

# CSE 4001

## PARALLEL AND DISTRIBUTED COMPUTING



### Lab Assessment – 1

L27+L28 | PLBG04

Dr. Narayanan Prasanth

FALL SEMESTER 2021-22

by

**SHARADINDU ADHIKARI**

19BCE2105

**Q1. Develop an OpenMP program to parallelize the matrix vector multiplication.**Code and Input:

*//Q1. Develop an OpenMP program to parallelize the matrix vector multiplication*

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

//19BCE2105, Sharadindu Adhikari

int main()
{
    int matrix[3][3]={1,2,3},{4,5,6},{7,8,9}};
    int vector[3]={1,2,3};
    int result[3];
    int i,j;
#pragma omp parallel shared(matrix,result,vector) private(i,j)
    {
#pragma omp for schedule(static)
        for (i=0; i<3; i=i+1){
            result[i]=0.;
            for (j=0; j<3; j=j+1){
                result[i]=(result[i])+((matrix[i][j])*(vector[j]));
            }
        }
        for(int k=0;k<3;k++)
        {
            printf("%d \n",result[k]);
        }
    }
    return 0;
}
```

Output:

```
shara-d@Rohans-Workstation:~/PDC$ cd Lab1
shara-d@Rohans-Workstation:~/PDC/Lab1$ gcc -o 1_1 -fopenmp 1_1.c
shara-d@Rohans-Workstation:~/PDC/Lab1$ ./1_1
14
32
50
```

## Screenshot:

```

1 //Q1.Develop an OpenMP program to parallelize the matrix vector multiplication
2
3 #include<stdio.h>
4 #include <omp.h>
5
6 //19BCE2105, Sharadindu Adhikari
7
8 int main()
9 {
10     int matrix[3][3]={1,2,3},{4,5,6},{7,8,9};
11     int vector[3]={1,2,3};
12     int result[3];
13     int i,j;
14     #pragma omp parallel shared(matrix,result,vector) private(i,j)
15     {
16         #pragma omp for schedule(static)
17         for (i=0; i<3; i=i+1){
18             result[i]=0;
19             for (j=0; j<3; j=j+1){
20                 result[i]=(result[i])+((matrix[i][j])*(vector[j]));
21             }
22         }
23     }
24 }

```

```

shara-d@Rohans-Workstation:~/PDC$ cd Lab1
shara-d@Rohans-Workstation:~/PDC/Lab1$ gcc -o 1_1 -fopenmp 1_1.c
shara-d@Rohans-Workstation:~/PDC/Lab1$ ./1_1
14
32
50
shara-d@Rohans-Workstation:~/PDC/Lab1$

```

**Q2. Develop an OpenMP program to find the occurrence of min and max element in the provided list. These operations have to be performed in different sections and thereby executed by different threads. Print the time taken by both the sections. List = {2,3,4,5,5,4,5,3,2,7,8,2}**

### Code and Input:

//Q2. Develop an OpenMP program to find the occurrence of min and max element in the provided list. These operations has to be performed in different sections and thereby executed by different threads. Print the time taken by both the sections. List = {2,3,4,5,5,4,5,3,2,7,8,2}

```

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

```

//19BCE2105, Sharadindu Adhikari

```

int main()
{
    int n=12, i;
    int arr[12]={2,3,4,5,5,4,5,3,2,7,8,2};

    #pragma omp parallel shared(arr) private(i)

```

```
{
#pragma omp sections nowait
{
#pragma omp section
{
    double x = omp_get_wtime();
    int max = arr[0];
    for (i = 1; i < n; i++)
    {
        if (arr[i] > max)
            max = arr[i];
    }
    printf("\nMAX in the array is: %d\n", max);
    double y = omp_get_wtime();
    printf("\nTime Taken for max section: %f\n", (y-x));
}

#pragma omp section
{
    double a = omp_get_wtime();
    int min = arr[0];
    for (i = 1; i < n; i++)
    {
        if (arr[i] < min)
            min = arr[i];
    }
    printf("\nMIN in the array is: %d\n", min);
    double b = omp_get_wtime();
    printf("\nTime Taken for min section: %f\n", (b-a));
}
}
}
```

### **Output:**

```
shara-d@Rohans-Workstation:~/PDC/Lab1$ gcc -o 1_2 -fopenmp 1_2.c
shara-d@Rohans-Workstation:~/PDC/Lab1$ ./1_2
```

