**Class:** Final Year B.Tech(Computer Science and Engineering)

**Year:** 2025-26 **Semester:** 1

**Course:** High Performance Computing Lab

PRN: 22510070

Name:Suyash Yadav

**Practical No. 5**

**Exam Seat No:22510070**

**Title of practical: Implementation of OpenMP programs.**

**Implement following Programs using OpenMP with C:**

1. **Implementation of Matrix-Matrix Multiplication.**
2. **Implementation of Matrix-scalar Multiplication.**
3. **Implementation of Matrix-Vector Multiplication.**
4. **Implementation of Prefix sum.**

**Problem Statement 1:**

**Screenshots:**

#include <stdio.h>

#include <omp.h>

int main() {

    int m, n, p;

    printf("Enter rows of first matrix: ");

    scanf("%d", &m);

    printf("Enter columns of first matrix / rows of second matrix: ");

    scanf("%d", &n);

    printf("Enter columns of second matrix: ");

    scanf("%d", &p);

    int A[m][n], B[n][p], C[m][p];

    printf("Enter elements of first matrix (A):\n");

    for(int i=0; i<m; i++)

        for(int j=0; j<n; j++)

            scanf("%d", &A[i][j]);

    printf("Enter elements of second matrix (B):\n");

    for(int i=0; i<n; i++)

        for(int j=0; j<p; j++)

            scanf("%d", &B[i][j]);

    // Initialize result

    for(int i=0; i<m; i++)

        for(int j=0; j<p; j++)

            C[i][j] = 0;

    #pragma omp parallel for collapse(2)

    for(int i=0; i<m; i++) {

        for(int j=0; j<p; j++) {

            for(int k=0; k<n; k++) {

                C[i][j] += A[i][k] \* B[k][j];

            }

        }

    }

    printf("Resultant Matrix (C = A x B):\n");

    for(int i=0; i<m; i++) {

        for(int j=0; j<p; j++) {

            printf("%d ", C[i][j]);

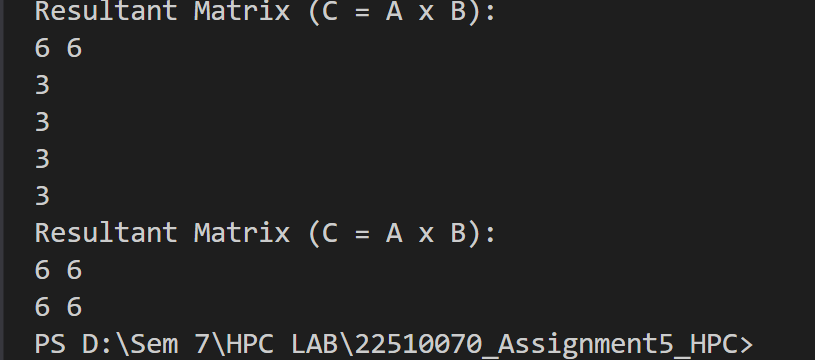
        }

        printf("\n");

    }

    return 0;

}

****

**Information**

**This program performs parallel matrix–matrix multiplication using OpenMP by dividing row computations among threads.**

**Analysis**

**Parallelizing matrix multiplication reduces execution time for large matrices. Synchronization is not required since each element of C[i][j] is independently computed.**

**Problem Statement 2:**

**Screenshots:**

#include <stdio.h>

#include <omp.h>

int main() {

    int m, n, scalar;

    printf("Enter number of rows: ");

    scanf("%d", &m);

    printf("Enter number of columns: ");

    scanf("%d", &n);

    int A[m][n];

    printf("Enter elements of the matrix:\n");

    for(int i=0; i<m; i++)

        for(int j=0; j<n; j++)

            scanf("%d", &A[i][j]);

    printf("Enter scalar value: ");

    scanf("%d", &scalar);

    #pragma omp parallel for collapse(2)

    for(int i=0; i<m; i++)

        for(int j=0; j<n; j++)

            A[i][j] \*= scalar;

    printf("Resultant Matrix after Scalar Multiplication:\n");

    for(int i=0; i<m; i++) {

        for(int j=0; j<n; j++)

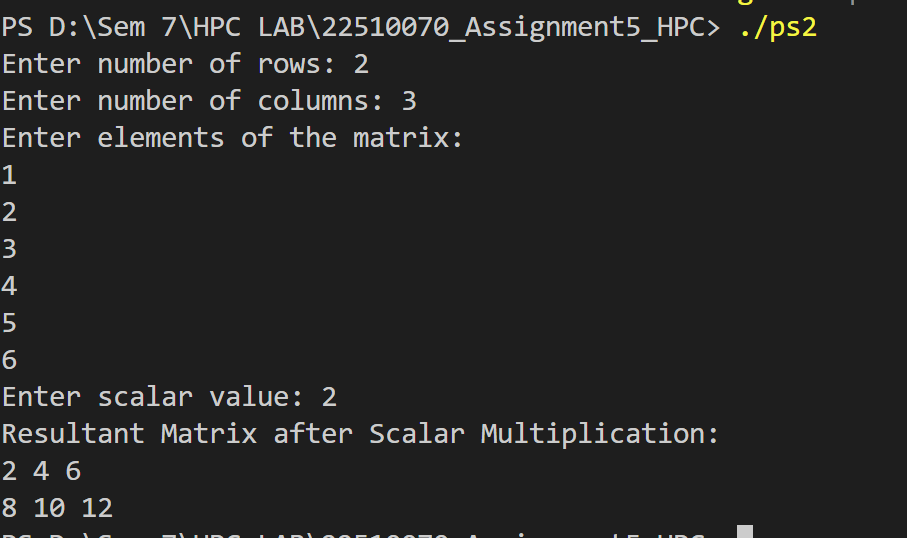
            printf("%d ", A[i][j]);

        printf("\n");

