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

High Performance Computing Lab

Class: Final Year (Computer Science and Engineering) **Year:** 2022-23

PRN: 2019BTECS00089 – Piyush Pramod Mhaske **Batch**: B3

Practical No. 3

github link: https://github.com/Piyush4620/2019BTECS00089HPCLab

Title of practical:

To Study and implementation of schedule, no wait, reduction, ordered and collapse.

Problem Statement 1:

To analyze and implement a Parallel code for minimum scalar product program using Open Mp.

Serial Implementation of program for calculating minimum scalar product.

```
// C Program to find the minimum scalar product of two vectors (dot product)
#include<stdio.h>
#include <time.h>       // for clock_t, clock(), CLOCKS_PER_SEC

int sort(int arr[], int n)
{
    int i, j;
    for (i = 0; i < n-1; i++) {
        for (j = 0; j < n-i-1; j++) {
            if (arr[j] > arr[j+1])
            {
                int temp = arr[j];
                arr[j] = arr[j+1];
                arr[j+1] = temp;
            }
        }
    }
}

int sort_des(int arr[], int n)
{
    int i,j;
```

```
    for (i = 0; i < n; ++i)
    {
        for (j = i + 1; j < n; ++j)
        {
            if (arr[i] < arr[j])
            {
                int a = arr[i];
                arr[i] = arr[j];
                arr[j] = a;
            }
        }
    }
}
int main()
{
    double time_spent = 0.0;
    clock_t begin = clock();

    //fill the code;
    int n=5;
    // printf("\nEnter number of elements: ");
    // scanf("%d",&n);
    int arr1[5]={2,1,4,6,3};
    int arr2[5]={2,1,5,4,3};
    int i;
    sort(arr1, n);
    sort_des(arr2, n);
    int sum = 0;
    for(i = 0; i < n ; i++)
    {
        sum = sum + (arr1[i] * arr2[i]);
    }
    printf("\nMinimum scalar product is: ");
    printf("%d",sum);

    clock_t end = clock();
    time_spent += (double)(end - begin) / CLOCKS_PER_SEC;
    printf("\nThe runtime is %f seconds", time_spent);
    return 0;
}
```

Output:

```
PS D:\Academics\Fourth Year\HPC Lab\Assignments\HPC\HPC\Practical3> .\SerialMinScalarProduct.exe

Minimum scalar product is: 36
The runtime is 0.000000 seconds
PS D:\Academics\Fourth Year\HPC Lab\Assignments\HPC\HPC\Practical3>
```

Parallel implementation of program for calculating minimum scalar product.

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

void swap(int *num1, int *num2)
{

  int temp = *num1;
  *num1 =  *num2;
  *num2 = temp;
}

int sort(int arr[],int n) {
    int i,j;
    for (i = 0; i < n-1; i++) {
        #pragma omp parallel for default(none),shared(arr,n,i)
        for (j = 0; j < n-i-1; j++) {
            if (arr[j] > arr[j+1])
            {
                int temp = arr[j];
                arr[j] = arr[j+1];
                arr[j+1] = temp;
            }
        }
    }
}

int dec_sort(int arr[],int n) {
    int i,j;
    for( i = 0; i < n; i++ )
  {
    #pragma omp parallel for default(none),shared(arr,n,i)
    for (j = i + 1; j < n; ++j)
        {
         printf("Thread:%d\n",omp_get_thread_num());
            if (arr[i] < arr[j])
            {
                int a = arr[i];
                arr[i] = arr[j];
                arr[j] = a;
            }
        }
  }

}


int main (int argc, char *argv[]) {
  int SIZE =5;
  int arr1[5]={2,1,4,6,3};
```

```
    int arr2[5]={2,1,5,4,3};
  int n = SIZE;
  int i=0, j=0;
  int first;
  double start,end;
  start=omp_get_wtime();
  sort(arr1,n);
    dec_sort(arr2,n);
    int sum = 0;
    for(i = 0; i < n ; i++)
    {
        sum = sum + (arr1[i] * arr2[i]);
    }
    printf("\nMinimum scalar product is: ");
    printf("%d",sum);
    end=omp_get_wtime();

    printf("\nTime Parallel= %f",(end-start));
}
```

ouput:

```
PS D:\Academics\Fourth Year\HPC Lab\Assignments\HPC\HPC\Practical3> .\ParallelMinScalarProduct.exe
Thread:2
Thread:1
Thread:3
Thread:0
Thread:1
Thread:2
Thread:0
Thread:1
Thread:0
Thread:0

