VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

School of Computer Science and Engineering (SCOPE)

M.Tech – CSE

Computer Architecture and Organisation-

MCSE503P

LAB RECORD (FALL SEM 2024)

Prepared By

Jagtap Mahesh (24MCS1017)

Submitted To

Dr. Thanikachalam V

VIT CHENNAI

Vandalur - Kelambakkam Road

Chennai – 600127

TABLE OF CONTENTS

CAO Lab Assignment 1

Name : Mahesh Jagtap
Reg no. : 24MCS1017

Problem statement 1:

BMI and find Thread number and master thread in c using OMP function.

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

int main() {
   int num_students;
   printf("Enter the number of students: ");
   scanf("%d", &num_students);

  float height[num_students], weight[num_students], bmi[num_students];

   for (int i = 0; i < num_students; i++) {
      printf("Enter height (in meters) and weight (in kilograms) for student
%d: ", i + 1);
      scanf("%f %f", &height[i], &weight[i]);
   }

   // Parallel region to calculate BMI for each student
   #pragma omp parallel
   {
      int tid = omp_get_thread_num(); // Get thread number
          int num_threads = omp_get_num_threads(); // Total number of
threads

      #pragma omp for
      for (int i = 0; i < num_students; i++) {
         bmi[i] = weight[i] / (height[i] * height[i]);
```

```
            printf("Thread %d calculated BMI for student %d: %.2f\n", tid, i
+ 1, bmi[i]);
        }

        // Identify the master thread
        if (tid == 0) {
                printf("Master thread (Thread %d) is managing the parallel
execution.\n", tid);
            printf("Total number of threads: %d\n", num_threads);
        }
    }
    return 0;
}
```

```
C:\Users\Jagta\OneDrive\Desktop\VIT SEM1\CAO>a.exe
Enter the number of students: 4
Enter height (in meters) and weight (in kilograms) for student 1: 1.5 45
Enter height (in meters) and weight (in kilograms) for student 2: 1.8 67
Enter height (in meters) and weight (in kilograms) for student 3: 1.65 78
Enter height (in meters) and weight (in kilograms) for student 4: 1.55 37
Thread 2 calculated BMI for student 3: 28.65
Thread 1 calculated BMI for student 2: 20.68
Thread 3 calculated BMI for student 4: 15.40
Thread 0 calculated BMI for student 1: 20.00
Master thread (Thread 0) is managing the parallel execution.
Total number of threads: 8
```

Problem statement 2:

Find sum of 2 arrays and print the result in the third array and find Thread number and master thread in c using OMP function.

```
#include <omp.h>
#include <stdio.h>
int main() {
    int n;
    printf("Enter the number of elements in the arrays: ");
    scanf("%d", &n);
```

```
    int arr1[n], arr2[n], result[n];
    printf("Enter elements of the first array:\n");
    for (int i = 0; i < n; i++) {
        scanf("%d", &arr1[i]);
    }
    printf("Enter elements of the second array:\n");
    for (int i = 0; i < n; i++) {
        scanf("%d", &arr2[i]);
    }
    #pragma omp parallel
    {
        int tid = omp_get_thread_num(); // Get thread number
            int num_threads = omp_get_num_threads(); // Total number of
threads
        #pragma omp for
        for (int i = 0; i < n; i++) {
            result[i] = arr1[i] + arr2[i];
             printf("Thread %d computed element %d: %d + %d = %d\n", tid,
i, arr1[i], arr2[i], result[i]);
        }
        if (tid == 0) {
                printf("Master thread (Thread %d) is managing the parallel
execution.\n", tid);
            printf("Total number of threads: %d\n", num_threads);
                }
    }
    printf("\nResulting array after summing the two arrays:\n");
    for (int i = 0; i < n; i++) {
        printf("result[%d] = %d\n", i, result[i]);
    }
    return 0;
}
```

```
C:\Users\Jagta\OneDrive\Desktop\VIT SEM1\CAO>a.exe
Enter the number of elements in the arrays: 6
Enter elements of the first array:
2 6 7 8 3 5
Enter elements of the second array:
9 3 5 7 2 1
Thread 3 computed element 3: 8 + 7 = 15
Thread 4 computed element 4: 3 + 2 = 5
Thread 1 computed element 1: 6 + 3 = 9
Thread 0 computed element 0: 2 + 9 = 11
Thread 5 computed element 5: 5 + 1 = 6
Thread 2 computed element 2: 7 + 5 = 12
Master thread (Thread 0) is managing the parallel execution.
Total number of threads: 8

