Walchand College of Engineering, Sangli
Department of Computer Science and Engineering

Class: Final Year (Computer Science and Engineering)

Year: 2024-25 **Semester:** 1

Course: High Performance Computing Lab

Practical No. 6

Exam Seat No: 21510111

Title of practical: Implementation of OpenMP programs.

Implement following Programs using OpenMP with C:

1. Implementation of Matrix-Matrix Multiplication.
2. Implementation of Matrix-vector Multiplication.

Problem Statement 1: Implementation of Matrix-Matrix Multiplication.

Screenshots:

1.c 2.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>

// Function to multiply two matrices A and B serially, store result in C
void serialMatrixMultiply(int **A, int **B, int **C, int rowA, int colA, int colB) {
    for (int i = 0; i < rowA; i++) {
        for (int j = 0; j < colB; j++) {
            C[i][j] = 0; // Initialize C[i][j]
            for (int k = 0; k < colA; k++) {
                C[i][j] += A[i][k] * B[k][j];
            }
        }
    }
}

// Function to multiply two matrices A and B in parallel using OpenMP, store result in C
void parallelMatrixMultiply(int **A, int **B, int **C, int rowA, int colA, int colB) {
    #pragma omp parallel for collapse(2)
    for (int i = 0; i < rowA; i++) {
        for (int j = 0; j < colB; j++) {
            C[i][j] = 0; // Initialize C[i][j]
            for (int k = 0; k < colA; k++) {
                C[i][j] += A[i][k] * B[k][j];
            }
        }
    }
}

// Function to allocate a matrix dynamically
int **allocateMatrix(int rows, int cols) {
    int **matrix = (int **)malloc(rows * sizeof(int *));
    for (int i = 0; i < rows; i++) {
        matrix[i] = (int *)malloc(cols * sizeof(int));
    }
    return matrix;
}

// Function to initialize a matrix with random integers
void initializeMatrix(int **matrix, int rows, int cols) {
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            matrix[i][j] = rand() % 10; // Random number between 0 and 9
        }
    }
}
```

```c
// Function to print a matrix
void printMatrix(int **matrix, int rows, int cols) {
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            printf("%d ", matrix[i][j]);
        }
        printf("\n");
    }
}

// Main function
int main() {
    int rowA, colA, rowB, colB;
    double start, end;

    // Input matrix dimensions
    printf("Enter rows and columns of matrix A: ");
    scanf("%d %d", &rowA, &colA);

    printf("Enter rows and columns of matrix B: ");
    scanf("%d %d", &rowB, &colB);

    if (colA != rowB) {
        printf("Matrix multiplication not possible with given dimensions!\n");
        return -1;
    }

    // Allocate and initialize matrices
    int **A = allocateMatrix(rowA, colA);
    int **B = allocateMatrix(rowB, colB);
    int **C_serial = allocateMatrix(rowA, colB); // For serial result
    int **C_parallel = allocateMatrix(rowA, colB); // For parallel result

    // Initialize A and B with random values
    initializeMatrix(A, rowA, colA);
    initializeMatrix(B, rowB, colB);

    // Perform serial matrix multiplication
    start = omp_get_wtime();
    serialMatrixMultiply(A, B, C_serial, rowA, colA, colB);
    end = omp_get_wtime();
    printf("Serial Time: %f seconds\n", end - start);

    // Perform parallel matrix multiplication using OpenMP
    start = omp_get_wtime();
    parallelMatrixMultiply(A, B, C_parallel, rowA, colA, colB);
    end = omp_get_wtime();
    printf("Parallel Time: %f seconds\n", end - start);
```

Information:

Analysis:

```
[main][~/acad/hpc_lab/as6]$ gcc -fopenmp 1.c -o 1 && ./1
Enter rows and columns of matrix A: 500 500
Enter rows and columns of matrix B: 500 500
Serial Time: 0.977917 seconds
Parallel Time: 0.261746 seconds
Results are identical.
[main][~/acad/hpc_lab/as6]$
```

Problem Statement 2:
Screenshots:

1.c 2.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>

// Function to multiply matrix A and vector v serially, store result in vector result
void serialMatrixVectorMultiply(int **A, int *v, int *result, int rowA, int colA) {
    for (int i = 0; i < rowA; i++) {
        result[i] = 0; // Initialize result[i] to 0
        for (int j = 0; j < colA; j++) {
            result[i] += A[i][j] * v[j];
        }
    }
}

// Function to multiply matrix A and vector v in parallel using OpenMP, store result in vector result
void parallelMatrixVectorMultiply(int **A, int *v, int *result, int rowA, int colA) {
    #pragma omp parallel for
    for (int i = 0; i < rowA; i++) {
        result[i] = 0; // Initialize result[i] to 0
        for (int j = 0; j < colA; j++) {
            result[i] += A[i][j] * v[j];
        }
    }
}

// Function to allocate a matrix dynamically
int **allocateMatrix(int rows, int cols) {
    int **matrix = (int **)malloc(rows * sizeof(int *));
    for (int i = 0; i < rows; i++) {
        matrix[i] = (int *)malloc(cols * sizeof(int));
    }
    return matrix;
}

// Function to allocate a vector dynamically
int *allocateVector(int size) {
    return (int *)malloc(size * sizeof(int));
}

// Function to initialize a matrix with random integers
void initializeMatrix(int **matrix, int rows, int cols) {
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            matrix[i][j] = rand() % 10; // Random number between 0 and 9
        }
    }
}
```

```c
// Function to initialize a vector with random integers
void initializeVector(int *vector, int size) {
    for (int i = 0; i < size; i++) {
        vector[i] = rand() % 10; // Random number between 0 and 9
    }
}

// Function to print a vector
void printVector(int *vector, int size) {
    for (int i = 0; i < size; i++) {
        printf("%d ", vector[i]);
    }
    printf("\n");
}

// Main function
int main() {
    int rowA, colA;
    double start, end;

    // Input matrix dimensions
    printf("Enter rows and columns of matrix A: ");
    scanf("%d %d", &rowA, &colA);

    // Allocate and initialize matrix A and vector v
    int **A = allocateMatrix(rowA, colA);
    int *v = allocateVector(colA);
    int *result_serial = allocateVector(rowA); // For serial result
    int *result_parallel = allocateVector(rowA); // For parallel result

    // Initialize matrix A and vector v with random values
    initializeMatrix(A, rowA, colA);
    initializeVector(v, colA);

    // Perform serial matrix-vector multiplication
    start = omp_get_wtime();
    serialMatrixVectorMultiply(A, v, result_serial, rowA, colA);
    end = omp_get_wtime();
    printf("Serial Time: %f seconds\n", end - start);

    // Perform parallel matrix-vector multiplication using OpenMP
    start = omp_get_wtime();
    parallelMatrixVectorMultiply(A, v, result_parallel, rowA, colA);
    end = omp_get_wtime();
    printf("Parallel Time: %f seconds\n", end - start);

```

Information:

Analysis:

```
[main][~/acad/hpc_lab/as6]$ gcc -fopenmp 2.c -o 2 && ./2
Enter rows and columns of matrix A: 500 500
Serial Time: 0.001119 seconds
Parallel Time: 0.021906 seconds
Results are identical.
[main][~/acad/hpc_lab/as6]$ gcc -fopenmp 2.c -o 2 && ./2
Enter rows and columns of matrix A: 1000 1000
Serial Time: 0.004044 seconds
Parallel Time: 0.001282 seconds
Results are identical.
[main][~/acad/hpc_lab/as6]$
```

Github Link: https://github.com/Sidd-77/hpc-lab/tree/main/as6