memory	Shared memory
Large number of processors (100's to 1000's)	Moderate number of processor (10's to 100)
High peak performance	Modest peak performance
Unlimited expansion	Limited expansion
Difficult to fully utilise	Relatively easy to fully utilise
Revolutionary parallel computing	Evolutionary parallel computing

65

08. Summing M numbers with N processors using shared and distribute memory.

Ans:

Summing m Numbers

Example: summing m numbers

On a sequential computer we have,

```
sum = a[0];
for (i=1;i<m;i++) {
        sum = sum + a[i];
}
```

Would expect the running time be roughly proportional to m. We say that the running time is Θ(m).

Summing Using Shared Memory

The m numbers, and the global sum, are held in global shared memory.

```
global_sum = 0;
for (each processor){
        local_sum = 0;
        calculate local sum of m/N numbers
        LOCK
                global_sum = global_sum + local_sum;
        UNLOCK
}
```

70

Summing m Numbers in Parallel

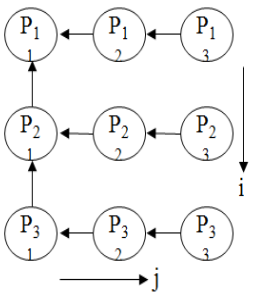
- What if we have N processors, with each calculating the m/N numbers assigned to it?
- We must add these partial sums together to get the total sum.

Notes on Shared Memory Algorithm

- Since global_sum is a shared variable each processor must have mutually exclusive access to it – otherwise the final answer may be incorrect.
- The running time (or *algorithm time complexity*) is
  Θ(m/N)+ Θ (N)
- where
  - m/N comes from finding the local sums in parallel
  - N comes from adding N numbers in sequence

Summing Using Distributed Memory

Suppose we have a square mesh of N processors.



The algorithm is as follows:

1. Each processor finds the local sum of its m/N numbers
2. Each processor passes its local sum to another processor in a coordinated way
3. The global sum is finally in processor P_{11}.

Distributed Memory Algorithm

The algorithm proceeds as follows:

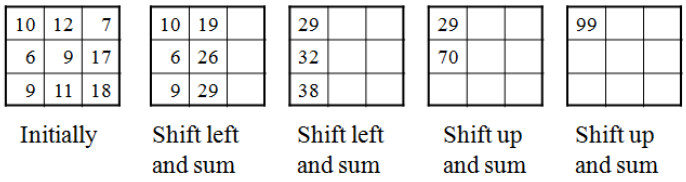
1. Each processor finds its local sum.
2. Sum along rows:
   a) If the processor is in the rightmost column it sends its local sum to the left.
   b) If the processor is not in the rightmost or leftmost column it receives the number form the processor on its right, adds it to its local, and send the result to the processor to the left.
   c) If the processor is in the leftmost column it receives the number from the processor on its right and adds it to it local sum to give the row sum.
3. Leftmost column only – sum up the leftmost column:
   a) If the processor is in the last row send the row sum to the processor above
   b) If the processor is not in the last or first row receive the number from the processor below, add it to the row sum, and send result to processor above
   c) If the processor is in the first row receive the number from the processor below. This is the global sum.

Summing Example

There are $\sqrt{N}$-1 additions and $\sqrt{N}$-1 communications in each direction, so the total time complexity is

$$\Theta(m/N) + \Theta(\sqrt{N}) + C$$

where C is the time spent communicating.



09. Suppose, there are N processors to search a list S={L1, L2....Lm}. 1<N≤m. Show how the different Sub-classes of shared memory behave to solve this problem.

Ans:

To show how the 4 subclasses of shared memory machines behave, consider the following example.

Problem:

We have N processors to search a list S = {L_1, L_2, …,L_m} for the index of a given element x. Assume x may appear several times, and any index will do. 1< N≤m.

The Algorithm

```
procedure SM_search (S, x, k)
  STEP 1:  for i=1 to N do in parallel
              read x
           end for
  STEP 2: for i=1 to N do in parallel
              S_i = {L_((i-1)m/N+1), …,L_(im/N)}
              perform sequential search on sublist S_i
              (return K_i= -1 if not in list, otherwise index)
           end for
  STEP 3: for i=1 to N do in parallel
              if K_i > 0 then k=K_i end if
           end for
end procedure
```

10. Name different inter connection Networks and the metrics used to evaluate the networks.

Ans:

Examples of Inter Connected-Networks

Important networks include:

- fully connected or all-to-all
- mesh
- ring
- hypercube
- shuffle-exchange
- butterfly
- cube-connected cycles

Network Metrics

A number of metrics can be used to evaluate and compare interconnection networks.