```
MIN in the array is: 2
Time Taken for min section: 0.000185
MAX in the array is: 8
Time Taken for max section: 0.000494
```

## Screenshot:

```

1 //Q2. Develop an OpenMP program to find the occurrence of min and max element in the provided List. These operations has :
2
3 #include <stdio.h>
4 #include <omp.h>
5 #include <stdlib.h>
6
7 //19BCE2105, Sharadindu Adhikari
8
9 int main()
10 {
11     int n=12, i;
12     int arr[12]={2,3,4,5,5,4,5,3,2,7,8,2};
13
14     #pragma omp parallel shared(arr) private(i)
15     {
16         #pragma omp sections nowait
17         {
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100

```

```

shara-d@Rohans-Workstation:~/PDC$ cd Lab1
shara-d@Rohans-Workstation:~/PDC/Lab1$ gcc -o 1_2 -fopenmp 1_2.c
shara-d@Rohans-Workstation:~/PDC/Lab1$ ./1_2

MIN in the array is: 2

Time Taken for min section: 0.000185

MAX in the array is: 8

Time Taken for max section: 0.000494
shara-d@Rohans-Workstation:~/PDC/Lab1$

```

**Q3. Develop a C program which counts the number of primes between 1 and N, use OpenMP to carry out the calculation in parallel. Display time taken for each computation of N successfully.**

### Code and Input:

*//Develop a C program which counts the number of primes between 1 and N, use OpenMP to carry out the calculation in parallel. Display the time taken for each computation of N successfully.*

```

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

```

*//19BCE2105, Sharadindu Adhikari*

```

int main ( int argc, char *argv[] );
void prime_number_sweep ( int n_lo, int n_hi, int n_factor );
int prime_number ( int n );

```

```

int main ( int argc, char *argv[] )
{
    int n_factor;
    int n_hi;

```

```
int n_lo;

printf ( "\n" );
printf ( "PRIME_OPENMP\n" );

printf ( "\n" );
printf ( "  Number of processors available = %d\n", omp_get_num_procs ( ) );
printf ( "  Number of threads =                %d\n", omp_get_max_threads ( ) );

n_lo = 1;
n_hi = 1000;
n_factor = 2;

prime_number_sweep ( n_lo, n_hi, n_factor );

printf ( "\n" );
printf ( "PRIME_OPENMP\n" );
printf ( "  Normal end of execution.\n" );

return 0;
}

//*****

void prime_number_sweep ( int n_lo, int n_hi, int n_factor )

{
    int n;
    int primes;
    double wtime;

    printf ( "\n" );
    printf ( "  Call PRIME_NUMBER to count the primes from 1 to N.\n" );
    printf ( "\n" );
    printf ( "          N          Pi          Time\n" );
    printf ( "\n" );

    n = n_lo;
    while ( n <= n_hi )
    {
        wtime = omp_get_wtime ( );

        primes = prime_number ( n );
        wtime = omp_get_wtime ( ) - wtime;
```

```
printf ( " %8d %8d %14f\n", n, primes, wtime );
n = n * n_factor;
}

return;
}

//*****

int prime_number ( int n )
{
    int i;
    int j;
    int prime;
    int total = 0;

#pragma omp parallel \
    shared ( n ) \
    private ( i, j, prime )

#pragma omp for reduction ( + : total )
    for ( i = 2; i <= n; i++ )
    {
        prime = 1;

        for ( j = 2; j < i; j++ )
        {
            if ( i % j == 0 )
            {
                prime = 0;
                break;
            }
        }
        total = total + prime;
    }

    return total;
}
```

### **Output:**

```
shara-d@Rohans-Workstation:~/PDC/Lab1$ gcc -o 1_3 -fopenmp 1_3.c
shara-d@Rohans-Workstation:~/PDC/Lab1$ ./1_3
```

## PRIME\_OPENMP

Number of processors available = 8

Number of threads = 8

Call PRIME\_NUMBER to count the primes from 1 to N.

N	Pi	Time
1	0	0.006347
2	1	0.000667
4	2	0.000005
8	4	0.000005
16	6	0.000005
32	11	0.000006
64	18	0.000006
128	31	0.000004
256	54	0.000008
512	97	0.000021

## PRIME\_OPENMP

Normal end of execution.

## Screenshot:

```

1 //Develop a C program which counts the number of primes between 1 and N, use OpenMP to carry out the calculation in paral
2
3 # include <stdlib.h>
4 # include <stdio.h>
5 # include <omp.h>
6
7 //19BCE2105, Sharadindu Adhikari
8
9 int main ( int argc, char *argv[] );
10 void prime_number_sweep ( int n_lo, int n_hi, int n_factor );
11 int prime_number ( int n );
12
shara-d@Rohans-Workstation:~/PDC/Lab1$ gcc -o 1_3 -fopenmp 1_3.c
shara-d@Rohans-Workstation:~/PDC/Lab1$ ./1_3

PRIME_OPENMP

Number of processors available = 8
Number of threads = 8

Call PRIME_NUMBER to count the primes from 1 to N.

      N      Pi      Time
      1       0  0.006347
      2       1  0.000667
      4       2  0.000005
      8       4  0.000005
     16       6  0.000005
     32      11  0.000006
     64      18  0.000006
    128      31  0.000004
    256      54  0.000008
    512      97  0.000021

PRIME_OPENMP
Normal end of execution.
shara-d@Rohans-Workstation:~/PDC/Lab1$

```



**Q4. Write an OpenMP program to demonstrate the time effectiveness of static, dynamic and guided schedule.**

**Code and Input:**

*//Q4. Write an OpenMP program to demonstrate the time effectiveness of static, dynamic and guided schedule.*

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

//19BCE2105, Sharadindu Adhikari

int main()
{
printf("\nTime effectiveness of Static Schedule\n");
#pragma omp parallel for schedule(static, 3)
for (int i = 0; i < 20; i++)
{
    printf("Thread %d is running number %d\n", omp_get_thread_num(), i);
}
printf("\nTime effectiveness of Dynamic Schedule\n");
#pragma omp parallel for schedule(dynamic, 1)
for (int i = 0; i < 20; i++)
{
    printf("Thread %d is running number %d\n", omp_get_thread_num(), i);
}
printf("\nTime effectiveness of Guided Schedule\n");
omp_set_num_threads(4);
#pragma omp parallel for schedule(guided, 3)
for (int i = 0; i < 20; i++)
{
    printf("Thread %d is running number %d\n", omp_get_thread_num(), i);
}
return 0;
}
```

**Output:**

```
shara-d@Rohans-Workstation:~/PDC/Lab1$ gcc -o 1_4 -fopenmp 1_4.c
shara-d@Rohans-Workstation:~/PDC/Lab1$ ./1_4
```

### Time effectiveness of Static Schedule

Thread 2 is running number 6  
Thread 2 is running number 7  
Thread 2 is running number 8  
Thread 5 is running number 15  
Thread 5 is running number 16  
Thread 5 is running number 17  
Thread 4 is running number 12  
Thread 4 is running number 13  
Thread 4 is running number 14  
Thread 1 is running number 3  
Thread 1 is running number 4  
Thread 1 is running number 5  
Thread 0 is running number 0  
Thread 0 is running number 1  
Thread 0 is running number 2  
Thread 3 is running number 9  
Thread 3 is running number 10  
Thread 3 is running number 11  
Thread 6 is running number 18  
Thread 6 is running number 19

