(DEEMED TO BE UNIVERSITY)

PARALLEL AND DISTRIBUTED COMPUTING

22CS4106R

STUDENT ID:

STUDENT NAME:

ACADEMIC YEAR: 2024-25

Table of Contents

A.Y. 2024-25 LAB CONTINUOUS EVALUATION

S.No.	Date	Experiment Name	Pre-Lab (10M)	In-Lab (25M)			Post-Lab (10M)	Viva Voce (5M)	Total (50M)	Faculty Signature
				Program/ Procedure (5M)	Data and Results (10M)	Analysis & Inference (10M)				
1.		Demonstrate various programs in OpenMP, Parallel global sum Session Parallel Matrix-Vector Multiplication with OpenMP								
2.		Parallel Computation with OpenMP: Summation Techniques and Matrix-Vector Multiplication								
3.		Parallel implementation of PI, Fundamentals of MPI Programming								
4.		Parallel Computation and Sorting Techniques Using MPI and OpenMP								
5.		Exploring Parallel Sorting Algorithms: Implementations and Analysis with MPI, OpenMP.								
6.		Advanced Parallel Computing in Mathematical Algorithms: Harnessing OpenMP, and MPI for Matrix Operations, Integration, and Sorting								
7.		Distributed Algorithms and Parallel Programming with MPI								

S.No.	Date	Experiment Name	Pre-Lab (10M)	In-Lab (25M)			Post-Lab (10M)	Viva Voce (5M)	Total (50M)	Faculty Signature
				Program/ Procedure (5M)	Data and Results (10M)	Analysis & Inference (10M)				
8.		Distributed Mutual Exclusion and Parallel Programming								
9.		Leader Election in Distributed Systems: Implementing Bully and Ring Algorithms Using MPI/C Library								
10.		Exploring Parallel Computing: Data Structures, Algorithmic Patterns, and VisualizationTechniques								
11.		Simulation and Distributed Programming: Consensus and Matching Algorithms								
12.		Distributed Systems and Cloud Computing: Web Services, MPI, and HPC Workflows								

Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	<TO BE FILLED BY STUDENT>

Experiment 1

Title: Demonstrate various programs in OpenMP, Parallel global sum

Aim/Objective:

Demonstrate the versatility of OpenMP through multiple programs implementing parallel global sum algorithms, emphasizing diverse parallelization approaches for optimal performance.

Description:

Explore OpenMP's capabilities by presenting distinct programs showcasing parallel global sum computations, utilizing different OpenMP directives and strategies to achieve efficient parallelization.

Pre-Requisites:

Participants should have a solid understanding of C programming, familiarity with parallel computing concepts, and access to a compiler with OpenMP support. A basic knowledge of multithreading and a multiprocessor system for parallel execution are also recommended.

Pre-Lab:

1. Explain the purpose of the critical directive in OpenMP and provide a scenario where it is beneficial?

2. Describe the role of the master directive in OpenMP. In what situations is it commonly used?

3. Discuss the significance of the default clause in OpenMP. How does it affect the sharing of variables in a parallel region?

4. What is the purpose of the threadprivate directive in OpenMP? Provide an example of a situation where using this directive is advantageous.

5. Explain the concept of the reduction clause in OpenMP. How does it facilitate parallel computation?

In-Lab:

1. Basic programs to demonstrate various #pragrams in OpenMP: Parallel • for • private • shared • critical • MASTER Directive • THREADPRIVATE Directive • DEFAULT Clause • FIRSTPRIVATE Clause •LASTPRIVATE Clause.
   - /* Program to Demonstrate #pragrams in OpenMP: Parallel, for */
   - /* Program to Demonstrate private clause */
   - /* Program to Demonstrate shared clause */
   - /* Program to Demonstrate Critical Directive */
   - /* Program to Demonstrate Master Directive */
   - /* Program to Demonstrate threadprivate Clause */
   - /* Program to Demonstrate default Clause */
   - /* Program to Demonstrate firstprivate clause */
   - /* Program to Demonstrate lastprivate clause */