- **Network connectivity** is the minimum number of nodes or links that must fail to partition the network into two or more disjoint networks.
- Network connectivity measures the resiliency of a network, and its ability to continue operation despite disabled components. When components fail we would like the network to continue operation with reduced capacity.

77

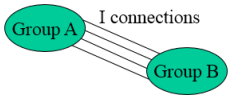
Network Metrics 2

- **Bisection width** is the minimum number of links that must be cut to partition the network into two equal halves (to within one). The *bisection bandwidth* is the bisection width multiplied by the data transfer rate of each link.
- **Network diameter** is the maximum internode distance, i.e., the maximum number of links that must be traversed to send a message to any node along the shortest path. The lower the network diameter the shorter the time to send messages to distant nodes.

Network Metrics 3

Network narrowness measures congestion in a network.

- Partition the network into two groups A and B, containing NA and NB nodes, respectively, with NB < NA.
- Let I be the number on connections between nodes in A and nodes in B. The narrowness of the network is the maximum value of NB/I for all partitionings of the network.



- If the narrowness is high (NB>I) then if the group B nodes want to communicate with the group A nodes congestion in the network will be high.

Network Metrics 4

- **Network Expansion Increment** is the minimum number of nodes by which the network can be expanded.
  - A network should be expandable to create larger and more powerful parallel systems by simply adding more nodes to the network.
  - For reasons of cost it is better to have the option of small increments since this allows you to upgrade your machine to the required size.
- **Number of edges per node**. If this is independent of the size of the network then it is easier to expand the system.

11. Define different inter connection with networks with diagram.

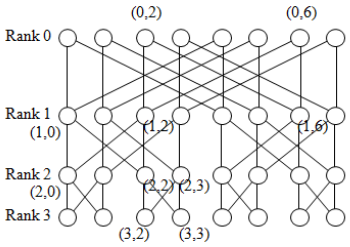
Ans:

Cube-Connected Cycles Network

- A cube-connected cycles network is a k-dimensional hypercube whose 2^k vertices are actually cycles of k nodes.
- The advantage compared with the hypercube is that the number of edges per node is a constant, 3.
- Disadvantages are that network diameter is twice that of a hypercube, and the bisection width is lower.

Example of a Butterfly Network

Here is a butterfly network for k = 3.



$i = 1, j = 2 = (010)_2, j' = (110)_2 = 6$

$i = 2, j = 2 = (010)_2, j' = (000)_2 = 0$

$i = 3, j = 2 = (010)_2, j' = (011)_2 = 3$

Butterfly Network

- A butterfly network consists of $(k+1)2^k$ nodes divided into k+1 rows, or *ranks*.
- Let node (i,j) refer to the jth node in the ith rank. Then for i > 0 node (i,j) is connected to 2 nodes in rank i-1, node (i-1,j) and node (i-1,m), where m is the integer found by inverting the ith most significant bit of j.
- Note that if node (i,j) is connected to node (i-1,m), then node (i,m) is connected to node (i-1,j). This forms a butterfly pattern.
  - Network diameter = 2k
  - Bisection width = 2^k

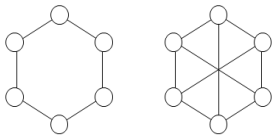
Mapping Grids to Hypercubes

- In the example in which we summed a set of numbers over a square mesh of processors each processor needs to know where it is in the mesh.
- We need to be able to map node numbers to locations in the process mesh
  - Given node number k what is its location (i,j) in the processor mesh?
  - Given a location (i,j) in the processor mesh what is the node number, k, of the processor at that location?
  - We want to choose a mapping such that neighbouring processes in the mesh are also neighbours in the hypercube. This ensures that when neighbouring processes in the mesh communicate, this entails communication between neighbouring processes in the hypercube.

