PART A

1. Develop a program to sum two large arrays of integers, A and B, each containing 100 million elements. Perform the addition sequentially, processing each element one by one using a single thread.Utilize OpenMP to parallelize the addition process, employing multiple threads (2, 4, 6, 8, 10) to enhance performance.Additionally, measure and report the execution time for each version during their respective executions.Explain any observed differences in performance between the serial and parallel versions. What factors contribute to the speedup (or lack thereof) in the parallel implementation?Consider a situation where the parallel version with 8 threads performs worse than the version with 4 threads. What could be the possible reasons for this, and how might you address them to improve performance?If the hardware you are using has 4 cores, would you expect the parallel version with 8 threads to perform better or worse than with 4 threads? Justify your answer.
2. Implement the matrix addition of two matrices, MatrixA and MatrixB, each of size 10,000 x 10,000, to produce a result matrix MatrixC. Implement sequentially, where each element of MatrixC is computed one at a time using a single thread.Use OpenMP to parallelize the matrix addition. Employ multiple threads (2, 4 and 8) to perform the operation concurrently.Measure and record the execution time of the parallel implementation for each thread count.Compare the execution times of the serial and parallel implementations. How does the performance of the parallel implementation change as the number of threads increases?
3. Compute the product of given two matrices, MatrixA of size 500 x 1000 and MatrixB of size 1000 x 500. Implement matrix multiplication sequentially and parallel using OpenMP. Employ multiple threads (2, 4, and 8) to perform the operation concurrently. Measure and record the execution time of the parallel implementation for each thread count.Compare the execution times of the serial and parallel implementations. How does the performance improve with the increase in the number of threads? What is the maximum speedup achieved?Explain any performance bottlenecks observed as the number of threads increases. Why might the parallel implementation with more threads not always lead to a proportionate decrease in execution time?
4. A prime pair or twin prime is a prime number that has a prime gap of two, in other words, the difference between the two prime numbers are two, for example the twin prime pair (41, 43). You need to write an OpenMP program to find the total number of prime pairs between 2 and 50,000,000. Your grade will be not only determined by the correctness of the total number, but also depends on your program performance. Your program should print the number of prime pairs and the total execution time of

your program. Report the speedup of your program using 1, 2, 4, and 8 threads respectively. Compare the execution times of the serial and parallel implementations. How does the performance improve with the increase in the number of threads? What is the maximum speedup and parallel efficiency achieved?

5. In parallel computing, merging two sorted lists into a single sorted list is a common problem that can be optimized using parallel algorithms. Given two large, sorted lists ListA and ListB, the goal is to merge them into a single sorted list Merged List efficiently using OpenMP.

   List A = [1, 4, 7, 10, 13, 16]

   List B = [2, 3, 8, 9, 14, 15]

   Implement a parallel algorithm using OpenMP that merges these two lists into a single sorted list.

   Explain how you will divide the work among multiple threads to ensure that the merging is done efficiently. Describe the parallelization strategy and how OpenMP will be utilized. Suppose you are using 4 threads to merge the lists. Explain how you would assign portions of the lists to different threads for merging. Discuss the process of determining the starting and ending points of the merge operation for each thread to avoid overlap and ensure that the final MergedList is correctly sorted. Simulate the parallel merging process for the given List A and List B. Show how the elements from both lists are assigned to different threads and merged. Provide the final output of MergedList after the parallel merge operation is completed. Compare the performance of your parallel merging algorithm with a sequential merge. Analyze the conditions under which the parallel merge offers significant speedup. Discuss potential bottlenecks in the parallel implementation, such as load imbalance among threads or contention for shared resources.

6. Implement a parallel program to process a large array of integers using OpenMP. The goal is to compute the square of each element in the array and store the results in another array. The array consists of 100 elements, and you want to utilize all available processor cores on your system to speed up the computation. Ensure that the parallelization is efficient and takes full advantage of the available cores. Measure the execution time for the parallelized loop and compare it with a serial implementation.

7. Given two matrices A and B. Write an OpenMP program to compute the product matrix C. Explain how the rows of matrix C will be distributed among threads when using 4 threads. Simulate the parallel matrix multiplication process and show the final output matrix C. Analyze the performance of the parallel implementation.