- **Program:**

2. Write programs to demonstrate SINGLE Directive BARRIER Directive ATOMIC Directive
   /* Program to Demonstrate single Directive */
   /* Program to demonstrate barrier Directive */
   /* Program to Demonstrate atomic Directive */

- **Program:**

Sample VIVA-VOCE Questions (In-Lab):

1. What does the SINGLE directive in OpenMP accomplish, and in what scenarios would you use it?
2. Explain the purpose of the BARRIER directive in OpenMP and why it is essential in parallel programming.
3. What is the role of the ATOMIC directive in OpenMP? Provide an example where using ATOMIC is crucial?
4. How does the BARRIER directive contribute to ensuring correct synchronization in parallel programs?
5. In what situations would you prefer using the SINGLE directive over the MASTER directive, and vice versa?

Post-Lab:

1. Implement a program to demonstrate Parallel global sum using OpenMP Reduction clause

- **Program:**

Evaluator Remark (if Any):	**Marks Secured:_____out of 50**
	Signature of the Evaluator with Date

Experiment 2

Title: Parallel Computation with OpenMP: Summation Techniques and Matrix-Vector Multiplication

Aim/Objective:

Illustrate parallel computation techniques using OpenMP for two fundamental algorithms—summation and matrix-vector multiplication—aiming to enhance performance through concurrent processing.

Description:

Implement OpenMP-based programs for parallel summation and matrix-vector multiplication, exploring various OpenMP directives and strategies to optimize computation speed and scalability.

Pre-Requisites:

Proficiency in a language supporting OpenMP (e.g., C/C++), understanding of OpenMP directives, familiarity with array and matrix manipulation, and a foundational grasp of parallel programming concepts including thread management, synchronization, and parallelization strategies.

Pre-Lab:

1. What is the primary objective of utilizing OpenMP in parallel summation techniques?

2. Briefly discuss one advantage of using OpenMP for parallel summation techniques compared to sequential approaches.

3 In the context of matrix-vector multiplication, define the term "data parallelism" and explain how OpenMP leverages this concept for parallel computation.

4. Explain the role of load balancing in the context of parallel matrix-vector multiplication using OpenMP, and why it is crucial for optimizing performance.

5. Name one key OpenMP directive used in the parallelization of computations. Provide a brief description of its purpose?

In-Lab:

1. Implement Parallel Summation using OMP - The Array Element Sum Problem, Tree structure global sum - Parallel-tree-sum.c

- **Program:**

2.Implement a program to demonstrate Parallel prefix sum using OpenMP - OMP Parallel prefix sumfinal.c

- **Program**:

Sample VIVA-VOCE Questions (In-Lab):

1. Explain the concept of parallel summation using OpenMP in the context of the array element sum problem
2. How does the #pragma omp parallel directive contribute to the parallelization of the array element sum problem?
3. Why is a critical section (#pragma omp critical) used in the parallel tree-sum implementation, and what would happen if it were omitted?
4. How can the performance of the parallel tree-sum algorithm be influenced by the number of threads used in the computation?
5. Explain the purpose of the reduction clause in OpenMP and discuss whether it could be applied to improve the parallel tree-sum implementation.

Post-Lab:

1. Implement Parallel matrix vector multiplication using OpenMP

- **Program:**

Evaluator Remark (if Any):	**Marks Secured:_____out of 50**
	Signature of the Evaluator with Date

Experiment 3

Title: Parallel implementation of PI, Fundamentals of MPI Programming

Aim/Objective:

To demonstrate the parallel computation of PI using the Monte Carlo method with OpenMP for improved performance and to showcase the MPI_Bcast function for broadcasting data from one process to all other processes in a communicator.