90

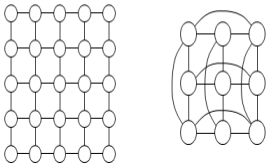
Examples of Ring Networks

- Here are a simple ring and a chordal ring with diametric links, each of size 6 nodes.



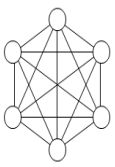
Mesh Networks

- In a mesh network nodes are arranged as a q-dimensional lattice, and communication is allowed only between neighboring nodes.
- In a *periodic mesh*, nodes on the edge of the mesh have wrap-around connections to nodes on the other side. This is sometimes called a *toroidal mesh*.



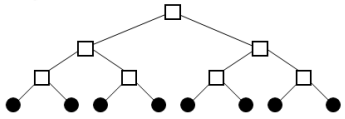
Fully Connected Network

- In the fully connected, or all-to-all, network each node is connected directly to all other nodes.
- This is the most general and powerful interconnection network, but it can be implemented for only a small number of nodes.



Complete Binary Tree Network

- Tree-based networks use switches to connect processors. An example is the binary tree network.



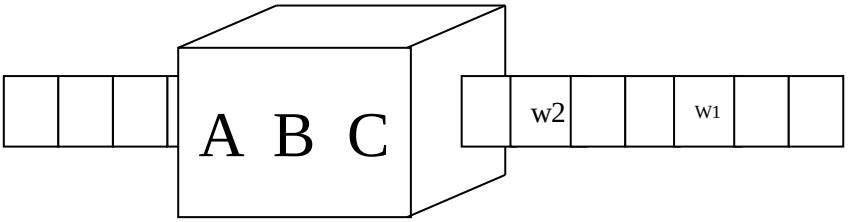
•This has a bisection width of 1, and a connectivity of 1. The low bisection width can result in congestion in the upper levels of the network.

108

12. How Pipelined Algorithm works?

Ans:

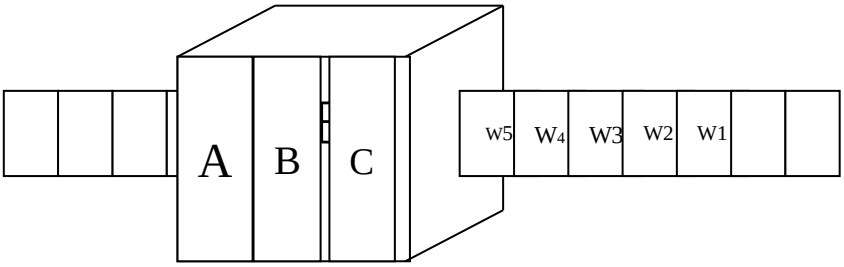
- A pipelined algorithm involves an ordered set of processes in which the output from one process is the input for the next.
- The input for the first process is the input for the algorithm.
- The output from the last process is the output of the algorithm.
- Data flows through the pipeline, being operated on by each process in turn.
- Example: Suppose it takes 3 steps, A, B, and C, to assemble a widget, and each step takes one unit of time.
- In the sequential case it takes 3 time units to assemble each widget.
- Thus it takes 3n time units to produce n widgets.



- In the pipelined case the following happens
  - Time step 1: A operates on W1
  - Time step 2: A operates on W2, B operates on W1
  - Time step 3: A operates on W3, B operates on W2, C completes W1
  - Time step 4: A operates on W4, B operates on W3, C completes W2

Pipelined Algorithm

- After 3 time units, a new widget is produced every time step.
- If the pipeline is n processes long, a new widget is produced every time step from the nth time step onwards. We then say the pipeline is *full*.
- The pipeline *start-up time* is n-1.
- This sort of parallelism is sometimes called *algorithmic parallelism*.



13. Define the factors that limit the speed-up of parallel algorithms.

--Factors That Limit Speed-up

1. Software Overhead

Even when the sequential and parallel algorithms perform the same computations, software overhead may be present in the parallel algorithm. This includes additional index calculations necessitated by how the data were decomposed and assigned to processors, and other sorts of "bookkeeping" required by the parallel algorithm but not the sequential algorithm.

02. Load Balance

Each processor should be assigned the same amount of work to do between synchronisation points. Otherwise some processors may be idle while waiting for others to catch up. This is known as load imbalance. The speedup is limited by the slowest processor.

