Name: Atharva Manoj Dagaonkar
Registration No: 20BCE0891
Slot: L27 + L28

Parallel and Distributed Computing Laboratory
Digital Assignment 2

Tools used: VS Code on Ubuntu in Lab and Kali Linux on personal machine.

Q1) (a) Create a Loop work-sharing program using OpenMp.

Procedure:

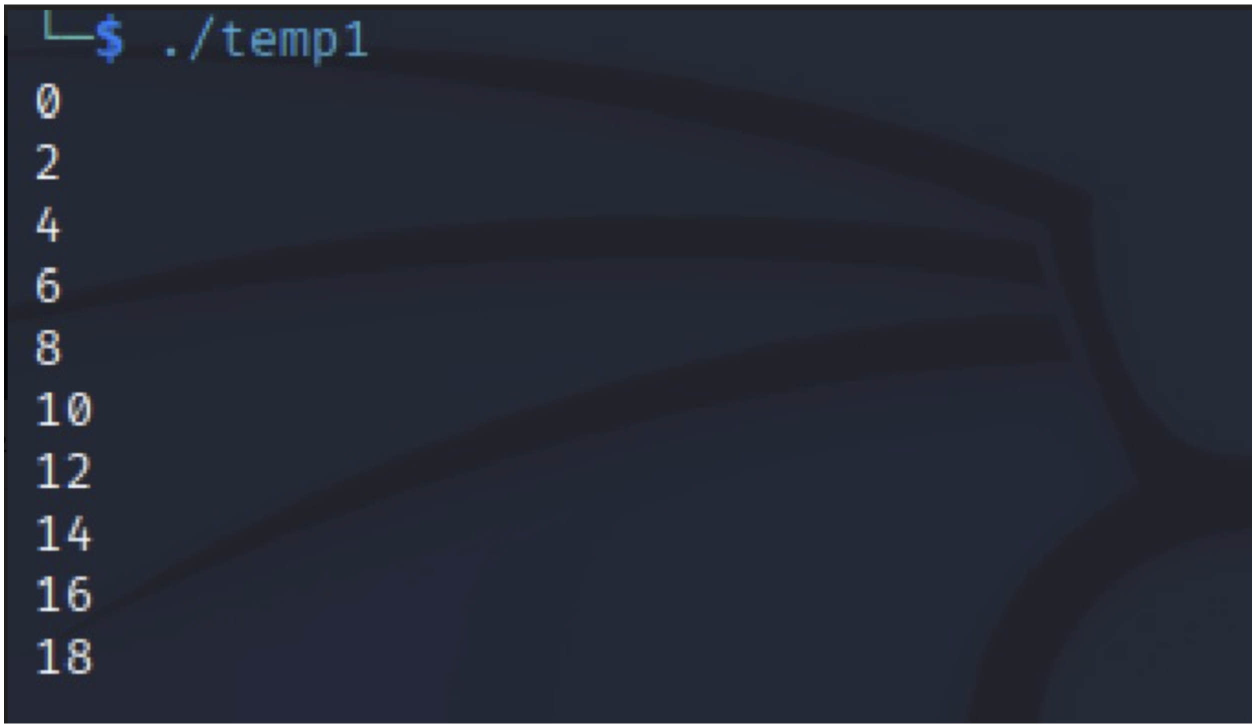
1. Define the array and count variables.
2. Declare the parallel program part by using pragma omp parallel. Use “for” to specify loop sharing
3. Assign values to array in parallel part(taken sequential here).
4. Print the values

Code:

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


int main()
{
   int arr[10],i, count = 10;

   #pragma omp parallel for
   for ( i = 0; i < count; i++)
   {
       arr[i] = 2 * i;
   }

   for ( i = 0; i < count; i++)
   {
      printf("%d \n", arr[i]);
   }
```

```
    return 0;
}
```

Output:



```
└─$ ./temp1
0
2
4
6
8
10
12
14
16
18
```

(b) Create a sections work-sharing program using OpenMp.

Procedure:

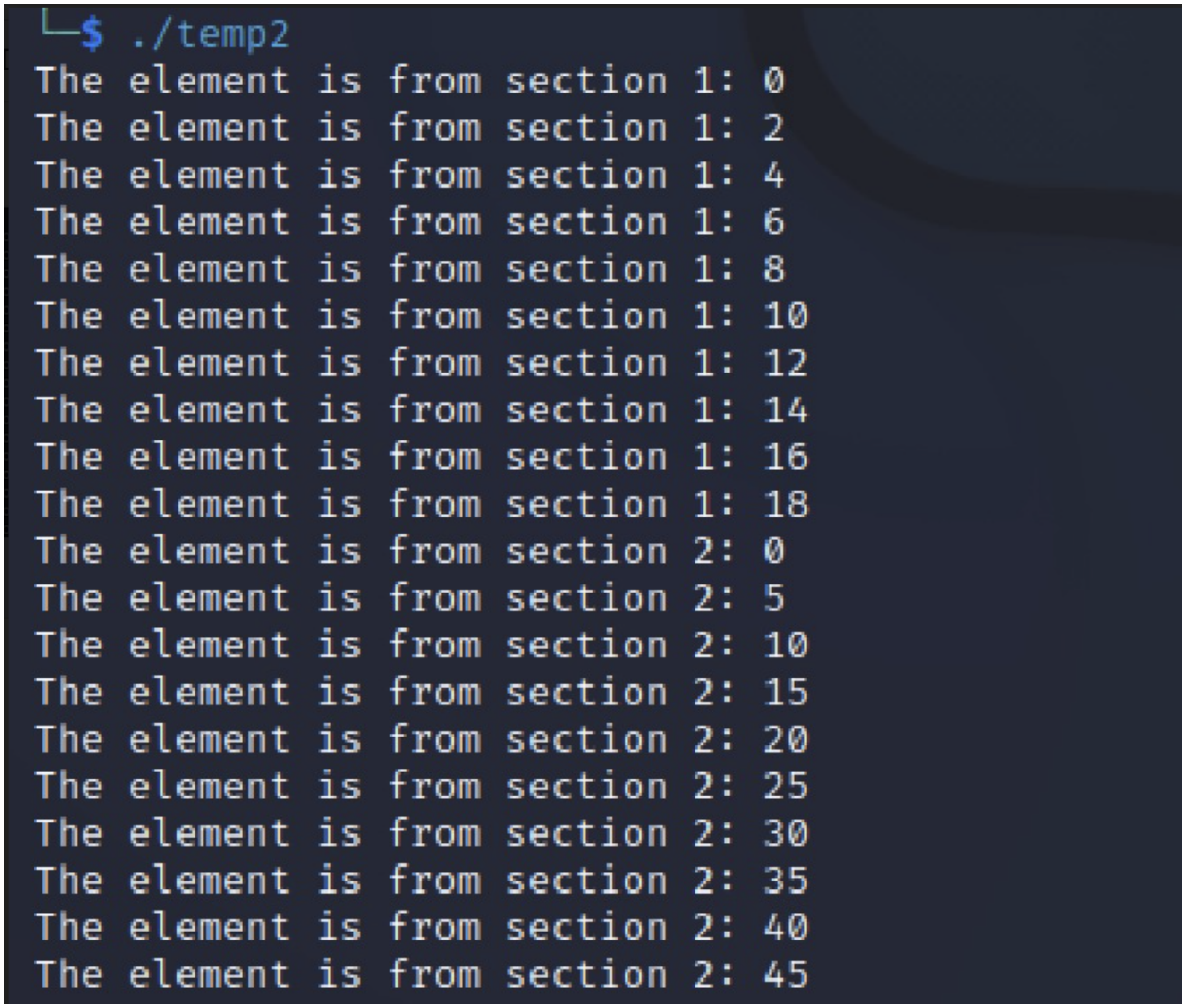
1. Define the arrays and initialize program variables.
2. Define the sections construct using pragma omp parallel sections
3. Write each section which is independent from the others using pragma omp parallel section.
4. In the first section, initialize array 1 values as 2 x index.
5. In the second section, initialize array2 values as 5 x index. This is independent.
6. Print the arrays

Code:

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

```
#include <math.h>


int main()
{
   int arr1[10], arr2[10],i, count = 10;

   #pragma omp sections
   {
       #pragma omp section
           #pragma omp parallel for
               for ( i = 0; i < count; i++)
               {
                   arr1[i] = 2 * i;
               }

               for ( i = 0; i < count; i++)
               {
                  printf("The element is from section 1: %d\n", arr1[i]);
               }

       #pragma omp section
           #pragma omp parallel for
           for ( i = 0; i < count; i++)
           {
               arr2[i] = 5 * i;
           }

           for ( i = 0; i < count; i++)
               {
                   printf("The element is from section 2: %d\n", arr2[i]);
               }

   }


   return 0;
}
```

Output:



```
└─$ ./temp2
The element is from section 1: 0
The element is from section 1: 2
The element is from section 1: 4
The element is from section 1: 6
The element is from section 1: 8
The element is from section 1: 10
The element is from section 1: 12
The element is from section 1: 14
The element is from section 1: 16
The element is from section 1: 18
The element is from section 2: 0
The element is from section 2: 5
The element is from section 2: 10
The element is from section 2: 15
The element is from section 2: 20
The element is from section 2: 25
The element is from section 2: 30
The element is from section 2: 35
The element is from section 2: 40
The element is from section 2: 45
```

Q2) (a) Develop a combined parallel loop reduction program using OpenMp .

Procedure:

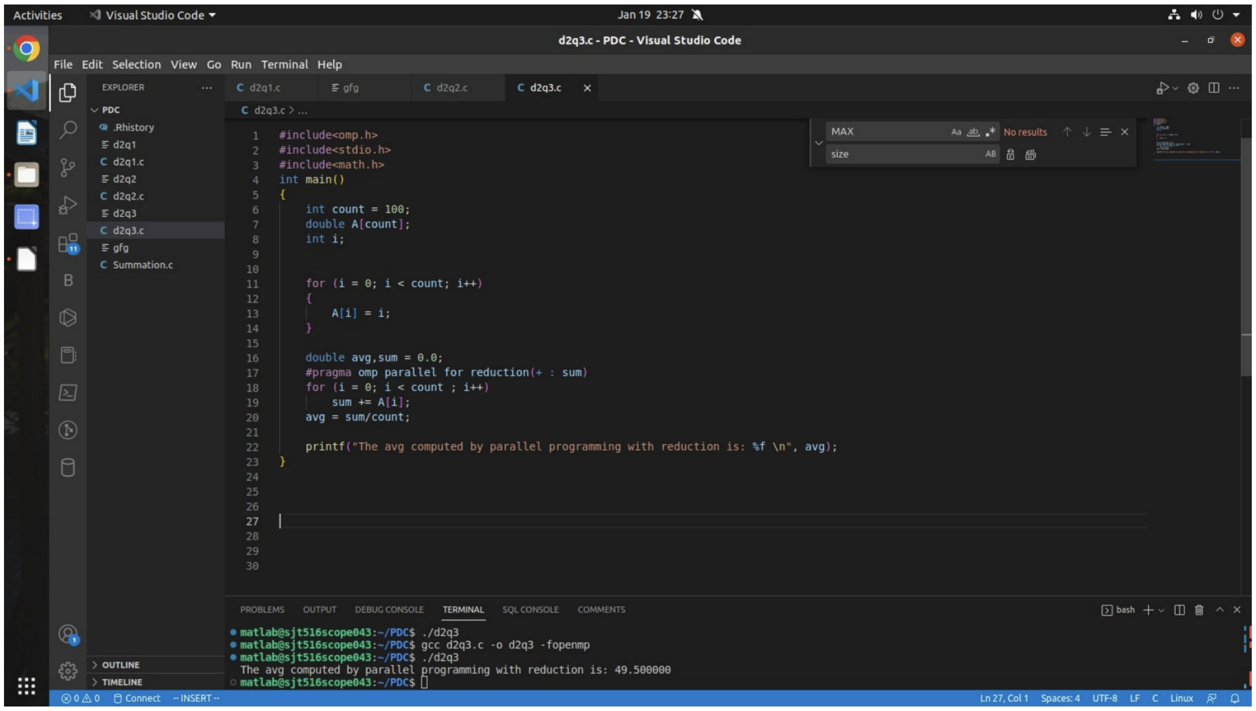
1. Define the array and program variables.
2. Give initialization to the array elements (taken sequential here).
3. Use pragma omp parallel for reduction and use sum as the variable. This will now parallelize the sum computation.
4. Divide sum by count to find average.
5. Print the average.

Code:

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

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


int main()
{
   int count = 10;
   double A[count];
   int i;

   for(i = 0;i<count;i++)
   {
       A[i] = i;
   }

double avg, sum = 0.0;

#pragma omp parallel for reduction(+ : sum)
for ( i = 0; i < count; i++)
{
   sum += A[i];
}
avg = sum/count;

printf("Thre avg computed by parallel programming with reduction is : %d \n", avg);

   return 0;
}
```