Description:

In this experiment, students will explore the program generates random points in a unit square and estimates PI by calculating the ratio of points within a quarter circle to the total points. OpenMP is utilized to parallelize the computation for efficient use of multiple threads

The program initializes a message on the root process, and then MPI_Bcast is used to broadcast this message to all other processes in the communicator. This demonstrates the concept of one-to-all communication.

Pre-Requisites:

- Basic understanding of the Monte Carlo method for PI estimation.
- Familiarity with OpenMP directives and parallel programming concepts.
- Understanding communication patterns in parallel computing.

Pre-Lab:

1. How does the MPI program partition the sequence of numbers in the example provided for computing π?

2. Discuss the challenges associated with load balancing when partitioning a sequence of numbers for parallel computation?

3. Explain the differences between the MPI_Send and MPI_Recv functions in MPI. How are these functions employed for point-to-point communication in the context of parallelizing the computation of π?

4. Discuss the trade-offs between parallel efficiency and communication overhead in the MPI implementation of the π computation?

5. How does the program attempt to address load balancing, and what factors can impact the efficiency of the load balancing strategy in the provided MPI example?

- **Program:**

In-Lab:

1. Implement Basic programs to demonstrate Broad cast and Reduction using mpi-reduce.c , mpi-bcast.c
   - /* Program to demonstrate MPI_Reduce for sum reduction */
   - /* Program to demonstrate MPI_BCAST for sum reduction */

- **Program:**

2. Write programs to Partition a sequence ofnumbers into parts and adding them using

a. Point-to-Point communications
b. MPI_Bcast
c. MPI_Scatter

Sample VIVA-VOCE Questions (In-Lab):

1. How does the Monte Carlo method work for estimating the value of PI in the provided OpenMP program?
2. Explain the significance of the #pragma omp parallel for directive in the program.
3. What is the purpose of the reduction(+:count) clause in the parallel for loop?
4. How does the number of OpenMP threads affect the performance and accuracy of the PI estimation?
5. Can you explain a scenario where the Monte Carlo method for PI estimation and parallelization with OpenMP might be particularly advantageous?

Post-Lab:

1. Explain MPI_Reduce, MPI_Send, MPI_Sendrecv, MPI_Barrier, MPI_Scatter, MPI_Gather, MPI_Allgather,MPI_Allreduce

Ans:

Evaluator Remark (if Any):	**Marks Secured:_____out of 50**
	Signature of the Evaluator with Date

Experiment 4

Title: Parallel Computation and Sorting Techniques Using MPI and OpenMP

Aim/Objective:

Implement parallel computation techniques using MPI (Message Passing Interface) and OpenMP. It focuses on distributed data processing, such as calculating averages and finding maxima using MPI, and parallel sorting algorithms like odd-even transposition sort using OpenMP.

Description:

This lab introduces the concepts of distributed data computation and parallel sorting using MPI and OpenMP. In the Pre-Lab, MPI programs are developed to compute averages and maxima by distributing and collecting data using collective communication functions like MPI_Scatter, MPI_Gather, and MPI_Reduce. The In-Lab task involves implementing an odd-even transposition sort, a parallel sorting algorithm, using OpenMP.

Pre-Requisites:

- Understanding of distributed systems and parallel computing concepts.
- Familiarity with MPI and OpenMP libraries.
- Basic knowledge of C/C++ programming.
- Tools: MPI (e.g., MPICH or OpenMPI) and GCC with OpenMP support.

Pre-Lab:

1. Develop an MPI program to compute the average numbers using the following functions and compute the time required for the same.

- MPI_Scatter and MPI_Gather.
- MPI_Scatter and MPI_Reduce.

- **Program:**

2. Write an MPI program to find the maximum number of each group, then master finds maximum of results.

1. Program:

In-Lab:

1. Program to implement odd-even transposition sort using OpenMP

- **Program:**

VIVA-VOCE Questions (In-Lab):

1. What is the difference between MPI Gather and MPI_Reduce?
2. How does MPI_Scatter distribute data among processes?
3. Why is odd-even transposition sort suitable for parallel computation?
4. Explain the difference between shared memory (OpenMP) and distributed memory (MPI) models.
5. What is the purpose of the schedule clause in OpenMP?

