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

Class: Final Year (Computer Science and Engineering)

Year: 2024-25 **Semester:** 1

Course: High Performance Computing Lab

Practical No. 2

Exam Seat No: 22520007

Name: Manish Namdev Barage

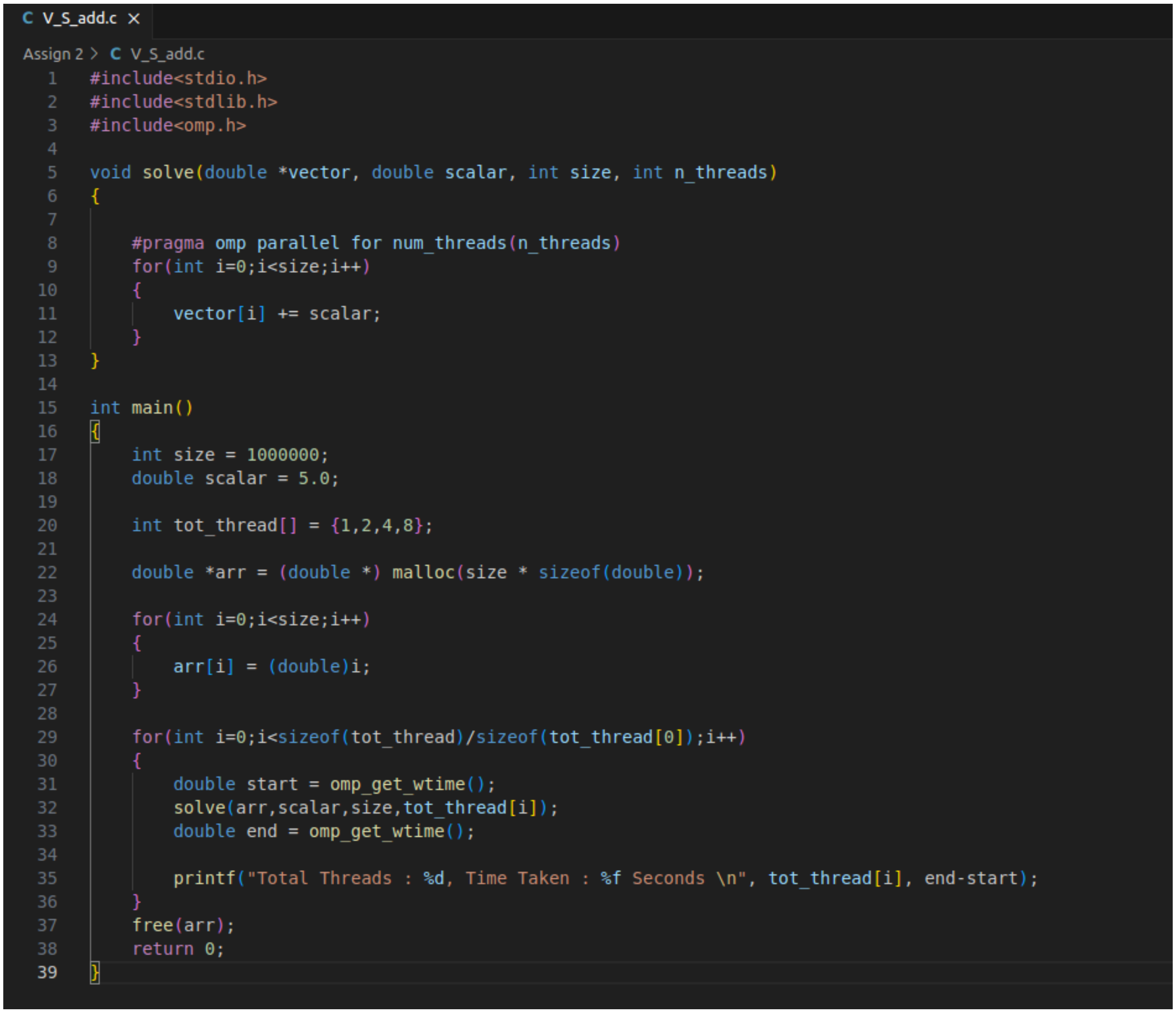
Batch: B6

Title of practical: Study and implementation of basic OpenMP clauses

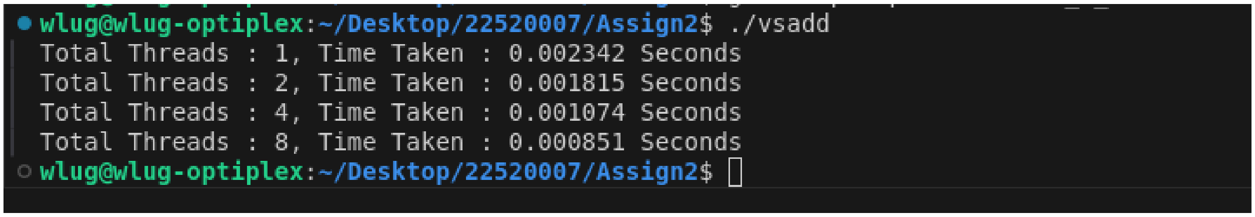
Implement following Programs using OpenMP with C:

1. Vector Scalar Addition
2. Calculation of value of Pi
   Analyse the performance of your programs for different number of threads and Data size.

Problem Statement 1: Vector Scalar Addition
Screenshots:



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

void solve(double *vector, double scalar, int size, int n_threads)
