WEEK NO.	PROBLEMS WITH DESCRIPTION.	
1	1	Write a C program to check if the entered number is a palindrome or not.
	2	Write a C program to find the roots of a quadratic equation.
	3	Write a C program to determine if the entered year is a leap year or not.
	4	Write a C program to calculate the compound and simple interest for a given amount.
	5	Write a C program to check whether a character is a vowel or consonant.
	6	Write a C program to print a list of prime numbers between given ranges.
	7	Write a C program to find the LCM and HCF of two entered numbers.
	8	Write an interactive program in C to find the largest and smallest number from a given array of size N.
	9	Write an interactive program in C to insert a value at a particular location in an array of size N.
	10	Write an interactive program in C to search for a particular value in an array of size N.
2.	1	Write an interactive program in C to convert a decimal number to binary, octal, and hexadecimal systems.
	2	Write an interactive program in C to merge two arrays of size N1 and N2 into an array of size N1+N2.
	3	Write an OpenMP C program to print "Hello World".
	4	Write an OpenMP C program to create multiple threads and print "Hello World".
	5	Write a C program to check if the given 20 integers are even or odd.
	6	Write an OpenMP C program to check if the given 20 integers are even or odd.
	7	Write a C program to check if the given 20 integers are Armstrong numbers or not.
	8	Write an OpenMP C program to check if the given 20 integers are Armstrong numbers or not.
3.	1	Write a C program to check if the given 20 integers are perfect numbers or not.
	2	Write an OpenMP C program to check if the given 20 integers are perfect numbers or not.
	3	Write an OpenMP C program to print "Hello World" 30 times.
	4	Write an OpenMP C program to print "Hello World" 100 times and distribute the work equally among threads.
	5	Write an OpenMP C program to create multiple threads to print "Hello World" and distribute the work randomly among threads.
	6	Write a C program to compute the value of pi.

	7	Write an OpenMP C program to compute the value of pi.
	8	Write an interactive program in C to find the factorial of a given integer.
	9	Write an interactive OpenMP C program to compute the factorial of a given integer.
4.	1	Use OpenMP to parallelize the calculation of Fibonacci numbers. Display the first n Fibonacci numbers. First 10 Fibonacci numbers: 0 1 1 2 3 5 8 13 21 34
	2	Write a C program to calculate the factorial of a number using OpenMP. Assume the number is 10. 10! = 3628800
	3	Print numbers from 1 to 10 using OpenMP.
	4	Write a C program that finds the maximum value in an array of 1000 random integers using OpenMP. Print the maximum value.
	5	Calculate the sum of an array of integers using OpenMP. Instead of using a single multiplication loop, divide the task among multiple threads to speed up the calculation.
	6	Compute the square of each element in an array using OpenMP. Instead of using a single multiplication loop, divide the task among multiple threads to speed up the calculation.
	7	Subtract two vectors (arrays) element-wise and store the result in a third vector using OpenMP.
5.	1	Calculate the sum of an array in a large array of integers using OpenMP. Instead of using a single multiplication loop, divide the task among multiple threads to speed up the calculation. Use proper constructs and clauses to avoid data races and similar conditions • Run with different numbers of threads (e.g., 2, 4, 8). • Measure execution time and calculate speedup and efficiency. • Compare results with the sequential version to observe performance gains.
	2	Calculate the average value of elements in a large array of integers using multiple threads. Divide the task among threads to speed up the computation and use OpenMP constructs to ensure that the computation is done correctly without data races. • Run with different numbers of threads (e.g., 2, 4, 8). • Measure execution time and calculate speedup and efficiency. • Compare results with the sequential version to observe performance gains.
	3	Initialize a large array of integers with consecutive values from 1 to n using multiple threads. Use OpenMP constructs to: • Ensure that each thread starts with a specific value (using firstprivate). • Capture the last initialized value by each thread (using lastprivate). • Avoid implicit barriers after the parallel region (using nowait).
	4	Double the values in a large array of integers using multiple threads. Each thread should work on a separate portion of the array. Use OpenMP to:

		• Initialize each thread’s starting index and chunk size (using firstprivate). • Capture the last processed index for each thread (using lastprivate). • Avoid implicit barriers at the end of the parallel region (using nowait).

6.	1	Compute the sum of all elements in a large array using OpenMP's, use
		proper constructs and clauses to avoid data races and similar conditions of
		the following given serial code.
		Snippet of the Sequential solution is given:
		`int n = 1000000;`
		`double array[n];`
		`double sum = 0.0;`
		`// Initialize the array with values`
		`for (int i = 0; i < n; i++) {`
		`    sum += array[i]; }`
		Analysis: To do
		• Test with different sizes of the array.
		• Run the program using different numbers of threads (e.g., 2, 4, 8).
		• Measure execution time, calculate speedup, and efficiency.
		• Understand how the reduction clause works and its importance in parallel programming.