Post-Lab:

1. Write an MPI program to perform matrix-vector multiplication using `MPI_Scatter` and `MPI_Gather`.

- **Program:**

2. Implement a parallel bubble sort using OpenMP and compare its performance with odd-even transposition sort.

- **Program:**

Evaluator Remark (if Any):	**Marks Secured:_____out of 50**
	Signature of the Evaluator with Date

Experiment 5

Title: Exploring Parallel Sorting Algorithms: Implementations and Analysis with MPI, OpenMP

Aim/Objective:

To Investigate parallel sorting algorithms through implementations and analysis using MPI, OpenMP, aiming to compare and optimize their performance on various architectures.

Description:

Develop sorting algorithms in parallel using MPI for distributed memory systems, OpenMP for shared-memory multi-core architectures. Conduct a comprehensive analysis to understand the strengths and weaknesses of each parallelization approach.

Pre-Requisites:

- Understanding of sorting algorithms such as merge sort, quicksort, Shell sort, and bitonic sort.
- Familiarity with OpenMP directives for multi-core CPU parallelism.

Pre-Lab:

1. Write a Program to implement Parallel Odd Even sort - MPI

- **Program:**

2.Write a Program to implement Merge sort using MPI

- **Program:**

3. Write a Program to implement Bitonic sort using OpenMP

- **Program:**

In-Lab:

1. A communicator has 3 processes (ID: 0, 1 and 2) such that process 0 sends a binary value 001 to process 1, in-turn process 1 increment the value by 1 (i.e. 010) and send the same to process 2, in-turn process 2 increment the value by 1 (i.e. 011) and send the same to process 0. Continue the process until the binary value becomes 111. Develop a MPI program to implement the above scenario (Rotating Binary Addition).

- **Program:**

Sample VIVA-VOCE Questions (In-Lab):

1. How does the parallel odd-even sort algorithm work in the MPI implementation?
2. How is the array distributed among MPI processes, and why is it important for efficiency?
3. Explain how point-to-point communication is utilized in the odd-even sort algorithm for parallelism.
4. What is the significance of the odd and even phases in the parallel odd-even sort?
5. How does the parallel odd-even sort handle the synchronization of processes after each phase?

Post-Lab:

1. Write a Program to implement Merge sort, Sample Sort using OpenMP

- **Program:**

2. Analyze odd-Even Merge sort and Hyper cube quick sort algorithm with an example

- **Program:**

Evaluator Remark (if Any):	**Marks Secured:_____out of 50**
	Signature of the Evaluator with Date

Experiment 6

Title: Advanced Parallel Computing in Mathematical Algorithms: Harnessing OpenMP, and MPI for Matrix Operations, Integration, and Sorting

Aim/Objective:

To explore the benefits of parallelizing numerical algorithms such as Gaussian Elimination, Trapezoidal Integration, and Row Sums, focusing on enhancing computational efficiency through parallel decomposition, load balancing, inter-process communication, and scalability considerations.

Description:

Parallel Gaussian Elimination divides the matrix manipulation tasks among multiple processing units. Parallel decomposition techniques distribute the workload, improving overall performance by exploiting parallelism in matrix operations.

Pre-Requisites:

- Understanding of numerical algorithms, particularly Gaussian Elimination, Trapezoidal Integration, and Row Sums.
- Familiarity with parallel decomposition techniques, load balancing strategies, and inter-process communication mechanisms.
- Knowledge of scalability factors, considering problem size and communication efficiency.

Pre-Lab:

1. How does parallelizing Gaussian Elimination enhance computational efficiency, and what role do parallel decomposition techniques play in this context?