{

    #pragma omp parallel for num_threads(n_threads)
    for(int i=0;i<size;i++)
    {
        vector[i] += scalar;
    }
}

int main()
{
    int size = 1000000;
    double scalar = 5.0;

    int tot_thread[] = {1,2,4,8};

    double *arr = (double *) malloc(size * sizeof(double));

    for(int i=0;i<size;i++)
    {
        arr[i] = (double)i;
    }

    for(int i=0;i<sizeof(tot_thread)/sizeof(tot_thread[0]);i++)
    {
        double start = omp_get_wtime();
        solve(arr,scalar,size,tot_thread[i]);
        double end = omp_get_wtime();

        printf("Total Threads : %d, Time Taken : %f Seconds \n", tot_thread[i], end-start);
    }
    free(arr);
    return 0;
}
```

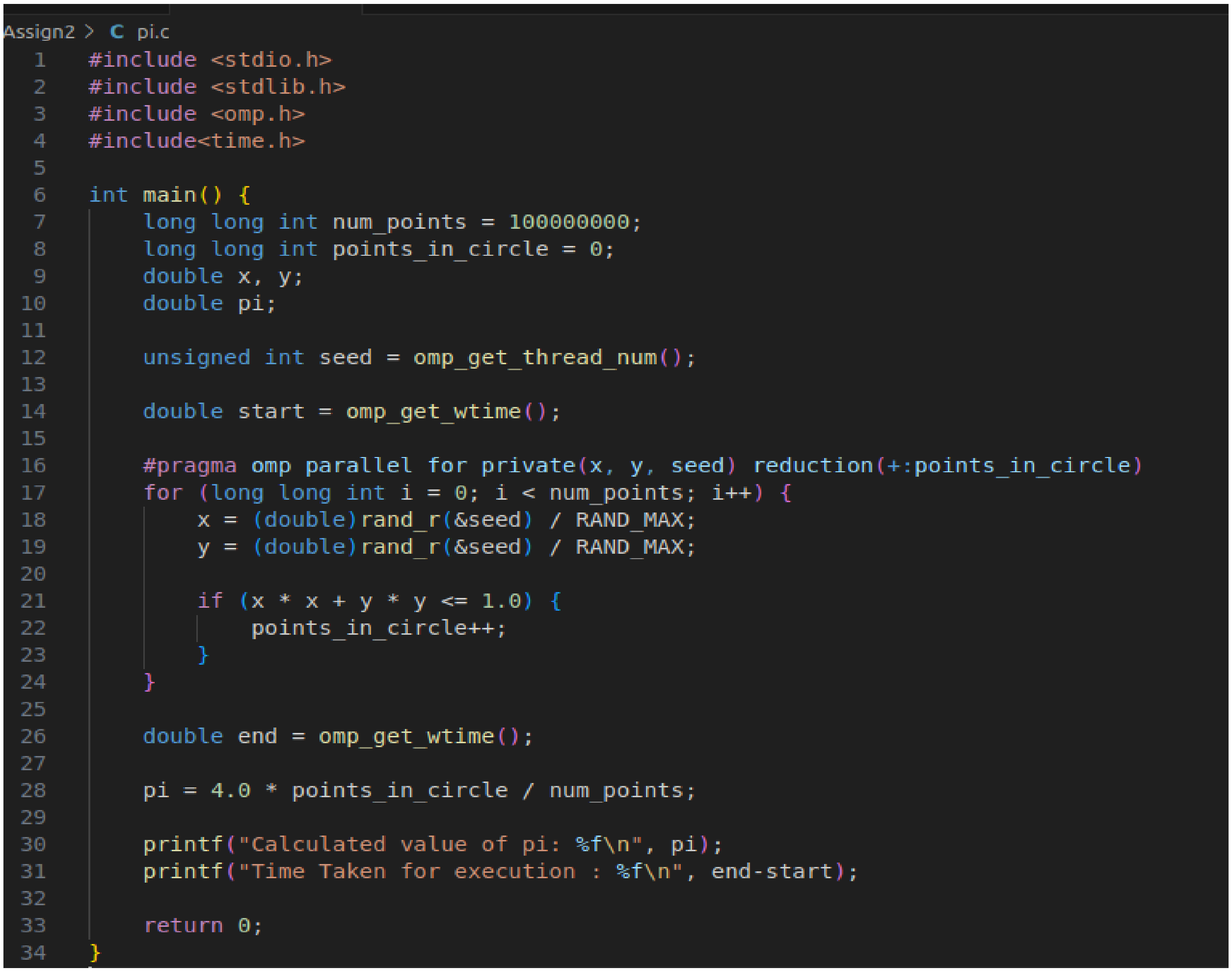