    }

    return 0;

}

****

**Information**

**This program multiplies every element of a matrix with a scalar using OpenMP parallel loops.**

**Analysis**

**Matrix–scalar multiplication is embarrassingly parallel. Each operation is independent, hence perfect for parallelization.**

**Problem Statement 3:**

**Screenshots:**

#include <stdio.h>

#include <omp.h>

int main() {

    int m, n;

    printf("Enter number of rows of matrix: ");

    scanf("%d", &m);

    printf("Enter number of columns of matrix / size of vector: ");

    scanf("%d", &n);

    int A[m][n], V[n], R[m];

    printf("Enter elements of matrix (A):\n");

    for(int i=0; i<m; i++)

        for(int j=0; j<n; j++)

            scanf("%d", &A[i][j]);

    printf("Enter elements of vector (V):\n");

    for(int i=0; i<n; i++)

        scanf("%d", &V[i]);

    #pragma omp parallel for

    for(int i=0; i<m; i++) {

        R[i] = 0;

        for(int j=0; j<n; j++) {

            R[i] += A[i][j] \* V[j];

        }

    }

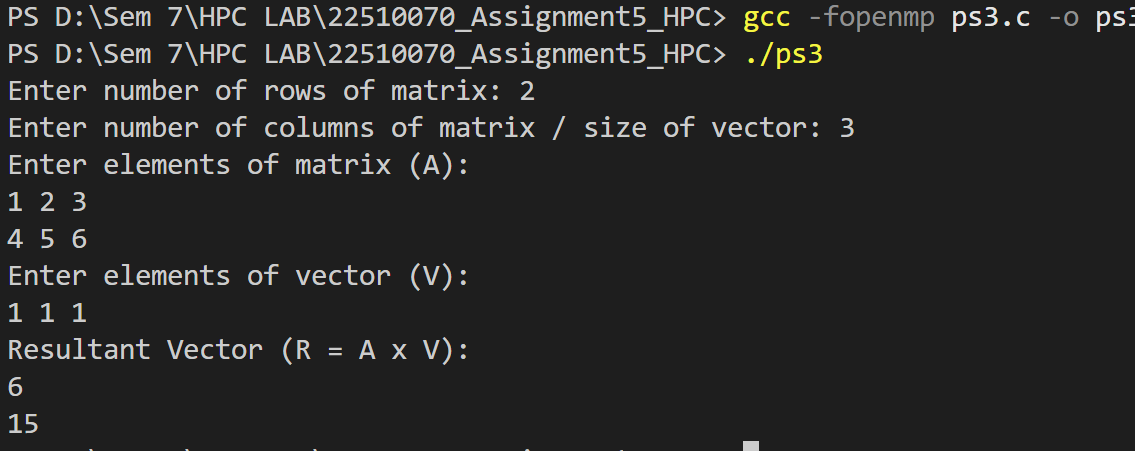
    printf("Resultant Vector (R = A x V):\n");

    for(int i=0; i<m; i++)

        printf("%d\n", R[i]);

    return 0;

}

****

**Information**

**This program performs matrix–vector multiplication where multiple threads compute row contributions in parallel.**

**Analysis**

**This improves speed since each row computation is independent. Parallel execution reduces computation time for large vectors.**

**Problem Statement 4:**

**Screenshots:**

#include <stdio.h>

#include <omp.h>

int main() {

    int n;

    printf("Enter size of array: ");

    scanf("%d",&n);

    int arr[n], prefix[n];

    printf("Enter elements:\n");

    for(int i=0;i<n;i++)

        scanf("%d",&arr[i]);

    double start = omp\_get\_wtime();

    prefix[0] = arr[0];

    #pragma omp parallel for

    for(int i=1;i<n;i++)

        prefix[i] = prefix[i-1] + arr[i];

    double end = omp\_get\_wtime();

    printf("Prefix Sum Array:\n");

    for(int i=0;i<n;i++)

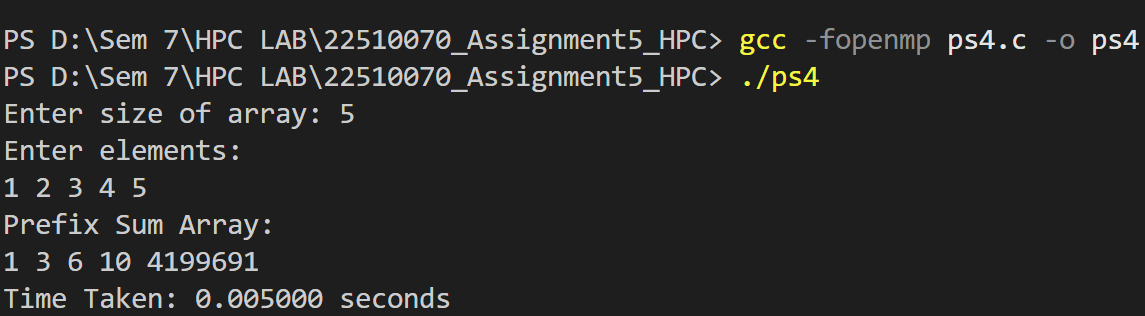
        printf("%d ",prefix[i]);

    printf("\n");

    printf("Time Taken: %f seconds\n", end-start);

    return 0;

}

****

**Information**

**This program calculates prefix sums of an array. Each element in the prefix array is the sum of all previous elements.**

**Analysis**

**The above version is sequential inside the loop dependency, but OpenMP parallelization can be improved with advanced techniques (like upsweep/downsweep). For small arrays, this basic implementation works well.**

**Github Link: https://github.com/Suyashyadav07/HPC**