2. Briefly explain the concept of load balancing in the context of parallel Trapezoidal Integration and identify one challenge associated with achieving optimal load distribution.

3. In a parallel environment, describe a key consideration for efficiently distributing the computation of row sums among multiple processing units.

4. How does inter-process communication contribute to the success of a parallel Gaussian Elimination implementation, and what is its impact on overall performance?

5. Defining scalability in the context of parallel computing and identify two factors that can affect the scalability of algorithms like Gaussian Elimination, Trapezoidal Integration,and Row Sums.

Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	<TO BE FILLED BY STUDENT>

In-Lab:

1. Write a program to perform Parallel Gaussian Elimination Using OpenMP

- **Program:**

2.Write a program to perform Trapezoidal Integration Using OpenMP

- **Program:**

3.Implementation of a distributed algorithm in MPI to find the Minimum Spanning Tree of an undirected graph.

- **Program:**

Sample VIVA-VOCE Questions (In-Lab):

1. What is OpenMP, and why is it used in parallel programming?
2. What is Trapezoidal Integration, and in what scenarios is it used?
3. How does OpenMP divide the workload for computing integration among threads?
4. Why is it essential to use synchronization mechanisms in parallel Gaussian Elimination?
5. Explain the role of pivoting in Gaussian Elimination. How is it handled in a parallel implementation?

Post-Lab:

1.Write a program to implement a parallel version of Trapezoidal rule Using MPI.

- **Program:**

2. Given a directed graph, write a MPI program to count all possible walks (both closed and open) of length k from every source and destination pair using DFS.

- **Program:**

Evaluator Remark (if Any):	**Marks Secured:_____out of 50**
	Signature of the Evaluator with Date

Experiment 7

Title: Distributed Algorithms and Parallel Programming with MPI

Aim/Objective:

Implement the fundamental concepts of the Message Passing Interface (MPI), a standard for parallel programming in distributed systems. It focuses on performing distributed computation tasks using MPI, such as synchronization, communication, and computation among multiple processes.

Description:

The lab explores the use of MPI to enable parallel computation and communication in distributed systems. Key MPI functions such as MPI_Barrier() and MPI_Wtime() are introduced, along with practical applications like tree-structured global sum, vector addition, and topology-based communication. Distributed algorithms for clock synchronization (Lamport's clock and Vector clock) are simulated to reinforce concepts of time and order in distributed systems. Additionally, advanced concepts such as non-blocking communication and the Monte Carlo method for numerical computation are covered.

Pre-Requisites:

- Basic understanding of distributed systems.
- Familiarity with C/C++ programming.
- Knowledge of parallel processing concepts.
- MPI library installation and environment setup.

Pre-Lab:

1. Demonstrate the use of MPI_Barrier() and MPI_Wtime() with a program

- **Program:**

In-Lab:

1. Lamports clock Algorithm Implementation in C – simulation.

- **Program:**

2. Vector clock Algorithm Implementation in C - simulation

- **Program:**

VIVA-VOCE Questions (In-Lab):

a. What is the purpose of `MPI_Barrier()`?
b. How does `MPI_Wtime()` differ from other time measurement functions?
c. Explain the significance of tree-structured communication in distributed systems.
d. How do Lamport's clocks differ from Vector clocks?
e. What is the advantage of non-blocking communication in MPI?

Post-Lab:

1.Write an MPI Program that implements Ring Topology using Non-Blocking Send/Receive

- **Program:**

2. Calculate PI using Monte Carlo method and MPI

- **Program:**

Evaluator Remark (if Any):	**Marks Secured:_____out of 50**
	Signature of the Evaluator with Date

Experiment 8

Title: Distributed Mutual Exclusion and Parallel Programming

Aim/Objective:

To implement various distributed mutual exclusion algorithms and parallel programming techniques for efficient resource sharing and computation.

Description:

This lab explores key distributed algorithms for mutual exclusion and parallel computation. These techniques ensure fair resource allocation and optimized execution in distributed and parallel computing environments.

