**Class:** Final Year (Computer Science and Engineering)

**Year:** 2022-23 **Semester:**1

**Course:** High Performance Computing Lab

**Practical No. 3**

**Exam Seat No:**

2019BTECS00005 – Ashish Sutar

**Title of practical:**

Study and implementation of schedule, nowait, reduction, ordered and collapse clauses

**Problem Statement 1:**

Implement a parallel code for dot product

**Screenshot 1:**

#include<stdio.h>

#include<omp.h>

int sort(int arr[], int n)

{

    int i, j;

    #pragma omp parallel private(j)

    #pragma omp for schedule(dynamic)

    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;

    #pragma omp parallel private(j)

    #pragma omp for schedule(dynamic)

    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()

{

    int arr1[5]={1,2,3,4,5}, arr2[5]={1,2,3,4,5};

    int i;

    double time1 = omp\_get\_wtime();

    sort(arr1, 5);

    sort\_des(arr2, 5);

    int sum = 0;

    #pragma omp parallel for reduction(+:sum)

    for(i = 0; i < 5 ; i++)

    {

        sum = sum + (arr1[i] \* arr2[i]);

    }

    printf("%d",sum);

    double time2 = omp\_get\_wtime();

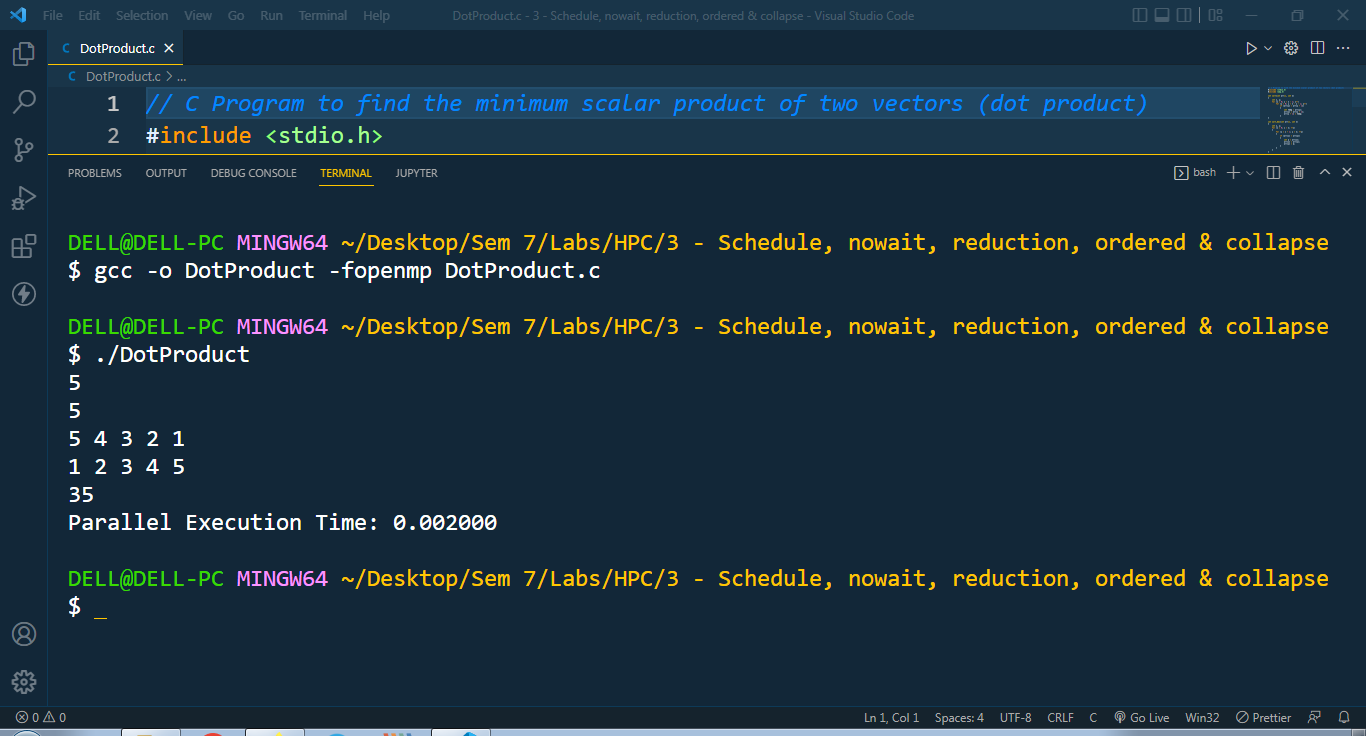
    double time\_requried = time2-time1;

    printf("\nParallel %lf\n", time\_requried);

    return 0;

}

**Screenshot 2:**

****

**Information 1:**

There reduction clause specifies how multiple local copies of a variable at different threads are combined into a single copy at the master when threads exit. The usage of the reduction clause is reduction (operator: variable list). This clause performs a reduction on the scalar variables specified in the list using the operator. The variables in the list are implicitly specified as being private to threads. The operator can be one of +, \*, -, &, |, ^, &&, and ||.

**Problem Statement 2:**

Implement a parallel code for Matrix Addition

**Screenshot 3:**

#include <omp.h>

#include <stdio.h>

#include <stdlib.h>

#define N 100

int main()

{

    int tid, nthreads, i, j;

    int a[N][N], b[N][N], c[N][N];

    omp\_set\_num\_threads(8);

    for (i = 0; i < N; i++)

        for (j = 0; j < N; j++)

            a[i][j] = i + j;

    for (i = 0; i < N; i++)

        for (j = 0; j < N; j++)

            b[i][j] = i + j;

    double time = omp\_get\_wtime();

#pragma omp parallel shared(a, b, c, nthreads) private(tid, i, j)

    {

#pragma omp for

        for (i = 0; i < N; i++)

        {

            for (j = 0; j < N; j++)

                c[i][j] = a[i][j] + b[i][j];

        }

    }

    // printf("\nResultant Matrix:\n");

    // for (i=0; i<N; i++)

    // {

    //  for (j=0; j<N; j++)

    //      printf("%d  ", c[i][j]);

    //  printf("\n");

    // }

    printf("\nDone In %f Seconds", omp\_get\_wtime() - time);

    printf("\n Using %d Threads", omp\_get\_max\_threads());

    return (0);

}

**Information 2:**

The parallel directive can be used in conjunction with other directives to specify concurrency across iterations and tasks. OpenMP provides two directives for and sections to specify concurrent iterations and tasks. The for directive is used to split parallel iteration spaces across threads.

**Github Link:**