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. 5

Exam Seat No: 22510064 – Parshwa Herwade

Github Link: [Sem-7-Assign/HPC lab at main · parshwa913/Sem-7-Assign · GitHub]

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:

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

int main() {
    int r1, c1, r2, c2;
    printf("Enter rows and cols of matrix A: ");
    scanf("%d %d", &r1, &c1);
    printf("Enter rows and cols of matrix B: ");
    scanf("%d %d", &r2, &c2);

    if (c1 != r2) {
        printf("Matrix multiplication not possible.\n");
        return 0;
    }

    int A[r1][c1], B[r2][c2], C[r1][c2];

    printf("Enter elements of matrix A:\n");
    for (int i = 0; i < r1; i++)
        for (int j = 0; j < c1; j++)
            scanf("%d", &A[i][j]);

    printf("Enter elements of matrix B:\n");
    for (int i = 0; i < r2; i++)
        for (int j = 0; j < c2; j++)
            scanf("%d", &B[i][j]);

    #pragma omp parallel for
    for (int i = 0; i < r1; i++) {
        for (int j = 0; j < c2; j++) {
            C[i][j] = 0;
            for (int k = 0; k < c1; k++) {
                C[i][j] += A[i][k] * B[k][j];
            }
        }
    }

    printf("Resultant Matrix:\n");
```

```c
    for (int i = 0; i < r1; i++) {
        for (int j = 0; j < c2; j++)
            printf("%d ", C[i][j]);
        printf("\n");
    }
    return 0;
}
```

Screenshots:

```
PS C:\Users\Parshwa\Desktop\ASSIGN\HPC lab\22510064_HPC_A5> gcc -fopenmp matrix_matrix.c -o matrix_matrix
PS C:\Users\Parshwa\Desktop\ASSIGN\HPC lab\22510064_HPC_A5> ./matrix_matrix
Enter rows and cols of matrix A: 2 2
Enter rows and cols of matrix B: 2 2
Enter elements of matrix A:
1 2
3 4
Enter elements of matrix B:
5 6
7 8
Resultant Matrix:
19 22
43 50
PS C:\Users\Parshwa\Desktop\ASSIGN\HPC lab\22510064_HPC_A5> gcc -fopenmp matrix_matrix.c -o matrix_matrix
PS C:\Users\Parshwa\Desktop\ASSIGN\HPC lab\22510064_HPC_A5> ./matrix_matrix
Enter rows and cols of matrix A: 4 4
Enter rows and cols of matrix B: 5 5
Matrix multiplication not possible.
PS C:\Users\Parshwa\Desktop\ASSIGN\HPC lab\22510064_HPC_A5> gcc -fopenmp matrix_matrix.c -o matrix_matrix
PS C:\Users\Parshwa\Desktop\ASSIGN\HPC lab\22510064_HPC_A5> ./matrix_matrix
Enter rows and cols of matrix A: 4 5
Enter rows and cols of matrix B: 5 4
Enter elements of matrix A:
1 2 3 4 5
2 3 4 5 6
4 5 6 7 8
4 6 7 8 0
Enter elements of matrix B:
1 2 3 4
4 5 7 8
3 5 6 8
1 3 5 6
6 7 8 2
Resultant Matrix:
52 74 95 78
67 96 124 106
97 140 182 162
57 97 136 168
PS C:\Users\Parshwa\Desktop\ASSIGN\HPC lab\22510064_HPC_A5>
```

Information:

Information (Theory):

Matrix multiplication is a fundamental operation in numerical computing.

If A is of dimension r1×c1 and B is of dimension r2×c2, multiplication is possible only when c1 = r2.
Result matrix C will be of dimension r1×c2 where:
C[i][j] = Σ (A[i][k] × B[k][j]) for k from 0 to c1−1.
OpenMP's #pragma omp parallel for can parallelize the outer loop, allowing different rows to be computed by different threads.

Analysis:
Matrix multiplication has a time complexity of $O(n^3)$ for naive implementation.
By parallelizing the computation, each thread handles a part of the iteration space, improving performance on multi-core systems. The speedup depends on the number of threads and cache efficiency.

Algorithm:

Input dimensions of matrices A and B.

Verify multiplication feasibility (c1 = r2).

Input matrix A and matrix B.

Initialize result matrix C to zero.

Use parallel loop to compute C[i][j] as sum of A[i][k] × B[k][j].

Display result.

Problem Statement 2:

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

int main() {
```

```
    int r, c, scalar;
    printf("Enter rows and cols of matrix: ");
    scanf("%d %d", &r, &c);
    printf("Enter scalar value: ");
    scanf("%d", &scalar);

    int A[r][c];

    printf("Enter elements of matrix:\n");
    for (int i = 0; i < r; i++)
        for (int j = 0; j < c; j++)
            scanf("%d", &A[i][j]);

    #pragma omp parallel for
    for (int i = 0; i < r; i++)
        for (int j = 0; j < c; j++)
            A[i][j] *= scalar;

    printf("Resultant Matrix:\n");
    for (int i = 0; i < r; i++) {
        for (int j = 0; j < c; j++)
            printf("%d ", A[i][j]);
        printf("\n");
    }
    return 0;
}
```