Pre-Requisites:

Familiarity with mutual exclusion algorithms, such as those used to manage access to shared resources in a distributed environment, is also required. Proficiency in C programming is necessary, along with knowledge of parallel programming libraries like OpenMP for shared-memory systems and MPI for distributed-memory systems. Additionally, the lab requires a properly configured environment with OpenMP and MPI support, such as GCC with OpenMP enabled and MPI libraries like MPICH or OpenMPI, to execute and test the programs effectively.

Pre-Lab:

1. What is mutual exclusion, and why is it required in distributed systems?

2. How does the Birman-Schiper-Stephenson protocol ensure causal ordering of events?

3. Explain how parallel computing improves computational performance.

4. Describe the working of OpenMP and MPI in parallel programming.

In-Lab:

1. C program to implement Birman-Schiper-Stephenson protocol - simulation

- **Program:**

2. Ricart Agrawala distributed mutual exclusion algorithm

- **Program:**

3. Implement token ring based mutual exclusion algorithm.

- **Program:**

VIVA-VOCE Questions (In-Lab):

1. What is the time complexity of a conventional serial algorithm for multiplying two n x n matrices?
2. Can you explain the step-by-step process of multiplying two matrices using the conventional serial approach?
3. Compare and contrast row-wise 1D partitioning and 2D partitioning for matrix-vector multiplication in terms of parallel efficiency and communication overhead.
4. How does the choice of partitioning strategy impact the parallel runtime of matrix-vector multiplication?
5. Describe the Cannons algorithm for matrix-matrix multiplication on 16 processes. What advantages does this algorithm offer in terms of parallelization?

Post-Lab:

1. Implement a Program to implement Matrix-Matrix multiplication using OpenMP

- **Program:**

2. Program to implement Matrix-Matrix multiplication using MPI

- **Program:**

Evaluator Remark (if Any):	**Marks Secured:_____out of 50**
	Signature of the Evaluator with Date

Experiment 9

Title: Leader Election in Distributed Systems: Implementing Bully and Ring Algorithms Using MPI/C Library

Aim/Objective:

To explore distributed computation essentials through the implementation of Bully and Ring algorithms using MPI and C library, focusing on demonstrating inter-process communication and fault-tolerant leader selection.

Description:

Develop and analyze parallel applications using MPI for distributed computing. Emphasize matrix operations, dot products, and numerical calculations, showcasing the versatility of parallel computation in solving computationally intensive tasks.

Pre-Requisites:

Proficiency in C/C++ programming, understanding of parallel programming concepts, familiarity with MPI programming models, and knowledge of matrix operations,dot products, and numerical algorithms.

Pre-Lab:

1. How can parallel dot product be implemented using multi-threading or parallel processing?

2. How does parallel processing or GPU acceleration impact the performance of matrix transposition operations?

3.Write the algorithm of prefix sum and define the use of prefix sum reduce time complexity in dynamic programming algorithms?

4. Write pseudocode for an in-place matrix transposition function, assuming the original matrix is stored in row-major order in memory. Analyse its time and space complexity.

5. What are the advantages of using parallel computation for the dot product in terms of performance?

In-Lab:

1. Write a program to implement Bully algorithm using MPI

- **Program:**

2. Write a program to implement ring algorithm using MPI

- **Program:**

VIVA-VOCE Questions (In-Lab):

1. How does the concept of parallel prefix sum differ from the sequential approach, and why is it beneficial in parallel computing?

2.Explain the role of MPI collective communications in the parallel prefix sum program. Which specific MPI function is commonly used for such operations, and why?

3. Describe the process of matrix transposition and its significance in parallel computing. How does MPI facilitate the parallelization of this operation across multiple processes?

4.What considerations should be taken into account when implementing matrix transposition using MPI, especially in terms of data distribution and communication patterns?