	2	Implement a matrix multiplication algorithm using OpenMP to improve performance. a. Create a sequential matrix multiplication function: Implement a function that takes two matrices as input and returns their product. Use nested loops to iterate through the elements of the matrices. b. Parallelize the matrix multiplication using OpenMP: Use the omp parallel for directive to parallelize the outer loop of the matrix multiplication. Consider using proper clauses to correctly accumulate the product elements or some other clauses. c. Analysis: To do • Test with different array sizes (e.g., 10,000; 100,000; 1,000,000). • Run with different numbers of threads (e.g., 2, 4, 8). • Measure execution time and calculate speedup and efficiency. • Compare results with the sequential version to observe performance gains.
	3	Create a program that simulates a mixed workload, where part of the work can be parallelized, and part cannot. Specifically, consider a scenario where 70% of the workload is parallelizable, and 30% is inherently sequential. Use OpenMP

		to parallelize the parallelizable part and study how actual speedup compares with predictions made by Amdahl's Law. Consider using proper clauses to correctly accumulate the results.
	4	You have a large array of integers, and you need to compute the total count of occurrences of a specific value (e.g., 7) in the array. To ensure that multiple threads do not update the count variable simultaneously, use OpenMP to parallelize the counting process while employing critical sections to avoid race conditions. Write the Sequential and parallel implementation: Analysis: To do • Test with different array sizes (e.g., 10,000; 100,000; 1,000,000). • Run with different numbers of threads (e.g., 2, 4, 8). • Measure execution time and calculate speedup and efficiency. Compare results with the sequential version to observe performance gains.
7.	1	Count the frequency of each integer in a large array. Use OpenMP to parallelize the counting process. Ensure that updates to a shared frequency array are done safely using the atomic construct to avoid race conditions. Analysis: To do • Test with different array sizes (e.g., 10,000; 100,000; 1,000,000). • Run with different numbers of threads (e.g., 2, 4, 8). • Measure execution time and calculate speedup and efficiency. • Compare results with the sequential version to observe performance gains.
	2	Implement an MPI program where: • Process 0 sends a greeting message to Process 1. • Process 1 receives the greeting message, modifies it, and sends it back to Process 0. • Process 0 receives the modified greeting message and prints it.
	3	Distribute an array among processes and have each process compute the sum of its portion. Use MPI to collect these sums and compute the total sum.
8.	1	Distribute parts of an array to all processes using MPI_Scatter, and then gather the processed results back to the root process using MPI_Gather.
	2	Write an MPI program to compute the average of a large dataset. Divide the dataset among processes, each process computes the local sum and count, and then combine these results to compute the global average.
	3	Write an MPI program to perform Gaussian elimination on a given system of linear equations. Distribute the rows of the matrix among the processes and perform the elimination steps in parallel. Each process will handle a portion of the matrix

		and communicate with other processes to synchronize the elimination steps.
9.	1	Implement a parallel version of the Gauss-Seidel method for solving a system of linear equations. This involves distributing the matrix and vector among processes, performing iterative updates in parallel, and ensuring synchronization among processes.
	2	Compare the performance of different methods for solving linear equations.
	3	Parallel Bitonic Sort : Implement and analyze the Bitonic Sort algorithm in parallel using OpenMP. Bitonic Sort is a comparison-based sorting algorithm that works well on parallel computers due to its predictable pattern of comparisons. Students will implement Bitonic Sort using OpenMP to sort an array of integers. Steps: a) Initialize OpenMP: Set up the OpenMP environment and create an array of random integers. b) Implement Bitonic Sort: Implement the Bitonic Sort algorithm in parallel. Use OpenMP directives to parallelize the sorting stages. c) Compare Results: Validate the sorted array and compare the performance of the parallel Bitonic Sort with a sequential implementation. d) Optimize Performance: Experiment with different OpenMP settings to optimize the performance of the parallel Bitonic Sort.

10.	1	Quick Sort is a widely used sorting algorithm that can be parallelized effectively. Students will implement Quick Sort using OpenMP to sort an array of integers and explore how parallelism affects performance. Steps: a) Initialize OpenMP: Set up the OpenMP environment and create an array of random integers. b) Implement Parallel Quick Sort: Implement the Quick Sort algorithm in parallel. Use OpenMP directives to parallelize the recursive sorting steps. c) Compare Results: Validate the sorted array and compare the performance of the parallel Quick Sort with a sequential implementation. d) Optimize Performance: Experiment with different OpenMP settings to optimize the performance of the parallel Quick Sort.
	2	Implement the Merge Sort algorithm using OpenMP or MPI, with and without load balancing.
11.	1	Implement a parallel sorting algorithm (e.g., Bitonic Sort) and introduce synchronization issues deliberately. Analyze how synchronization issues affected the correctness and performance of the parallel sorting algorithm.

	2	Implement a parallel version of the Bubble Sort algorithm and introduce data races by incorrectly sharing or modifying data across threads. Execute the program and observe incorrect results. Use debugging tools or techniques to identify data races. Apply synchronization constructs (e.g., critical sections) to correct the issues and validate the sorted results.
12.	1	Demonstrate data encryption in cloud storage.
	2	Compare different cloud security tools.
	3	Write a program to implement user authentication in a cloud environment.
	4	Implement basic web service on cloud environment.
13.	1	Demonstrate data encryption in cloud storage.
	2	Demonstrate the use of cloud computing in data analytics.
	3	Compare different cloud security tools.
	4	Write a program to implement user authentication in a cloud environment.
14.	1	Create a simple simulation environment using CloudSim, defining basic elements such as data centers, virtual machines (VMs), and cloudlets (tasks).
	2	Implement and compare different resource allocation and scheduling policies in CloudSim. such as First-Come-First-Serve (FCFS) and Round Robin scheduling. They will analyze the impact of these policies on the performance of cloud services.
	3	Implement and test load balancing and resource provisioning strategies in a cloud environment using CloudSim.