Synchronous Computations

1 Why we need barrier in Synchronization and Explain Library call approach for a barrier.

-> Need of Barrier:

Barriers are essential in synchronization to ensure that a set of processes reach a specific point in their execution before any of them can proceed further.

Data Dependency: A Barriers ensures all processes are synchronized before data exchange occurs.

Coordinated Process: Processes need to proceed together from a consistent state to avoid any error.

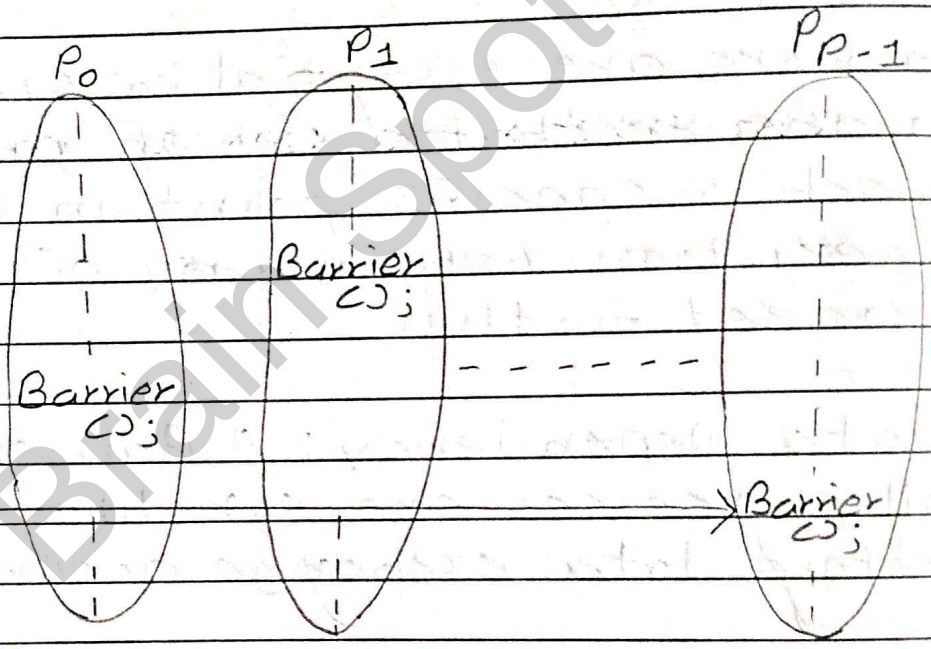
Synchronization in Parallel Computation: Barriers help maintain synchronization ensuring all processes complete one phase of communication.

Without barriers, some processes may continue execution while others are still working.

Data : / /

-> Library Call Approach for a Barrier:

The Library call approach uses predefined routines provided by libraries like MPI to implement barriers.



Each process in the group calls, Barrier() with a communicator as its parameter.

When a process reaches the barrier, it becomes blocked and waits until all other processes in the group also call the same barrier.

Only when every process in the group reaches the barrier, the call is complete, and all processes are released to proceed further.

The barrier does not require message tags for communication.

2 Explain Butterfly Barrier in detail.

=> The Butterfly Barrier is a synchronization mechanism used in parallel computing where processes coordinate with each other in multiple stage.

This method ensures that all process reach the synchronization point.

The Butterfly barrier is particularly efficient for systems with process organized in powers of two.