Resulting array after summing the two arrays:
result[0] = 11
result[1] = 9
result[2] = 12
result[3] = 15
result[4] = 5
result[5] = 6
```

Problem statement 3:

Print sum of odd numbers and even numbers in an array and find Thread number and master thread in c using OMP function.

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

#define MAX_SIZE 100

int main() {
   int i, size;
   int array[MAX_SIZE];
   int sum_even = 0, sum_odd = 0;

   printf("Enter the size of the array (max %d): ", MAX_SIZE);
   scanf("%d", &size);

   if (size > MAX_SIZE) {
      printf("Size exceeds maximum allowed value of %d.\n", MAX_SIZE);
      return 1;
```

```
}

printf("Enter elements of the array:\n");
for (i = 0; i < size; i++) {
   printf("Element %d: ", i);
   scanf("%d", &array[i]);
}

#pragma omp parallel
{
   // Get the thread ID
   int thread_id = omp_get_thread_num();
   int num_threads = omp_get_num_threads();

   // Print total threads information only once
   #pragma omp master
   {
      printf("Total threads: %d\n", num_threads);
   }

   // Print thread-specific information
   printf("Thread %d: ", thread_id);
   if (thread_id == 0) {
      printf("This is the master thread.\n");
   } else {
      printf("This is not the master thread.\n");
   }

   // Private variables for sums
   int local_sum_even = 0;
   int local_sum_odd = 0;

   #pragma omp for
   for (i = 0; i < size; i++) {
      if (array[i] % 2 == 0) {
         local_sum_even += array[i];
      } else {
         local_sum_odd += array[i];
      }
   }

   #pragma omp critical
   {
      sum_even += local_sum_even;
      sum_odd += local_sum_odd;
   }
}

printf("Sum of even numbers: %d\n", sum_even);
printf("Sum of odd numbers: %d\n", sum_odd);
```

```
    return 0;
}
```

```
C:\Users\Jagta\OneDrive\Desktop\VIT SEM1\CAO>a.exe
Enter the size of the array (max 100): 5
Enter elements of the array:
Element 0: 10
Element 1: 11
Element 2: 46
Element 3: 78
Element 4: 51
Thread 3: This is not the master thread.
Thread 5: This is not the master thread.
Thread 6: This is not the master thread.
Total threads: 8
Thread 0: This is the master thread.
Thread 1: This is not the master thread.
Thread 7: This is not the master thread.
Thread 4: This is not the master thread.
Thread 2: This is not the master thread.
Sum of even numbers: 134
Sum of odd numbers: 62
```

