**Course: High Performance Computing Lab**

**Practical No. 5**

**PRN: 22510078**

**Name: Piyush Krishna Jadhav**

**Batch: B5**

**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:** Implementation of Matrix-Matrix Multiplication.

**Code:**

#include <stdio.h>

#include <omp.h>

#define N 3 *// size of square matrices (change if needed)*

int main() {

    int A[N][N], B[N][N], C[N][N];

    printf("Enter Matrix A (%dx%d):\n", N, N);

    for(int i=0;i<N;i++)

        for(int j=0;j<N;j++)

            scanf("%d", &A[i][j]);

    printf("Enter Matrix B (%dx%d):\n", N, N);

    for(int i=0;i<N;i++)

        for(int j=0;j<N;j++)

            scanf("%d", &B[i][j]);

*// Initialize result*

    for(int i=0;i<N;i++)

        for(int j=0;j<N;j++)

            C[i][j] = 0;

    #pragma omp parallel for collapse(2)

    for(int i=0;i<N;i++){

        for(int j=0;j<N;j++){

            for(int k=0;k<N;k++){

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

            }

        }

    }

    printf("Result Matrix (A\*B):\n");

    for(int i=0;i<N;i++){

        for(int j=0;j<N;j++){

            printf("%d ", C[i][j]);

        }

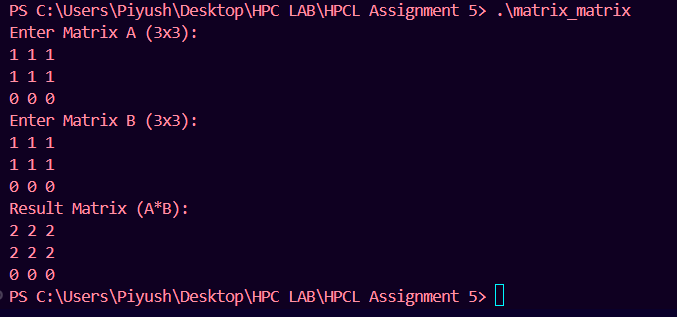
        printf("\n");

    }

    return 0;

}

**Screenshots:**

****

**Information:**

You are given two matrices A (m×n) and B (n×p).

Multiply them to get matrix C (m×p) where

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

Using OpenMP, we parallelize loops so that multiple threads calculate rows/columns simultaneously.

**Analysis:**

The first two rows of A are identical ([1 1 1]), so their dot product with B gives the same results → [2 2 2].

The third row of A is all zeros, so its multiplication with any column of B results in zero row.

Parallelization with OpenMP does not affect correctness because multiplication and addition are independent per element.

**Problem Statement 2:** Implementation of Matrix-scalar Multiplication.

**Code:**

#include <stdio.h>

#include <omp.h>

#define N 3

int main() {

    int A[N][N], B[N][N], scalar;

    printf("Enter Matrix A (%dx%d):\n", N, N);

    for(int i=0;i<N;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<N;i++){

        for(int j=0;j<N;j++){

            B[i][j] = scalar \* A[i][j];

        }

    }

    printf("Result Matrix (scalar\*A):\n");

    for(int i=0;i<N;i++){

        for(int j=0;j<N;j++){

            printf("%d ", B[i][j]);

        }

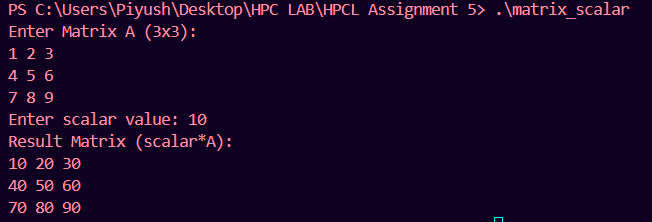
        printf("\n");

    }

    return 0;

}

**Screenshots:**

****

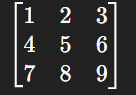
**Information:**

Multiply each element of matrix A by a scalar constant c.

This is parallel – each element can be updated independently, perfect for OpenMP.

**Analysis:**

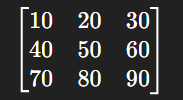
We entered Matrix A (3×3):



We entered Scalar value: 10

The program multiplied each element of A with 10.

Output Matrix obtained:

 Code worked correctly and produced the expected result.