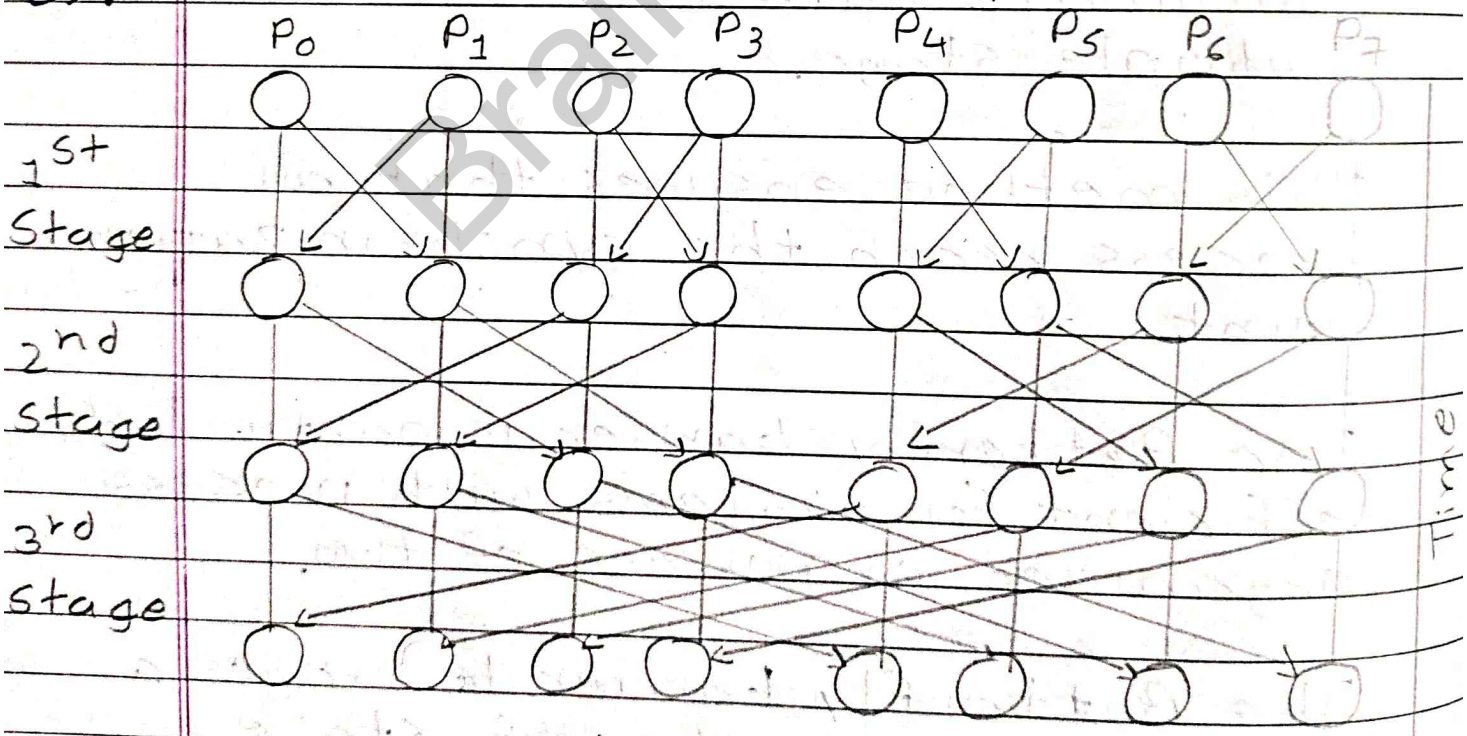
The Butterfly barrier leverages a logarithmic number of stage to achieve synchronization.

In each stage, processes pair up, exchange synchronization message and proceed to the next stage with new pairings.

This approach is designed for system where the number of process is a power of 2.

For non-power of 2 processes, modulo p arithmetic is used to determine communication partners.

Ex.



At each stage, process i synchronizes with a process that is offset by 2^{s-1} indices.

SMVS
Page No.
Date : / /

With p = 8, the synchronization progresses as followes:

- Stage 1 : $P_0 \leftrightarrow P_1$, $P_2 \leftrightarrow P_3$, $P_4 \leftrightarrow P_5$, $P_6 \leftrightarrow P_7$
- Stage 2 : $P_0 \leftrightarrow P_2$, $P_1 \leftrightarrow P_3$, $P_4 \leftrightarrow P_6$, $P_5 \leftrightarrow P_7$
- Stage 3 : $P_0 \leftrightarrow P_4$, $P_1 \leftrightarrow P_5$, & $P_2 \leftrightarrow P_6$, $P_3 \leftrightarrow P_7$

3 Briefly discuss data parallel and prefix sum operation in synchronized computations.

=> Data Parallel Computation:

Data Parallel computation involves performing the same operations simultaneously on multiple data elements.