CAO Lab Assignment 2

Name: Mahesh Jagtap

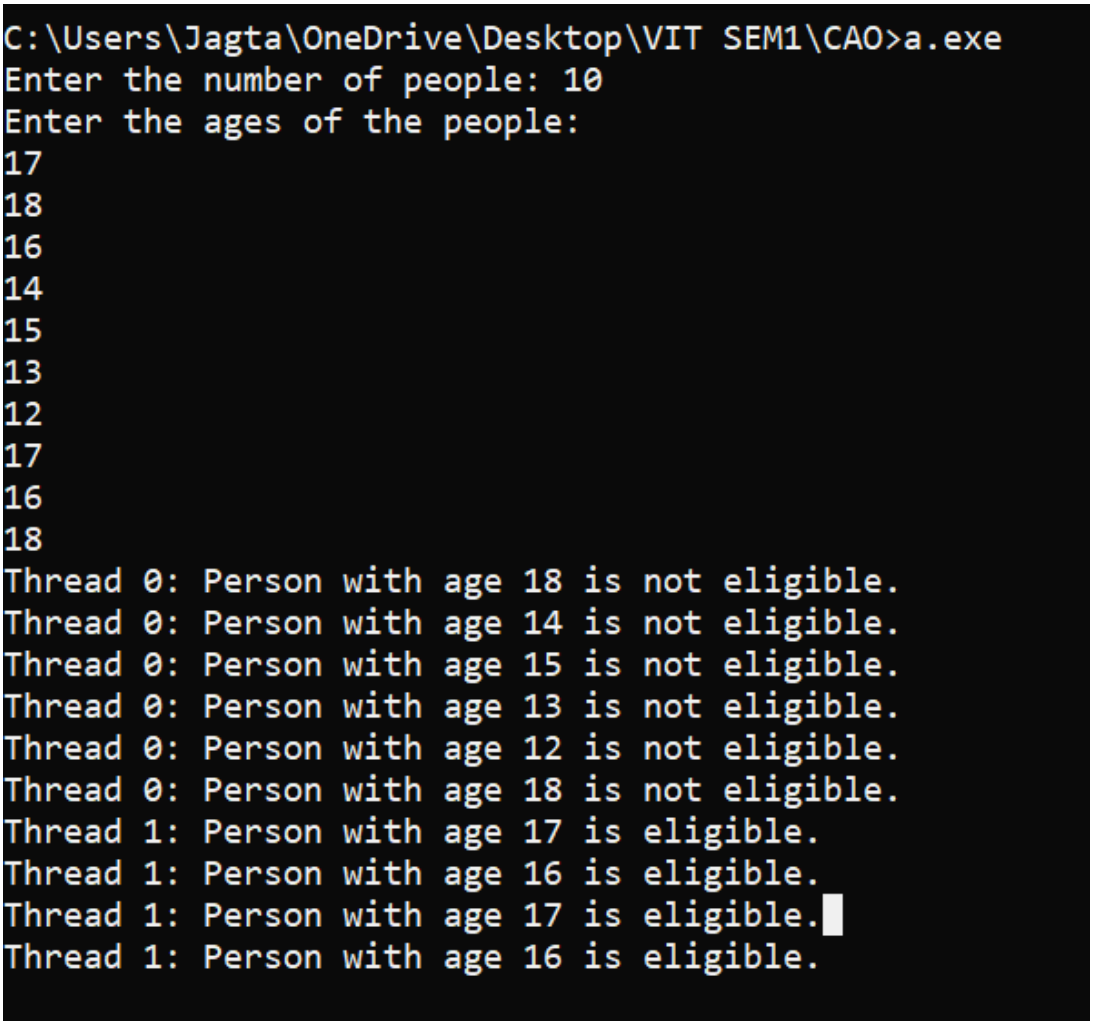
Reg no. 24MCS1017

Problem statement 1:

The election commission has decided to organise a special camp to include young people(age greater than or equal to 16 and less than 18) in electoral role. Help the officials to identify the eligible people. Use thread "1" to print eligible people and thread "0" to not eligible candidate. Get minimum 10 people data.

```
#include <stdio.h>
#include <omp.h>
#define MAX_PEOPLE 100  // Define a maximum number of people
int main() {
   int ages[MAX_PEOPLE];  // Array to store ages
      int eligible[MAX_PEOPLE];  // Array to store eligibility (1 for eligible, 0 for not eligible)
   int N;  // Number of people
   // Get the number of people from the user
   printf("Enter the number of people: ");
   scanf("%d", &N);
   // Ensure N does not exceed the maximum limit
   if (N > MAX_PEOPLE) {
         printf("Number of people exceeds maximum limit of %d.\n", MAX_PEOPLE);
      return 1;
   }
   // Get the ages from the user
   printf("Enter the ages of the people:\n");
```

```
    for (int i = 0; i < N; i++) {
        scanf("%d", &ages[i]);
    }
    // Determine eligibility in the main thread
    for (int i = 0; i < N; i++) {
        eligible[i] = (ages[i] >= 16 && ages[i] < 18) ? 1 : 0;
    }
    // Start parallel region with two threads for printing
    #pragma omp parallel num_threads(2)
    {
        int thread_num = omp_get_thread_num();
        // Thread 0: Print non-eligible people
        if (thread_num == 0) {
            for (int i = 0; i < N; i++) {
                if (eligible[i] == 0) {
                        printf("Thread %d: Person with age %d is not eligible.\n",
thread_num, ages[i]);
                }
            }
        }
        // Thread 1: Print eligible people
        if (thread_num == 1) {
            for (int i = 0; i < N; i++) {
                if (eligible[i] == 1) {
                            printf("Thread %d: Person with age %d is eligible.\n",
thread_num, ages[i]);
                }
        }
        }   }
    return 0;
}
```



Problem statement 2:

For the above election commission using section calculate

i. **The total no. of eligible candidates and**

ii. **Total no. of not eligible candidates**

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

#define MAX_PEOPLE 100  // Define a maximum number of people

int main() {
   int ages[MAX_PEOPLE];  // Array to store ages
```

```
  int eligible[MAX_PEOPLE];  // Array to store eligibility (1 for
     eligible, 0 for not eligible)
int N;  // Number of people

// Get the number of people from the user
printf("Enter the number of people: ");
scanf("%d", &N);

// Ensure N does not exceed the maximum limit
if (N > MAX_PEOPLE) {
       printf("Number of people exceeds maximum limit of %d.\n",
     MAX_PEOPLE);
   return 1;
}

// Get the ages from the user
printf("Enter the ages of the people:\n");
for (int i = 0; i < N; i++) {
   scanf("%d", &ages[i]);
}

