

Course: High Performance Computing Lab

Practical No. 5

PRN: 22510057

Name: Ashutosh Gundu Birje

Batch: B8

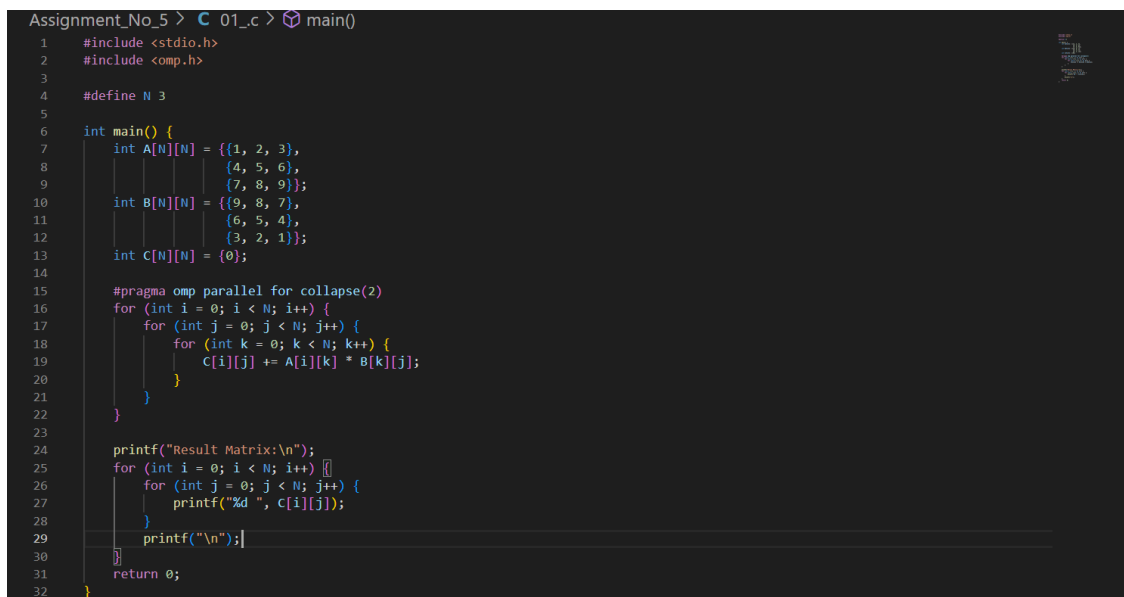
Title of practical: Implementation of OpenMP programs.

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:



```
Assignment_No_5 > C 01_c > main()
1  #include <stdio.h>
2  #include <omp.h>
3
4  #define N 3
5
6  int main() {
7      int A[N][N] = {{1, 2, 3},
8                    {4, 5, 6},
9                    {7, 8, 9}};
10     int B[N][N] = {{9, 8, 7},
11                  {6, 5, 4},
12                  {3, 2, 1}};
13     int C[N][N] = {0};
14
15     #pragma omp parallel for collapse(2)
16     for (int i = 0; i < N; i++) {
17         for (int j = 0; j < N; j++) {
18             for (int k = 0; k < N; k++) {
19                 C[i][j] += A[i][k] * B[k][j];
20             }
21         }
22     }
23
24     printf("Result Matrix:\n");
25     for (int i = 0; i < N; i++) {
26         for (int j = 0; j < N; j++) {
27             printf("%d ", C[i][j]);
28         }
29         printf("\n");
30     }
31     return 0;
32 }
```

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

```
E:\ashutosh\LY\SEM 1\LAB\HPC>cd "e:\ashutosh\LY\SEM 1\LAB\HPC\Assignment_No_5\" && gcc 01_.c -o 01_ && "e:\ashutosh\LY\SEM 1\LAB\HPC\Assignment_No_5\"01_
Result Matrix:
30 24 18
84 69 54
138 114 90
```

Information:

OpenMP Directive Used: #pragma omp parallel for collapse(2) runs the two outer loops in parallel.

Each thread computes one or more rows of the resulting matrix independently.

Synchronization is not needed for each cell since each (i,j) position is computed by exactly one thread.

Analysis:

Without OpenMP: Sequential execution → higher execution time for large N.

With OpenMP: Speed-up depends on number of threads and system cores.

The collapse(2) clause merges two nested loops for better load balancing.

Problem Statement 2:

Screenshots:

```
Assignment_No_5 > C 02_.c > ...
1  #include <stdio.h>
2  #include <omp.h>
3
4  #define N 3
5
6  int main() {
7      int A[N][N] = {{1, 2, 3},
8                    {4, 5, 6},
9                    {7, 8, 9}};
10     int scalar = 3;
11
12     #pragma omp parallel for collapse(2)
13     for (int i = 0; i < N; i++) {
14         for (int j = 0; j < N; j++) {
15             A[i][j] *= scalar;
16         }
17     }
18
19     printf("Result Matrix:\n");
20     for (int i = 0; i < N; i++) {
21         for (int j = 0; j < N; j++) {
22             printf("%d ", A[i][j]);
23         }
24         printf("\n");
25     }
26     return 0;
27 }
```

```
E:\ashutosh\LY\SEM 1\LAB\HPC>cd "e:\ashutosh\LY\SEM 1\LAB\HPC\Assignment_No_5\" && gcc 02_.c -o 02_ && "e:\ashutosh\LY\SEM 1\LAB\HPC\Assignment_No_5\"02_
Result Matrix:
3 6 9
12 15 18
21 24 27
```

Information:

OpenMP Directive: parallel for collapse(2) allows rows and columns to be processed simultaneously.

No synchronization required since each matrix element is updated independently.