5. What is the collective communication model in MPI, and how does it contribute to the implementation of the dot product program?

Post-Lab:

1. Implement a Fault-Tolerant Ring Algorithm with Failure Detection

- **Program:**

2. Modify and Analyze the Bully Algorithm for Dynamic Node Changes.

- **Program:**

Evaluator Remark (if Any):	**Marks Secured:_____out of 50**
	Signature of the Evaluator with Date

Experiment 10

Title: Exploring Parallel Computing: Data Structures, Algorithmic Patterns, and Visualization Techniques

Aim/Objective:

Explore parallel computing by investigating data structures, algorithmic patterns, and visualization techniques, aiming to enhance understanding and proficiency in leveraging parallelism for computational tasks.

Description:

Develop and analyze parallel algorithms, employ advanced data structures, and utilize visualization techniques to understand and optimize the performance of parallel computations.

Pre-Requisites:

Proficiency in a programming language (e.g., C/C++), foundational knowledge of data structures and algorithms, and a basic understanding of parallel computing concepts.

Pre-Lab:

1. What is a parallel prefix sum in the context of linked lists, and how does it differ from the array-based approach?

2. Explain the significance of list ranking in parallel computing and its relevance to linked lists.

3. Briefly describe the Mandelbrot Set and how parallel computing accelerates its rendering process.

4. Compare shared memory and distributed memory architectures in parallel computing, highlighting their key differences.

5. How would you assess the performance of a parallel algorithm for linked lists? Can you name one optimization technique for enhancing its efficiency?

In-Lab:

1. Write a program to demonstrate Linked list Parallel prefix sum using MPI

- **Program:**

2. Write a program to implement concurrent linked list

- **Program:**

3.Parallel Program to render the Mandelbrot Set

- **Program:**

VIVA-VOCE Questions (In-Lab):

1. Explain the concept of parallel prefix sum applied to a linked list. How does MPI facilitate communication and coordination in the implementation of this algorithm?
2. Discuss the advantages and challenges of parallelizing prefix sum on a linked list data structure. How does the linked list structure impact the parallelization approach compared to arrays?
3. What is the significance of implementing a concurrent linked list? Describe the key challenges and solutions involved in ensuring concurrent access and modification of a linked list.
4. Explain the potential concurrency issues that may arise in a linked list and how the concurrent linked list program addresses them. What synchronization mechanisms are employed?
5. Walk through the steps involved in rendering the Mandelbrot Set in parallel. How does parallelization enhance the efficiency of generating the Mandelbrot Set?

Post-Lab:

1. Parallel Program to implement List Ranking

- **Program:**

2. Parallel Program to render the Mandelbrot Set.

- **Program:**

Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	<TO BE FILLED BY STUDENT>

3. OpenMP and MPI implementation of Travelling Salesman Problem using Branch and Bound algorithm.

- **Program:**

Evaluator Remark (if Any):	**Marks Secured:_____out of 50**
	Signature of the Evaluator with Date

Experiment 11

Title: Simulation and Distributed Programming: Consensus and Matching Algorithms

Aim/Objective:

To simulate the Lamport-Shostak-Pease Algorithm to achieve consensus in distributed systems despite the presence of faulty nodes and to implement the Stable Marriage Problem using MPI, utilizing asynchronous message passing for inter-process communication and logging events such as pairings and rejections to a trace file.

Description:

The implementation of the Stable Marriage Problem using MPI demonstrates distributed computing techniques, including asynchronous message passing for efficient inter-process communication and event logging for monitoring and debugging. achieving consensus and solving combinatorial problems using distributed programming. The Lamport-Shostak-Pease Algorithm simulation addresses the challenge of reaching agreement in the presence of faulty nodes, showcasing fault tolerance in distributed systems

Pre-Requisites:

Understanding of distributed systems concepts, particularly fault tolerance and consensus mechanisms. Familiarity with the Lamport-Shostak-Pease Algorithm is essential, as it forms the basis for achieving consensus in the presence of faulty nodes. Additionally, a foundational knowledge of combinatorial optimization problems, such as the Stable Marriage Problem, is necessary. Proficiency in C programming and experience with the Message Passing Interface (MPI) are required to implement distributed solutions effectively.