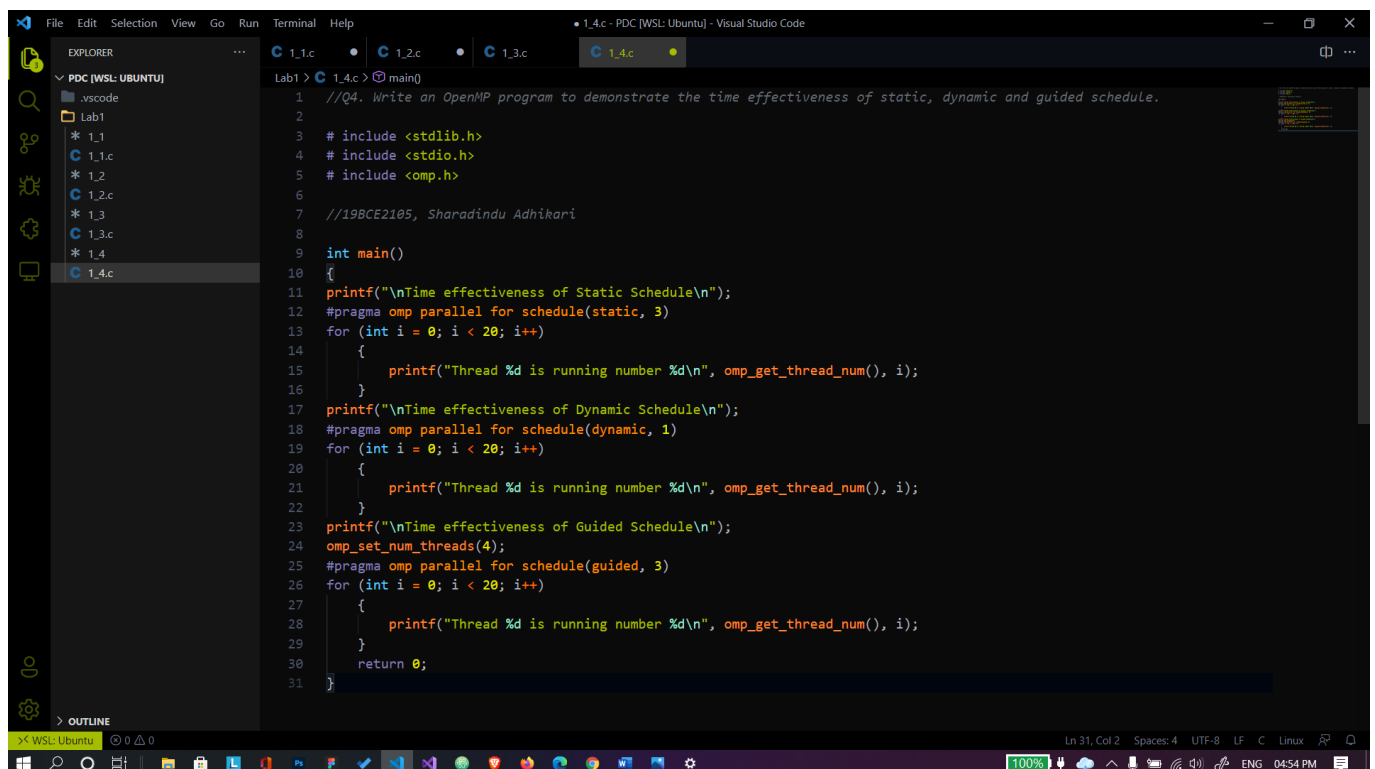
### Time effectiveness of Dynamic Schedule

Thread 2 is running number 0  
Thread 2 is running number 1  
Thread 2 is running number 2  
Thread 2 is running number 3  
Thread 2 is running number 4  
Thread 2 is running number 5  
Thread 2 is running number 6  
Thread 2 is running number 7  
Thread 2 is running number 8  
Thread 2 is running number 9  
Thread 2 is running number 10  
Thread 2 is running number 11  
Thread 2 is running number 12  
Thread 2 is running number 13  
Thread 2 is running number 14  
Thread 2 is running number 15  
Thread 2 is running number 16  
Thread 2 is running number 17  
Thread 2 is running number 18  
Thread 2 is running number 19

## Time effectiveness of Guided Schedule

Thread 0 is running number 0  
Thread 0 is running number 1  
Thread 0 is running number 2  
Thread 0 is running number 3  
Thread 0 is running number 4  
Thread 0 is running number 15  
Thread 0 is running number 16  
Thread 0 is running number 17  
Thread 0 is running number 18  
Thread 0 is running number 19  
Thread 2 is running number 9  
Thread 2 is running number 10  
Thread 2 is running number 11  
Thread 1 is running number 5  
Thread 1 is running number 6  
Thread 1 is running number 7  
Thread 1 is running number 8  
Thread 3 is running number 12  
Thread 3 is running number 13  
Thread 3 is running number 14