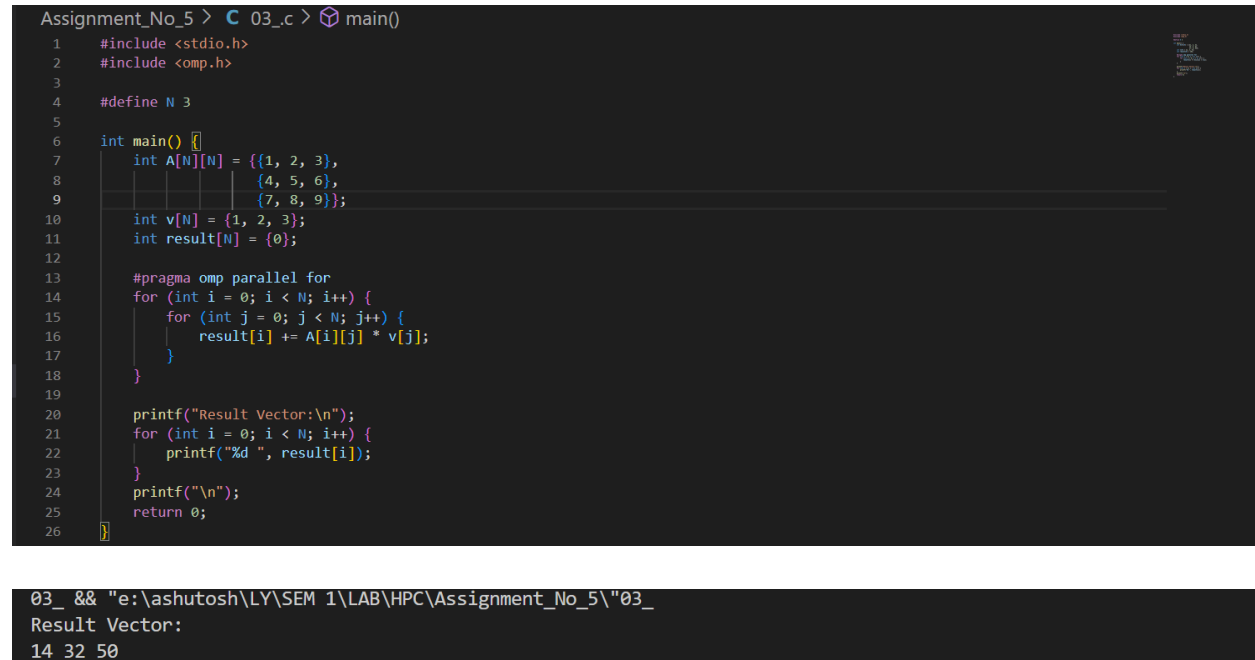
Analysis:

Perfect parallelization with no dependencies.

Achieves nearly linear speed-up for large matrices.

Problem Statement 3:

Screenshots:



```
Assignment_No_5 > C 03_c > main()
1  #include <stdio.h>
2  #include <omp.h>
3
4  #define N 3
5
6  int main() {
7      int A[N][N] = {{1, 2, 3},
8                    {4, 5, 6},
9                    {7, 8, 9}};
10     int v[N] = {1, 2, 3};
11     int result[N] = {0};
12
13     #pragma omp parallel for
14     for (int i = 0; i < N; i++) {
15         for (int j = 0; j < N; j++) {
16             result[i] += A[i][j] * v[j];
17         }
18     }
19
20     printf("Result Vector:\n");
21     for (int i = 0; i < N; i++) {
22         printf("%d ", result[i]);
23     }
24     printf("\n");
25     return 0;
26 }
```

```
03_ && "e:\ashutosh\LY\SEM 1\LAB\HPC\Assignment_No_5\03_
Result Vector:
14 32 50
```

Information:

Each thread calculates one row's dot product with the vector.

Since result[i] is unique to each thread, no synchronization is required.

Analysis:

Highly parallelizable, memory access pattern is sequential for the vector (good cache usage).

Performance improves with larger N.

Problem Statement 4:

Screenshots:

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

```
Assignment_No_5 > C 04_.c > main()
1  #include <stdio.h>
2  #include <omp.h>
3
4  #define N 8
5
6  int main() {
7      int arr[N] = {1, 2, 3, 4, 5, 6, 7, 8};
8      int prefix[N];
9
10     prefix[0] = arr[0];
11     #pragma omp parallel for
12     for (int i = 1; i < N; i++) {
13         prefix[i] = prefix[i - 1] + arr[i];
14     }
15
16     for (int i = 1; i < N; i++) {
17         prefix[i] = prefix[i - 1] + arr[i];
18     }
19
20     printf("Prefix Sum:\n");
21     for (int i = 0; i < N; i++) {
22         printf("%d ", prefix[i]);
23     }
24     printf("\n");
25     return 0;
26 }
```

```
E:\ashutosh\LY\SEM 1\LAB\HPC>cd "e:\ashutosh\LY\SEM 1\LAB\HPC\Assignment_No_5\" && gcc 04_.c -o 04_ && "e:\ashutosh\LY\SEM 1\LAB\HPC\Assignment_No_5\04_
Prefix Sum:
1 3 6 10 15 21 28 36
```

Information:

Prefix sum has data dependency (each element depends on previous).
Fully parallel version requires special algorithms like Blelloch scan.
This example shows why naive parallelization fails and how to fix it.

Analysis:

Sequential dependency limits parallelism.
Optimized parallel algorithms can break work into partial sums and combine results.

Github Link:

<https://github.com/Ashutoshbirje/HPC-LAB/tree/master/Assignment No 5>