```
wlug@wlug-optiplex:~/Desktop/22520007/Assign2$ ./vsadd
Total Threads : 1, Time Taken : 0.002342 Seconds
Total Threads : 2, Time Taken : 0.001815 Seconds
Total Threads : 4, Time Taken : 0.001074 Seconds
Total Threads : 8, Time Taken : 0.000851 Seconds
wlug@wlug-optiplex:~/Desktop/22520007/Assign2$
```

Information:

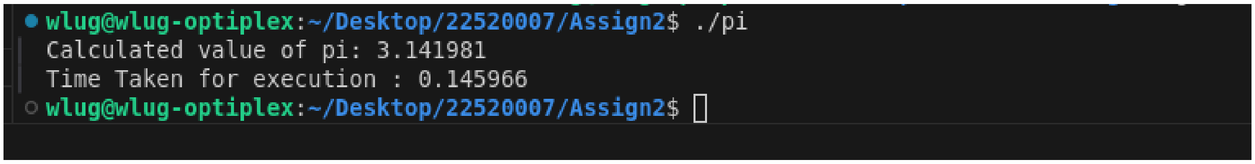
Analysis: As number of threads increase, time taken to perform the operations is reduced

Problem Statement 2: Calculation of value of Pi



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

int main() {
    long long int num_points = 100000000;
    long long int points_in_circle = 0;
    double x, y;
    double pi;

    unsigned int seed = omp_get_thread_num();

    double start = omp_get_wtime();

    #pragma omp parallel for private(x, y, seed) reduction(+:points_in_circle)
    for (long long int i = 0; i < num_points; i++) {
        x = (double)rand_r(&seed) / RAND_MAX;
        y = (double)rand_r(&seed) / RAND_MAX;

        if (x * x + y * y <= 1.0) {
            points_in_circle++;
        }
    }

    double end = omp_get_wtime();

    pi = 4.0 * points_in_circle / num_points;

    printf("Calculated value of pi: %f\n", pi);
    printf("Time Taken for execution : %f\n", end-start);

    return 0;
}
```

Screenshots:



```
wlug@wlug-optiplex:~/Desktop/22520007/Assign2$ ./pi
Calculated value of pi: 3.141981
Time Taken for execution : 0.145966
wlug@wlug-optiplex:~/Desktop/22520007/Assign2$ 
```

Information: Pi calculation can be done by various methods. Here the method used is Monte Carlo method of π calculation. In this method, pi is calculated using random nuber generation.

Formula Used:

$$\pi = 4 * (no\_of\_points\_in\_circle / no\_of\_points\_in\_square)$$

Analysis: As we increase number of points precision goes on increasing. The method relies on probabilistic sampling. As the sample size increases, the estimate of π converges to its true value.