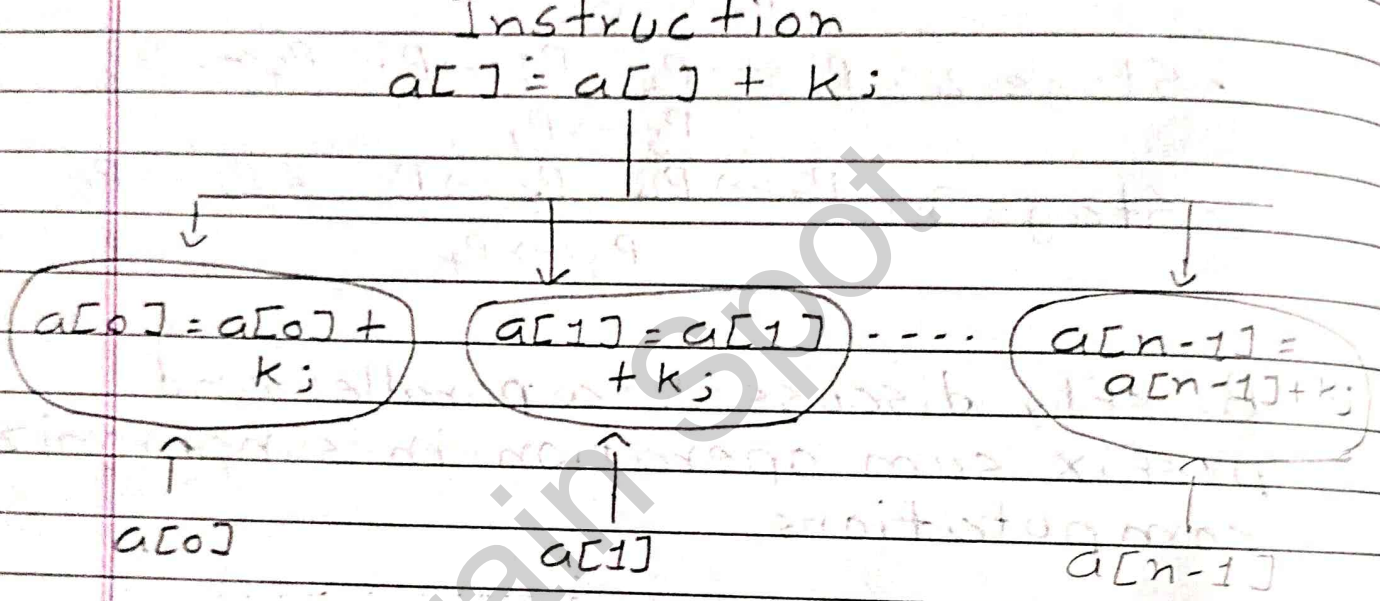
This method is efficient due to its simplicity and scalability.

The same instruction is executed by multiple processors on different data elements in parallel.

A construct like forall can be used

SMVS
Page No.
Date : / /

in parallel programming to specify operations that can run simultaneously.



Implicit synchronization ensures all operations complete before advancing.

The programmer writes one logical program, which is implicitly executed in parallel.

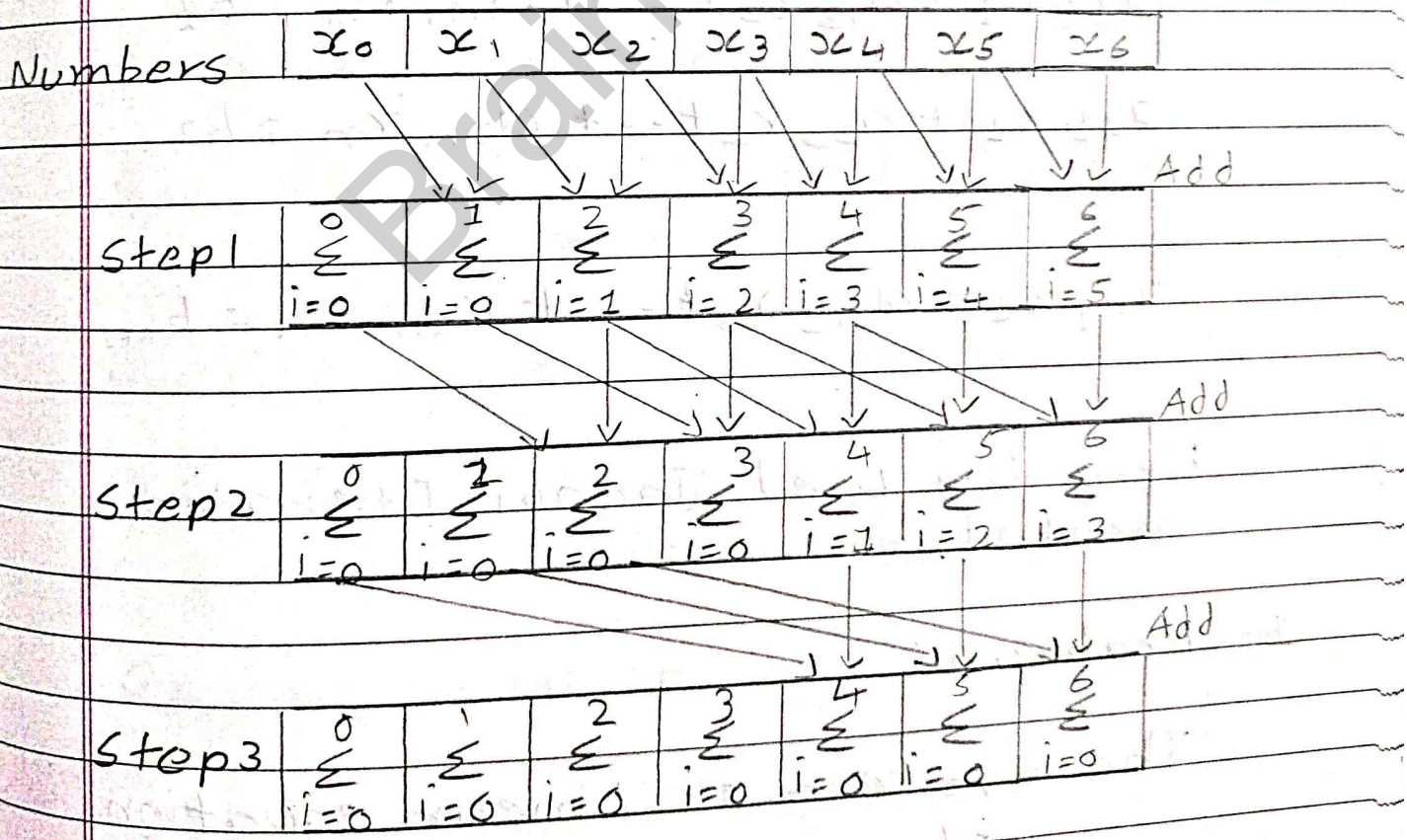
SIMD systems exemplify data parallel computation by executing a single instruction on multiple data elements simultaneously.