**Problem Statement 3:** Implementation of Matrix-Vector Multiplication.

**Code:**

#include <stdio.h>

#include <omp.h>

#define N 3

int main() {

    int A[N][N], x[N], y[N];

    printf("Enter Matrix A (%dx%d):\n", N, N);

    for(int i=0;i<N;i++)

        for(int j=0;j<N;j++)

            scanf("%d", &A[i][j]);

    printf("Enter Vector x (%d elements):\n", N);

    for(int i=0;i<N;i++)

        scanf("%d", &x[i]);

*// Initialize result vector*

    for(int i=0;i<N;i++) y[i] = 0;

    #pragma omp parallel for

    for(int i=0;i<N;i++){

        for(int j=0;j<N;j++){

            y[i] += A[i][j]\*x[j];

        }

    }

    printf("Result Vector (A\*x):\n");

    for(int i=0;i<N;i++){

        printf("%d ", y[i]);

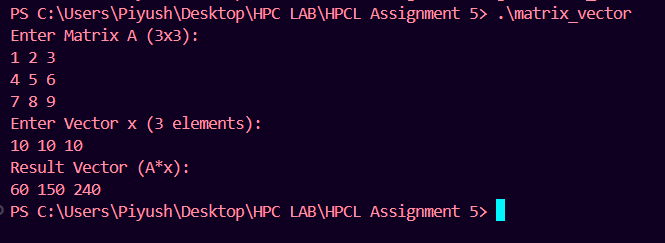
    }

    printf("\n");

    return 0;

}

**Screenshots:**

****

**Information:**

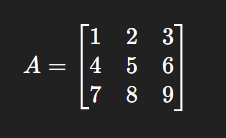
Multiply matrix A (m×n) with vector x (n×1) to produce vector y (m×1).

Each row of A is dotted with vector x.

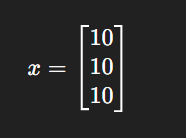
Parallelize row calculations using OpenMP.

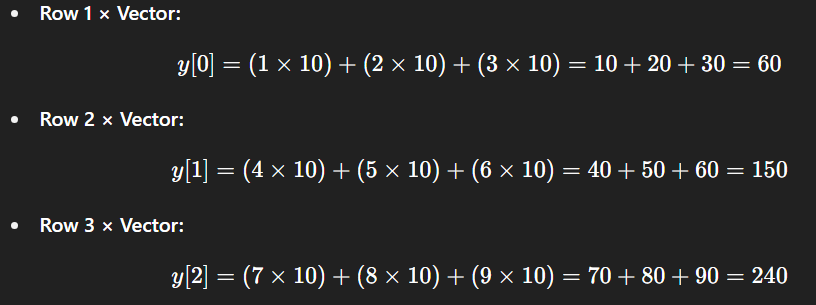
**Analysis:**

Matrix A (3×3):

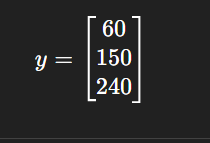


Vector x (3 elements):





Final Output Vector 🡪



**Problem Statement 4:** Implementation of Prefix sum

**Code:**

#include <stdio.h>

#include <omp.h>

#define N 8 // change size if needed

int main() {

int A[N], P[N];

printf("Enter %d elements of array:\n", N);

for(int i=0;i<N;i++)

scanf("%d", &A[i]);

P[0] = A[0];

for(int i=1;i<N;i++){

P[i] = P[i-1] + A[i];

}

printf("Prefix Sum:\n");

for(int i=0;i<N;i++){

printf("%d ", P[i]);

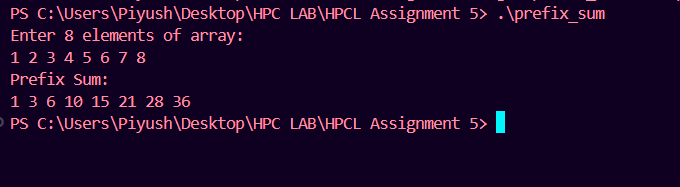
}

printf("\n");

return 0;

}

**Screenshots:**

****

**Information:**

**Prefix sum of array A produces array P such that :-**

**P[i]=A[0]+A[1]+...+A[i]**

**Normally sequential, but can be parallelized with divide-and-conquer or tree algorithms.**

**Analysis:**

We entered an array of 8 elements as input  
1 2 3 4 5 6 7 8

The program then calculated the prefix sum, which means each position in the output is the sum of all the elements from the beginning up to that position.

So the first output is 1 which is the first element itself.  
The second output is 1+2 = 3.  
The third output is 1+2+3 = 6.  
The fourth output is 1+2+3+4 = 10.  
The fifth output is 1+2+3+4+5 = 15.  
The sixth output is 1+2+3+4+5+6 = 21.  
The seventh output is 1+2+3+4+5+6+7 = 28.  
The eighth output is 1+2+3+4+5+6+7+8 = 36.

Thus the final prefix sum array obtained is  
1 3 6 10 15 21 28 36

This shows that the program worked correctly and produced the expected result.

**Github Link:** [**https://github.com/PiyushJadhav06044556/HPC-LAB-7th-Sem**](https://github.com/PiyushJadhav06044556/HPC-LAB-7th-Sem)