Minimum scalar product is: 36
Time Parallel= 0.006000
PS D:\Academics\Fourth Year\HPC Lab\Assignments\HPC\HPC\Practical3>
```

Information 1:

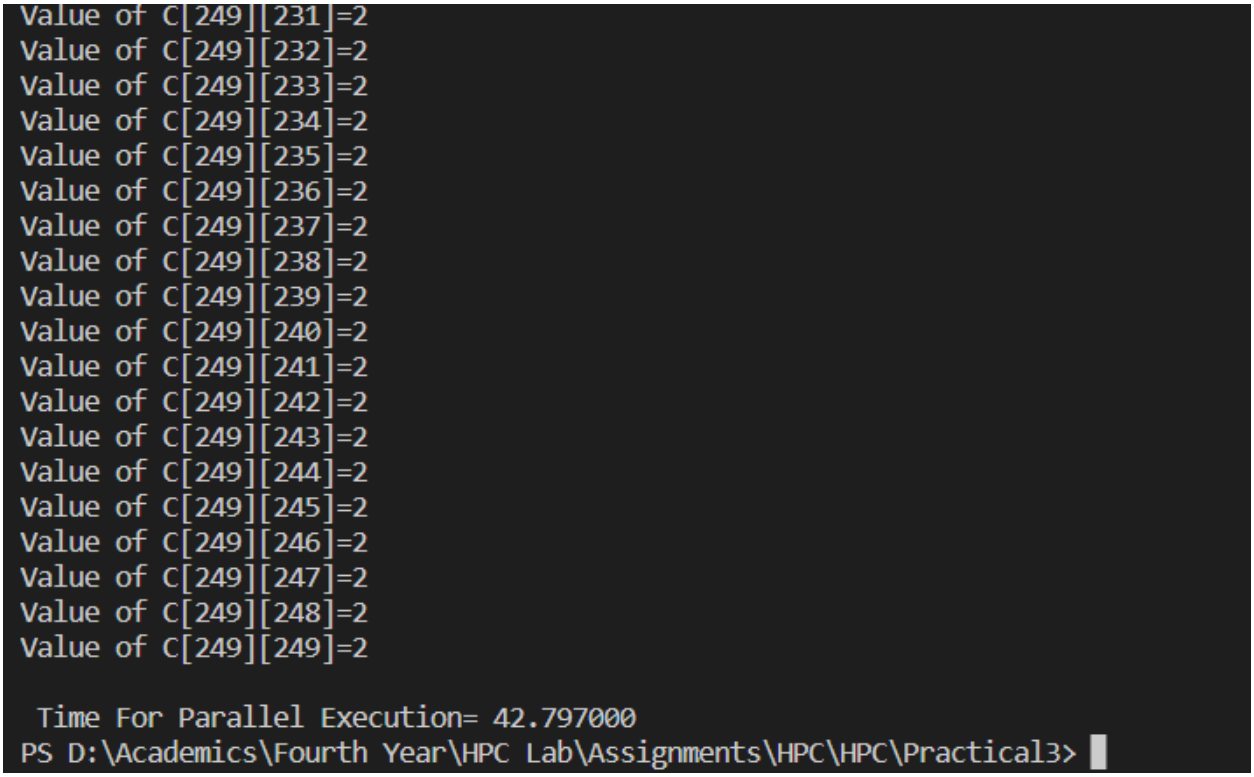
The default clause explicitly determines the data-sharing attributes of variables that are referenced in a parallel, teams, or task generating construct and would otherwise be implicitly determined. The default (shared) clause causes all variables referenced in the construct that have implicitly determined data-sharing attributes to be shared.
The default (none) clause requires that each variable that is referenced in the construct, and that does not have a predetermined data-sharing attribute, must have its data-sharing attribute explicitly determined by being listed in a data-sharing attribute clause.

Problem Statement 2:

To implement a Parallel code in for Two 2D Matrix Addition

```
#include<stdio.h>
#include<stdlib.h>
#include<omp.h>
int main()
{
  int tid;
  int i,j,d;
  int rows=250,cols=250;

  int a[250][250];
  int b[250][250];
  int c[250][250]={0};
  const int value = 1;
//  std::fill(*a, *a + 62500, value);
//  std::fill(*b, *b + 62500, value);
  for(int i=0;i<250;i++){
    for(int j=0;j<250;j++)
      a[i][j]=1;
  }
  for(int i=0;i<250;i++){
    for(int j=0;j<250;j++)
      b[i][j]=1;
  }
  double start,end;
  start=omp_get_wtime();

    for(int i=0;i<rows;i++)
    {
        #pragma omp parallel for reduction(+:d) num_threads(8)
         for(j=0;j<cols;j++)
        {
            printf("Thread:%d\n",omp_get_thread_num());
            c[i][j]=a[i][j]+b[i][j];
            d+=c[i][j];
        }
    }

printf("Values of Resultant Matrix C are as follows:\n");

for(i=0;i<rows;i++)
  for(j=0;j<cols;j++)
    {
       printf("Value of C[%d][%d]=%d\n",i,j,c[i][j]);
    }
    end=omp_get_wtime();

    printf("\n Time For Parallel Execution= %f",(end-start));
```

```
return 0;
}
```



Information 2:

The reduction clauses are data-sharing attribute clauses that can be used to perform some forms of recurrence calculations in parallel. Reduction clauses include reduction scoping clauses and reduction participating clauses. Reduction scoping clauses define the region in which a reduction is computed. Reduction participating clauses define the participants in the reduction. Reduction clauses specify a reduction-identifier and one or more list items. A reduction-identifier is either an identifier or one of the following operators: +, -, *, &, |, ^, && and ||.