SMVS
Page No.
Date : / /

=> Prefix Sum Operation:

The Prefix sum Operation is a comman example of data parallel computation.

Prefix sum Operation involves computing partial sums of an array, where each element in the output array is the sum of all preceding elements in the input array.



Date : / /

Given an array x[0], x[1], ----, x[n-1], compute an output array sum such that:

sum[i] = x[0] + x[1] + --- + x[i]

4 Explain Solving a system of Linear equation by Iteration in detail.

-> Suppose, The system of Linear equation,

$$a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = b_1$$

$$a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n = b_2$$

$$\vdots$$

$$a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n = b_n$$

Here, We used Jacobi Iteration method.

1 Problem:

The system of Linear equation,

$$\sum_{j=0}^{n-1} a_{ij}x_j = b_i$$

Rearrage to express x_i,

$$x_i = \frac{1}{a_{i,i}}\left(b_i - \sum_{j=0}^{n-1} a_{ij} x_j\right)$$

2 Convergence Condition:

The method is converges, if matrix is diagonally dominant,

$$|a_{i,i}| > \sum_{\substack{j=0 \\ j \neq i}}^{n-1} |a_{i,j}|$$

3 Iteration Process:

Start with an initial guess for all x_i, $x_i^{(0)} = b_i$

Update all x_i, simultaneously,

$$x_i^{(K+1)} = \frac{1}{a_{i,i}}\left(b_i - \sum_{j=0}^{n-1} a_{i,j} x_j^{(K)}\right)$$

Repeat Until the values converge.

4 Termination Condition:

Absolute difference:

$$|x_i^{(t)} - x_i^{(t-1)}| < \text{tolerance}$$

Page No.
Date : / /

5 Explain Heat Distribution Problem in detail.

=> The heat distribution problem involves calculating the temperature at each point on a square metal plate using iterative methods.

The Temperature is a given at a point is influenced by the temperature of its neighboring points.

Aim to compute steady-state temperature distribution across the plate.

-> Sequential Algorithm :

1 Initialize the Grid.

2 Iterate for a fixed number or Until Convergence.

3 Check Convergence

4 Repeat Until Convergence.

SMVS
Page No.
Date : / /

-> Parallel Algorithm :

1 Initialize Grid

2 Divide the Grid

3 Iterate in Parallel

4 Communicate Boundary Values

5 Check Convergence

6 Repeat Until Convergence.

6 Explain Cellular Automata Problem in detail with example.

=> Cellular Automata are mathematical model used to simulate complex system through simple rules applied to cells arranged in a grid-like structure.

A Cellular Automaton is particularly suitable for problems that can be modeled by synchronous iteration.

Page No.
Date : / /

In this context, the problem space is divided into cells and the cells evolve based on certain predefined rules.

One of the most famous cellular Automata examples is the "Game of Life".

It consists of two dimensional grid, where each cell can either be alive or dead.

The evolution of the grid is governed by the following rules.

1) Survival: A live cell with 2 or 3 live neighbors stays alive.

2) Death From Overpopulation: A Live cell with 4 or more neighbors dies.

3) Death From Isolation: A Live cell with fewer than 2 neighbors dies.

4) Birth: A empty cell with exactly 3 live neighbors becomes alive.

SMVS
Page No.
Date : / /

7 Write Short note on: Partially Synchronous Methods.

=> Partially Synchronous methods are a technique used to reduce synchronization overhead in parallel computations.

In traditional synchronous iterative methods, all processes synchronize at every iteration.

In Partially Synchronous Methods allow processes to continue their computation without waiting for others.

This method enhance performance by removing unnecessary synchronization points.

Partially Synchronous methods remove the barrier of synchronization.

Processes are allowed to continue with the next iteration even before other processes complete the current one.