03. Communication Overhead

Assuming that communication and calculation cannot be overlapped, then any time spent communicating data between processors reduces the speed-up.

14. Calculate the speed-up and efficiency of a problem of summing m numbers on squire mash of N processors.

- We now define some metrics which measure how effectively an algorithm exploits parallelism.
- **Speed-up** is the ratio of the time taken to run the best sequential algorithm on one processor of the parallel machine divided by the time to run on N processors of the parallel machine.

  $S(N) = T_{seq}/T_{par}(N)$
- **Efficiency** is the speed-up per processor.

  $e(N) = S(N)/N=(1/N)(T_{seq}/T_{par}(N))$
- **Overhead** is defined as $f(N) = 1/ e(N) - 1$

Example:

- Suppose the best known sequential algorithm takes 8 seconds, and a parallel algorithm takes 2 seconds on 5 processors. Then

  Speed-up = 8/2 = 4

  Efficiency = 4/5 = 0.8

  Overhead = 1/0.8 – 1 = 0.25

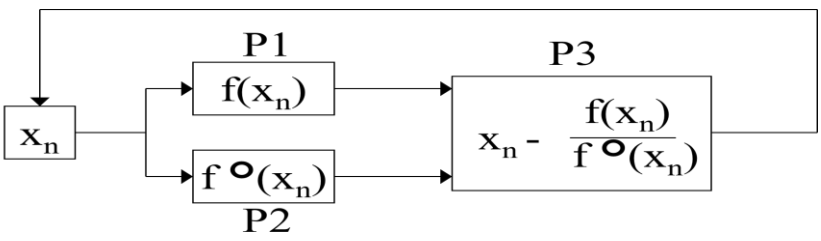
15. Suppose and iterative algorithm for solving non-liner equations f(x)=0, Where
.Explain the asynchronous parallel mode of the algorithm with example, if t1=2, t2=3 and t3=1. Ci indicates processor is using x in its calculation. Show upto 10time unit.

The Newton-Raphson method is an iterative algorithm for solving non-linear equations f(x)=0.

$x_{n+1} = x_n - f(x_n) / f'(x_n)$ generates a sequence of approximations to the root, starting with some initial value x_0.

Suppose we have 3 processors

- P1: given x, P1 calculates f(x) in t_1, and sends it to P3.
- P2: given y, P2 calculates f '(y) in t_2, and sends it to P3.
- P3: given a, b, and c, P3 calculates d = a - b/c.
  If |d - a| > e then d is sent to P1 and P2; otherwise it is output.



Parallel mode:

- P1 and P2 compute $f(x_n)$ and $f^{¢}(x_n)$ simultaneously, and when *both* have finished the values of $f(x_n)$ and $f^{¢}(x_n)$ are used by P3 to find x_{n+1}.
- Time per iteration is $\max(t_1,t_2) + t_3$.
- k iterations are necessary so the total time is, $k(\max(t_1,t_2) + t_3)$.
- P1 and P2 begin computing as soon as they receive a new input value from P3.
- P3 computes a new value as soon as it receives a new input value from *either* P1 *or* P2.

Asynchronous Parallel Mode Example

- For example, if $t_1=2$, $t_2=3$ and $t_3=1$.
- Ci indicates processor is using x_i in its calculation.
- Cannot predict number of iterations.

Time	P1	P2	P3
1	C0	C0	–
2	$f(x_0)$	C0	–
3	–	$f'(x_0)$	
4	–	–	$x_1=x_0-f(x_0)/f'(x_0)$
5	C1	C1	–
6	$f(x_1)$	C1	–
7	–	$f'(x_1)$	$x_2=x_1-f(x_1)/f'(x_0)$
8	C2	C2	$x_3=x_2-f(x_1)/f'(x_1)$
9	$f(x_2)$	C2	–
10	C3	$f'(x_2)$	$x_4=x_3-f(x_2)/f'(x_1)$
11	$f(x_3)$	C4	$x_5=x_4-f(x_2)/f'(x_2)$
12	C5	C4	$x_6=x_5-f(x_3)/f'(x_2)$

16. Name and define the scalability of parallel algorithms.

Ans:

- Scalability is a measure of how effectively an algorithm makes use of additional processors.
- An algorithm is said to be *scalable* if it is possible to keep the efficiency constant by increasing the problem size as the number of processors increases.
- An algorithm is said to be *perfectly scalable* if the efficiency remains constant when the problem size and the number of processors increase by the same factor.
- An algorithm is said to be *highly scalable* if the efficiency depends only weakly on the number of processors when the problem size and the number of processors increase by the same factor.