Screenshots:

```
PS C:\Users\Parshwa\Desktop\ASSIGN\HPC lab\22510064_HPC_A5> gcc -fopenmp matrix_scalar.c -o matrix_scalar
>>
PS C:\Users\Parshwa\Desktop\ASSIGN\HPC lab\22510064_HPC_A5> ./matrix_scalar
Enter rows and cols of matrix: 2 2
Enter scalar value: 3
Enter elements of matrix:
1 2
3 4
Resultant Matrix:
3 6
9 12
PS C:\Users\Parshwa\Desktop\ASSIGN\HPC lab\22510064_HPC_A5> gcc -fopenmp matrix_scalar.c -o matrix_scalar
PS C:\Users\Parshwa\Desktop\ASSIGN\HPC lab\22510064_HPC_A5> ./matrix_scalar
Enter rows and cols of matrix: 4 5
Enter scalar value: 3
Enter elements of matrix:
1 2 3 5 6
2 3 5 6 8
1 2 22 3 5
33 44 55 666 777
Resultant Matrix:
3 6 9 15 18
6 9 15 18 24
3 6 66 9 15
99 132 165 1998 2331
PS C:\Users\Parshwa\Desktop\ASSIGN\HPC lab\22510064_HPC_A5>
```

Information:

Information (Theory):
Matrix–scalar multiplication multiplies each element of the matrix by the scalar value. OpenMP can parallelize the iteration over matrix elements so that multiple elements are updated simultaneously.
Analysis:
Time complexity is $O(n^2)$ for an n×n matrix.
With OpenMP, the work is divided among available threads, giving near-linear speedup for large matrices.
Algorithm:

1. Input dimensions of the matrix.
2. Input matrix elements.
3. Input scalar value.
4. Multiply each element by scalar using a parallel loop.
5. Display the resulting matrix.

Problem Statement 3:

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

int main() {
    int r, c;
    printf("Enter rows and cols of matrix: ");
    scanf("%d %d", &r, &c);

    int A[r][c], V[c], R[r];

    printf("Enter elements of matrix:\n");
    for (int i = 0; i < r; i++)
        for (int j = 0; j < c; j++)
            scanf("%d", &A[i][j]);

    printf("Enter elements of vector:\n");
    for (int i = 0; i < c; i++)
        scanf("%d", &V[i]);

    #pragma omp parallel for
    for (int i = 0; i < r; i++) {
        R[i] = 0;
        for (int j = 0; j < c; j++)
            R[i] += A[i][j] * V[j];
    }

    printf("Resultant Vector:\n");
    for (int i = 0; i < r; i++)
        printf("%d ", R[i]);
    printf("\n");

    return 0;
}
```

Screenshots:

```
PS C:\Users\Parshwa\Desktop\ASSIGN\HPC lab\22510064_HPC_A5> gcc -fopenmp matrix_vector.c -o matrix_vector
PS C:\Users\Parshwa\Desktop\ASSIGN\HPC lab\22510064_HPC_A5> ./matrix_vector
Enter rows and cols of matrix: 2 3
Enter elements of matrix:
1 2 4
2 5 6
Enter elements of vector:
3 5 6
Resultant Vector:
37 67
PS C:\Users\Parshwa\Desktop\ASSIGN\HPC lab\22510064_HPC_A5> gcc -fopenmp matrix_vector.c -o matrix_vector
PS C:\Users\Parshwa\Desktop\ASSIGN\HPC lab\22510064_HPC_A5> ./matrix_vector
Enter rows and cols of matrix: 2 3
Enter elements of matrix:
1 2 3
4 5 6
Enter elements of vector:
1  1 1
Resultant Vector:
6 15
PS C:\Users\Parshwa\Desktop\ASSIGN\HPC lab\22510064_HPC_A5>
```

Information:

Information (Theory):
Matrix–vector multiplication produces a vector where each element is the dot product of a row of the matrix with the input vector.
OpenMP parallelizes over rows, assigning each to a different thread.

Analysis:
The complexity is O(r×c).
Parallelization significantly improves performance for large matrices, as each row can be computed independently.

Algorithm:

Input dimensions of the matrix.

Input matrix and vector elements.

Multiply each row of the matrix with the vector using parallel loop.

Store and display the resulting vector.

Problem Statement 4:

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

int main() {
    int n;
    printf("Enter number of elements: ");
    scanf("%d", &n);

    int arr[n], prefix[n];
    printf("Enter elements:\n");
    for (int i = 0; i < n; i++)
        scanf("%d", &arr[i]);

    prefix[0] = arr[0];
    for (int i = 1; i < n; i++) {
        prefix[i] = prefix[i - 1] + arr[i];
    }

    printf("Prefix Sum Array:\n");
    for (int i = 0; i < n; i++)
        printf("%d ", prefix[i]);
    printf("\n");
```

```
    return 0;
}
```

Screenshots:

```
PS C:\Users\Parshwa\Desktop\ASSIGN\HPC lab\22510064_HPC_A5> gcc -fopenmp prefix_sum.c -o prefix_sum
PS C:\Users\Parshwa\Desktop\ASSIGN\HPC lab\22510064_HPC_A5> ./prefix_sum
Enter number of elements: 5
Enter elements:
2 3 5 6 7
Prefix Sum Array:
2 5 10 16 23
PS C:\Users\Parshwa\Desktop\ASSIGN\HPC lab\22510064_HPC_A5> gcc -fopenmp prefix_sum.c -o prefix_sum
PS C:\Users\Parshwa\Desktop\ASSIGN\HPC lab\22510064_HPC_A5> ./prefix_sum
Enter number of elements: 3
Enter elements:
55 678 999
Prefix Sum Array:
55 733 1732
PS C:\Users\Parshwa\Desktop\ASSIGN\HPC lab\22510064_HPC_A5>
```

Information:

Information (Theory):
Prefix sum of an array is an array where each element is the sum of all previous elements including itself.
Formula: prefix[i] = prefix[i−1] + arr[i].
The sequential dependency makes naive parallelization tricky; advanced algorithms like the scan method can parallelize it, but here we demonstrate the basic computation.
Analysis:
Sequential complexity is O(n).
Using advanced parallel algorithms, computation can be reduced to O(log n) with enough threads.
For small arrays, sequential is often faster due to overhead.
Algorithm:

1. Input the number of elements.
2. Input the array elements.
3. Compute prefix sums sequentially (or using parallel scan if implemented).
4. Display the prefix sum array.