Output:



```c
#include<omp.h>
#include<stdio.h>
#include<math.h>
int main()
{
    int count = 100;
    double A[count];
    int i;


    for (i = 0; i < count; i++)
    {
        A[i] = i;
    }

    double avg,sum = 0.0;
    #pragma omp parallel for reduction(+ : sum)
    for (i = 0; i < count ; i++)
        sum += A[i];
    avg = sum/count;

    printf("The avg computed by parallel programming with reduction is: %f \n", avg);
}
```

```
matlab@sjt516scope043:~/PDC$ ./d2q3
matlab@sjt516scope043:~/PDC$ gcc d2q3.c -o d2q3 -fopenmp
matlab@sjt516scope043:~/PDC$ ./d2q3
The avg computed by parallel programming with reduction is: 49.500000
matlab@sjt516scope043:~/PDC$ 
```

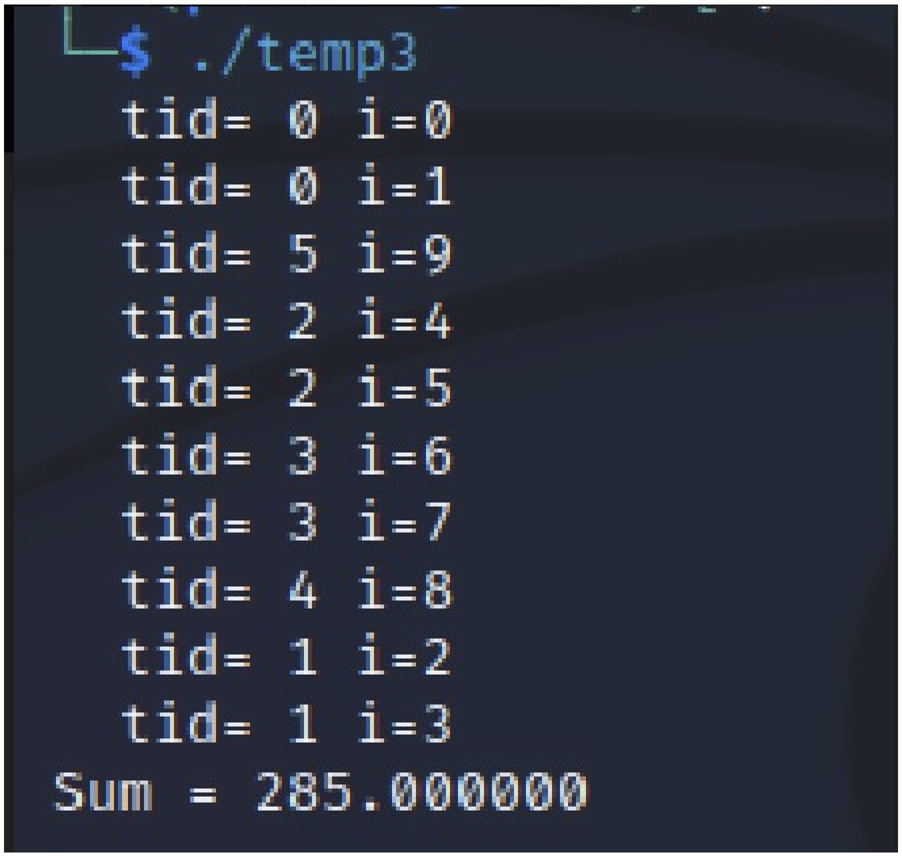
(b) Develop an orphaned parallel loop reduction program using OpenMp.

1. Define global arrays and sum.
2. In main function, initialize both arrays with sequential values.
3. Declare the parallel part using pragma omp parallel. Call the function dotprod to parallelize the dotproduct function.
4. Find the thread ids, and use reduction to calculate the dot product. Add it to sum.
5. Display the dot product.
6. Orphaned section is hence achieved.

Code:

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


int main()
{
    int arr1[10], arr2[10],i, count = 10;

    #pragma omp sections
    {
        #pragma omp section
        {
            #pragma omp parallel
                for ( i = 0; i < count; i++)
                {
                    arr1[i] = 2 * i;
                }

                for ( i = 0; i < count; i++)
                {
                   printf("The element is from section 1: %d\n", arr1[i]);
                }
        }
        #pragma omp section
        {
            #pragma omp parallel
                    for ( i = 0; i < count; i++)
                    {
                           arr2[i] = 5 * i;
                    }

            for ( i = 0; i < count; i++)
                {
                    printf("The element is from section 2: %d\n", arr2[i]);
                }
        }
    }
    return 0;
}
```

Output:



```
└─$ ./temp3
   tid= 0 i=0
   tid= 0 i=1
   tid= 5 i=9
   tid= 2 i=4
   tid= 2 i=5
   tid= 3 i=6
   tid= 3 i=7
   tid= 4 i=8
   tid= 1 i=2
   tid= 1 i=3
 Sum = 285.000000
```