-

- The summing algorithm is scalable since we can take g µ ÖN.
- The summing algorithm is not perfectly scalable, but it is highly scalable.
- "Problem size" may be either:
  - the work performed, or
  - the size of the data.

17. State Amdahl law with derivation and example.

Ans:

- Amdahl's Law states that the maximum speedup of an algorithm is limited by the relative number of operations that must be performed sequentially, i.e., by its *serial fraction*.
- If a is the serial fraction, n is the number of operations in the sequential algorithm, and N the number of processors, then the time for the parallel algorithm is:

  $T_{par}(N) = (an + (1-a)n/N)t + C(n,N)$

  where C(n,N) is the time for overhead due to communication, load balancing, etc., and t is the time for one operation.

Derivation:

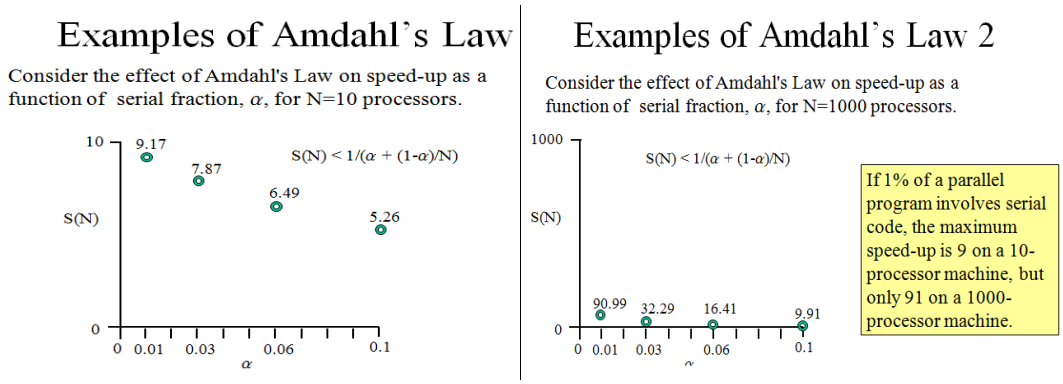
- The speed-up satisfies:

  $S(N) = Tseq/Tpar(N) = nt/[(an + (1-a)n/N)t + C(n,N)]$

  $= 1/[(a + (1-a)/N) + C(n,N)/(nt)]$

  $< 1/(a + (1-a)/N)$
- Note that as N®¥, then S(N)®1/a, so the speed-up is always limited to a maximum of 1/a no matter how many processors are used.

Example:



18. Write the implications of Amdahl law.

Ans:

- Amdahl's Law says that the serial fraction puts a severe constraint on the speed-up that can be achieved as the number of processors increases.
- Amdahl's Law suggests that it is not cost effective to build systems with large numbers of processors because sufficient speed-up will not be achieved.
- It turns out that most important applications that need to be parallelised contain very small serial fractions, so large machines are justified.
-

19. Explain the semantics of message sends and receives.

Ans:

Message Sending:

- Suppose one node sends a message to another node:

  send (data, count, datatype, destination)
- There are two possible behaviours:

  -Blocking send

  -Non-blocking send

Blocking Send:

- The send does not return until the data to be sent has “left” the application.
- This usually means that the message has been copied by the message passing system, or it has been delivered to the destination process.
- On return from the send() routine the *data* buffer can be reused without corrupting the message.

Non-blocking Send:

- Upon return from the send() routine the *data* buffer is volatile.
- This means that the data to be sent is not guaranteed to have left the application, and if the *data* buffer is changed the message may be corrupted. The idea here is for the send() routine to return as quickly as possible so the sending process can get on with other useful work.
- A subsequent call is used to check for completion of the send.

Message Receiving:

- Suppose one node receives a message from another node:

  receive (data, count, datatype, source)
- There are two possible behaviours:

  Blocking receive

  Non-blocking receive

==========0==========