Pre-Lab:

1. What is the significance of the Lamport-Shostak-Pease Algorithm in distributed systems?

2. Explain the Byzantine Generals Problem and its relevance to achieving consensus computation?

3. What is the Stable Marriage Problem, and how can it be solved using distributed programming?

4. How does the asynchronous message passing differ from synchronous communication in MPI?

In-Lab:

1. Implement Lamport-Shostak-Pease Algorithm in C- simulation

- **Program:**

2. MPI Distributed program to solve the **Stable Marriage Problem**. The processes should communicate using asynchronous message passing. Log the trace of events (pairing, breaking), as they happen, in a text file named as Log.txt.

- **Program:**

VIVA-VOCE Questions (In-Lab):

1. How does the Lamport-Shostak-Pease Algorithm handle faulty nodes in a distributed system?
2. What are the challenges of implementing the Stable Marriage Problem in a distributed environment?
3. Explain the advantages of using asynchronous communication in MPI programs.
4. How can logging improve debugging and analysis in distributed systems?
5. What modifications would you make to the Lamport-Shostak-Pease Algorithm for non-Byzantine faults?

Post-Lab:

1. Modify the table Marriage MPI Program and include the dynamic addition.

- **Program:**

- In the above task Log these changes to Log.txt
- **Program:**

Evaluator Remark (if Any):	**Marks Secured:_____out of 50**
	Signature of the Evaluator with Date

Experiment 12

Title: Distributed Systems and Cloud Computing: Web Services, MPI, and HPC Workflows

Aim/Objective:

To explore and implement distributed systems using web services, Message Passing Interface (MPI), and High-Performance Computing (HPC) workflows by creating, consuming, and running distributed applications and jobs on multi-node and single-node systems in HPC and cloud environments.

Description:

It emphasizes hands-on implementation, effective resource management, and performance analysis, providing a comprehensive understanding of real-world computing environments. Developing and deploying RESTful web services, running distributed MPI applications on high-performance computing (HPC) systems with both multi-node and single-node configurations, and configuring and testing cloud-based MPI clusters on AWS

Pre-Requisites:

A basic understanding of distributed systems and web services, including REST and SOAP, is essential for this lab. Familiarity with MPI programming and parallel computing concepts is also required to successfully implement the tasks. Additionally, knowledge of cloud platforms such as AWS and job schedulers like Slurm is necessary. Hands-on experience with Linux/Unix commands and shell scripting will be beneficial, along with pre-configured access to KLU HPC and AWS accounts to facilitate the practical exercises.

Pre-Lab:

1. What are web services, and how do they facilitate communication in distributed systems?

2. What is the difference between multi-node and single-node jobs?

3. Describe the importance of workload managers like Slurm in HPC

4. How does AWS Parallel Cluster simplify running HPC workloads on the cloud?

In-Lab:

1. Create a simple web service and write distributed application(calculator) to consume the Web Service

2. Run a multi-node MPI job with Slurm in KLU HPC

- **Program:**

3. Run a single node job in AWS PCS

- **Program:**

VIVA-VOCE Questions (In-Lab):

1. What are the key differences between REST and SOAP web services?
2. Explain the process of submitting and monitoring jobs in Slurm.
3. What is the significance of using AWS PCS for single-node jobs?
4. What factors affect the performance of multi-node jobs in HPC environments?

Post-Lab:

1. Run a multi-node MPI job with Slurm in AWS PCS

- **Program:**

2. Launching an Amazon EC2 MPI Cluster

- **Program:**

Evaluator Remark (if Any):	**Marks Secured:_____out of 50**
	Signature of the Evaluator with Date