Discuss how the choice of the number of threads and scheduling strategy affects the efficiency of the program.

$$A = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix} \quad B = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix}$$

8. Write a parallel program using OpenMP to compute the largest gap between consecutive prime numbers for all integers less than 1,000,000. Identify all prime numbers less than 1,000,000 in parallel. Use multiple threads to calculate the gaps between consecutive primes (1, 2, 4, 8). Determine the largest gap efficiently using OpenMP's features like critical sections or reduction. Explain how the program parallelizes the task of identifying primes and calculating the gaps. Also, discuss potential bottlenecks in the implementation and suggest ways to optimize the program further.
9. A college plans to assign unique six-digit identification numbers to its students but must adhere to specific constraints for "acceptable" identifiers:

   The first digit cannot be 0.

   No two consecutive digits can be the same.

   The sum of the digits cannot equal 7, 11, or 13.

   Write a parallel program using OpenMP to count the total number of valid six-digit identification numbers that satisfy these constraints. Explanation of how OpenMP is used for parallelization and how the constraints are checked.

   Show the total number of valid six-digit identifiers.
10. Write a parallel program using OpenMP to compute the sum of integers from I to p, i.e., I+(I+1)+⋯+p. where I=1 and p=10. Each thread should compute a portion of the sum, and the program should use a parallel reduction to calculate the total sum. Additionally, the program should compute the expected sum using the formula and print both the computed sum and the expected sum for verification.

$$\text{Sum} = \frac{p(p+1)}{2} - \frac{(I-1)I}{2}$$

PART B

11. Implement a MPI program to perform the matrix-vector multiplication for the give matrices.

$$A = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix} \quad B = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix}$$

Execute a MPI program in C/C++ that uses point-to-point communication to perform matrix-vector multiplication.

12. Implement a distributed banking system using MPI with one central server (rank 0) and three branch servers (ranks 1, 2, and 3). Each branch starts with five accounts having initial balances: Account 0: 1000, Account 1: 1500, Account 2: 2000, Account 3: 2500, and Account 4: 3000. Each branch performs 5 random transactions per synchronization cycle, where a transaction involves selecting a random account (0–4) and either depositing or withdrawing a random amount (1–500). Withdrawals should only occur if the account has sufficient balance. After completing the transactions, each branch sends its updated balances to the central server. The central server aggregates these updates to calculate synchronized global balances and sends them back to all branches. The program should execute 10 synchronization cycles, with the central server printing the global balances after each cycle and branches printing the received global balances for consistency verification. Write the MPI program to meet these requirements.

13. Write an MPI-based parallel program to calculate the number of times two consecutive odd integers less than 1,000,000 are both prime numbers. For example, 3 and 5, 5 and 7, and 11 and 13 are pairs of consecutive odd primes, while 7 and 9 are not since 9 is not prime. Your program should divide the range [1,000,000] among multiple processes, where each process independently checks for prime pairs within its assigned range. The program should then use MPI communication to aggregate the results and print the total count of such prime pairs.

14. Develop a distributed MPI program to simulate the growth of a bacteria population across multiple colonies, where each process is responsible for simulating the growth of a specific colony. The initial population count is **500**, and there are **four colonies**, each with a fixed growth rate as follows:

    Colony A: 10 new bacteria per iteration

    Colony B: 15 new bacteria per iteration

    Colony C: 20 new bacteria per iteration

Colony D: 5 new bacteria per iteration

Each process calculates the growth of its assigned colony and sends the contribution to the **master process (rank 0)**. The master process aggregates the contributions from all colonies to update the total population and broadcasts the updated total back to all processes.

Each process computes the growth of one colony. Processes send their growth contributions to the master process. The master process updates the total population and broadcasts it to all processes. Run the simulation for **one iteration**, and calculate the final total population.

15. Design a financial analytics tool to process large transaction datasets from multiple accounts using MPI. Each dataset is represented as an array, where each element corresponds to a transaction amount. To analyze spending patterns, the goal is to compute a running total for each account, providing a cumulative amount spent up to each transaction.

    Describe how a parallel prefix sum could be applied to efficiently compute the running total for each transaction in the dataset.

    By using MPI library, how would you partition the transaction dataset among threads or processors to enable efficient prefix sum computation?

    Identify any synchronization or data dependency issues that may arise during the parallel prefix sum implementation, and explain how they can be managed.