## Screenshots:



The screenshot shows the Visual Studio Code editor with a file explorer on the left and a terminal at the bottom. The file explorer shows a project named 'PDC [WSL: UBUNTU]' with a subdirectory 'Lab1' containing files '1.1', '1.1.c', '1.2', '1.2.c', '1.3', '1.3.c', '1.4', and '1.4.c'. The '1.4.c' file is selected. The editor displays the code for '1.4.c', which is an OpenMP program. The code includes headers for `<stdlib.h>`, `<stdio.h>`, and `<omp.h>`. It defines a `main` function that prints the time effectiveness of three scheduling methods: Static, Dynamic, and Guided. Each method is tested with a parallel loop of 20 iterations, with thread IDs and iteration numbers printed for each thread. The Guided schedule is tested with 4 threads. The terminal at the bottom shows the output of the program, which is the same as the text in the first block.

```
1 //Q4. Write an OpenMP program to demonstrate the time effectiveness of static, dynamic and guided schedule.
2
3 #include <stdlib.h>
4 #include <stdio.h>
5 #include <omp.h>
6
7 //19BCE2105, Sharadindu Adhikari
8
9 int main()
10 {
11     printf("\nTime effectiveness of Static Schedule\n");
12     #pragma omp parallel for schedule(static, 3)
13     for (int i = 0; i < 20; i++)
14     {
15         printf("Thread %d is running number %d\n", omp_get_thread_num(), i);
16     }
17     printf("\nTime effectiveness of Dynamic Schedule\n");
18     #pragma omp parallel for schedule(dynamic, 1)
19     for (int i = 0; i < 20; i++)
20     {
21         printf("Thread %d is running number %d\n", omp_get_thread_num(), i);
22     }
23     printf("\nTime effectiveness of Guided Schedule\n");
24     omp_set_num_threads(4);
25     #pragma omp parallel for schedule(guided, 3)
26     for (int i = 0; i < 20; i++)
27     {
28         printf("Thread %d is running number %d\n", omp_get_thread_num(), i);
29     }
30     return 0;
31 }
```

```
File Edit Selection View Go Run Terminal Help • 1_4.c - PDC [WSL: Ubuntu] - Visual Studio Code
EXPLORER
PDC [WSL: UBUNTU]
  .vscode
  Lab1
    * 1_1
    * 1_1.c
    * 1_2
    * 1_2.c
    * 1_3
    * 1_3.c
    * 1_4
    * 1_4.c

Lab1 > C 1_4.c > main()
1 //Q4. Write an OpenMP program to demonstrate the t
2
3 # include <stdlib.h>
4 # include <stdio.h>
5 # include <omp.h>
6
7 //19BCE2105, Sharadindu Adhikari
8

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE bash - Lab1 + - - -
shara-d@Rohans-Workstation:~/PDC$ cd Lab1
shara-d@Rohans-Workstation:~/PDC/Lab1$ gcc -o 1_4 -fopenmp 1_4.c
shara-d@Rohans-Workstation:~/PDC/Lab1$ ./1_4

Time effectiveness of Static Schedule
Thread 2 is running number 6
Thread 2 is running number 7
Thread 2 is running number 8
Thread 5 is running number 15
Thread 5 is running number 16
Thread 5 is running number 17
Thread 4 is running number 12
Thread 4 is running number 13
Thread 4 is running number 14
Thread 1 is running number 3
Thread 1 is running number 4
Thread 1 is running number 5
Thread 0 is running number 0
Thread 0 is running number 1
Thread 0 is running number 2
Thread 3 is running number 9
Thread 3 is running number 10
Thread 3 is running number 11
Thread 6 is running number 18
Thread 6 is running number 19

Time effectiveness of Dynamic Schedule
Thread 2 is running number 0
Thread 2 is running number 1
Thread 2 is running number 2
Thread 2 is running number 3
Thread 2 is running number 4
Thread 2 is running number 5
Thread 2 is running number 6
Thread 2 is running number 7
Thread 2 is running number 8
Thread 2 is running number 9
Thread 2 is running number 10
Thread 2 is running number 11
Thread 2 is running number 12
Thread 2 is running number 13
Thread 2 is running number 14
Thread 2 is running number 15
Thread 2 is running number 16
Thread 2 is running number 17
Thread 2 is running number 18
Thread 2 is running number 19

Time effectiveness of Guided Schedule
Thread 0 is running number 0
Thread 0 is running number 1
Thread 0 is running number 2
Thread 0 is running number 3
Thread 0 is running number 4
Thread 0 is running number 15
Thread 0 is running number 16
Thread 0 is running number 17
Thread 0 is running number 18
Thread 0 is running number 19
Thread 2 is running number 9
Thread 2 is running number 10
Thread 2 is running number 11
Thread 1 is running number 5
Thread 1 is running number 6
Thread 1 is running number 7
Thread 1 is running number 8
Thread 3 is running number 12
Thread 3 is running number 13
Thread 3 is running number 14
shara-d@Rohans-Workstation:~/PDC/Lab1$
```