    As the transaction dataset increases in size, discuss how the parallel prefix sum would scale in comparison to a sequential approach, and what factors would influence the choice between parallel and sequential methods.

16. Design an MPI-based program to compute the **running total (prefix sum)** for a transaction dataset representing the spending of a single account (Dataset: [100, 200, 50, 150, 300, 250, 320, 400,150, 750, 125, 275, 390, 112, 555, 789]). Divide the dataset among **4 processes**, where each process handles a portion of the transactions, computes local prefix sums, and then communicates with other processes to calculate the global running total using a parallel prefix sum algorithm. Ensure that the final running total is consistent across all processes. Discuss how the program handles synchronization and scalability for large datasets (e.g., 1 million transactions) compared to a sequential approach, and explain how MPI communication is used to resolve data dependencies.

17. Develop an MPI application to perform matrix addition for scientific computations. Given two 4x4 matrices, A and B, compute their sum and store the result in a third matrix, C.

    Write an MPI program that computes the sum of the matrices using a row-oriented approach, where each process handles a row of the matrices.

    Modify the program to use a column-oriented approach, where each process handles a column of the matrices. Describe the changes needed in the code and discuss how this change might affect performance.

    Test both approaches and compare their parallel efficiency and speedup.

    Consider the following matrices A and B:

$$A = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix} \quad B = \begin{bmatrix} 16 & 15 & 14 & 13 \\ 12 & 11 & 10 & 9 \\ 8 & 7 & 6 & 5 \\ 4 & 3 & 2 & 1 \end{bmatrix}$$

18. Implement a parallelized version of the HyperQuickSort algorithm using the MPI (Message Passing Interface) library to sort the following array: Array = {34, 72, 13, 88, 21, 90, 45, 3, 56, 27, 67, 1, 99, 12, 38, 79, 40, 25, 61, 92, 17, 84, 19, 75, 8, 53, 22, 66, 49, 31, 5, 43}. Write the MPI code for HyperQuickSort, explaining how the array is split, how each processor sorts its portion, and how communication occurs to ensure the entire array is sorted. Describe how the array will be initially divided among the processors and how you will balance the workload to maximize efficiency. Explain the recursive sorting process for each sub-array and how processors will exchange data during the merging phase to complete the sorting. Explain how performance improves as the number of processors increases.

19. The problem involves finding the maximum value of a mathematical function that models a real-world phenomenon. The function is defined as:

    $f(x, y, z) = -x^2 + 1{,}000{,}000x - y^2 - 40{,}000y - z^2$

    To efficiently solve this, you will implement a **parallel genetic algorithm** approach using **MPI** (Message Passing Interface). Write a parallel program that uses this algorithm to identify the optimal values for x, y, and z that maximize the function.Explain how parallelization could improve performance in finding the maximum. Finally, describe how the program will converge towards the best solution across iterations.

For this scenario, assume the following parameters:

• Each processor maintains a population of 4 candidates, with a total of 4 processors.

• The initial values for x, y, and z are constrained within the range [0, 1000].

• Information sharing is facilitated through the Island Migration model.

• Crossover is performed using a linear crossover method, defined as 0.5×(Parent 1+Parent 2)

• The mutation rate is set to 10%.

• The maximum number of iterations allowed is 10.

20. The problem involves finding the maximum value of a mathematical function that models a real-world phenomenon. The function is defined as:

$f(x, y, z) = -x^2 + 1{,}000{,}000x - y^2 - 40{,}000y - z^2$

To efficiently solve this, you will implement a **parallel genetic algorithm** approach using **MPI** (Message Passing Interface). Write a parallel program that uses this algorithm to identify the optimal values for x, y, and z that maximize the function.Explain how parallelization could improve performance in finding the maximum. Finally, describe how the program will converge towards the best solution across iterations.

For this scenario, assume the following parameters:

• Each processor maintains a population of 4 candidates, with a total of 4 processors.

• The initial values for x, y, and z are constrained within the range [0, 1000].

• Information sharing is facilitated through the Stepping-Stone model

.• Crossover is performed using a linear crossover method, defined as 0.5×(Parent 1+Parent 2)

• The mutation rate is set to 10%.

• The maximum number of iterations allowed is 10.