// Determine eligibility
for (int i = 0; i < N; i++) {
   eligible[i] = (ages[i] >= 16 && ages[i] < 18) ? 1 : 0;
}

// Initialize counters
int eligible_count = 0;
int not_eligible_count = 0;

// Start parallel region with sections for counting
#pragma omp parallel sections
{
   #pragma omp section
   {
      // Count non-eligible people
      int local_not_eligible_count = 0;
```

```
            for (int i = 0; i < N; i++) {
                if (eligible[i] == 0) {
                    local_not_eligible_count++;
                }
            }
            #pragma omp atomic
            not_eligible_count += local_not_eligible_count;
        }

        #pragma omp section
        {
            // Count eligible people
            int local_eligible_count = 0;
            for (int i = 0; i < N; i++) {
                if (eligible[i] == 1) {
                    local_eligible_count++;
                }
            }
            #pragma omp atomic
            eligible_count += local_eligible_count;
        }
    }

    // Print the results
    printf("Total number of eligible candidates: %d\n", eligible_count);
          printf("Total number of not eligible candidates: %d\n",
         not_eligible_count);

    return 0;
}
```

```
C:\Users\Jagta\OneDrive\Desktop\VIT SEM1\CAO>a.exe
Enter the number of people: 10
Enter the ages of the people:
12
49
16
17
8
16
12
14
17
9
Total number of eligible candidates: 4
Total number of not eligible candidates: 6
```

Combined code:

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

#define MAX_PEOPLE 100  // Define a maximum number of people

int main() {
    int ages[MAX_PEOPLE];  // Array to store ages
    int eligible[MAX_PEOPLE];  // Array to store eligibility (1 for eligible, 0 for not eligible)
    int N;  // Number of people

    // Get the number of people from the user
    printf("Enter the number of people: ");
    scanf("%d", &N);

    // Ensure N does not exceed the maximum limit
    if (N > MAX_PEOPLE) {
        printf("Number of people exceeds maximum limit of %d.\n", MAX_PEOPLE);
        return 1;
```

```
    }

    // Get the ages from the user
    printf("Enter the ages of the people:\n");
    for (int i = 0; i < N; i++) {
        scanf("%d", &ages[i]);
    }

    // Determine eligibility in the main thread
    for (int i = 0; i < N; i++) {
        eligible[i] = (ages[i] >= 16 && ages[i] < 18) ? 1 : 0;
    }


    // Start parallel region with two threads for printing
    #pragma omp parallel num_threads(2)
    {
        int thread_num = omp_get_thread_num();

        // Thread 0: Print non-eligible people
        if (thread_num == 0) {
            for (int i = 0; i < N; i++) {
                if (eligible[i] == 0) {
                    printf("Thread %d: Person with age %d is not eligible.\n",
thread_num, ages[i]);
                }
            }
        }
        // Thread 1: Print eligible people
        if (thread_num == 1) {
            for (int i = 0; i < N; i++) {
                if (eligible[i] == 1) {
                    printf("Thread %d: Person with age %d is eligible.\n", thread_num,
ages[i]);
                }
            }
        }
```

```
    }

     int eligible_count = 0;
      int not_eligible_count = 0;
     #pragma omp parallel sections
    {
       #pragma omp section
       {
          // Count non-eligible people

          for (int i = 0; i < N; i++) {
             if (eligible[i] == 0) {
                not_eligible_count++;
             }
          }
       }

       #pragma omp section
       {
          // Count eligible people
          for (int i = 0; i < N; i++) {
             if (eligible[i] == 1) {
                eligible_count++;
             }
          }
       }
    }
    printf("Total number of eligible candidates: %d\n", eligible_count);
    printf("Total number of not eligible candidates: %d\n", not_eligible_count);
    return 0;
}
```

```
C:\Users\Jagta\OneDrive\Desktop\VIT SEM1\CAO>a.exe
Enter the number of people: 5
Enter the ages of the people:
12
16
17
18
16
Thread 0: Person with age 12 is not eligible.
Thread 0: Person with age 18 is not eligible.
Thread 1: Person with age 16 is eligible.
Thread 1: Person with age 17 is eligible.
Thread 1: Person with age 16 is eligible.
Total number of eligible candidates: 3
Total number of not eligible candidates: 2
```

CAO Lab Assignment 3

Name : Mahesh Jagtap

Reg no. : 24MCS1017

Problem statement:

Consider you have to write a program for the VIT placement cell where 10 students are placed in 4 companies namely, Amazon, Google, Shell, and Intel. Assume no student is offered more than one placement offer. The program has to do the following tasks in parallel and display the result with thread id. Use separate sections to perform each operation

•Get as input the name, register number, the pay package of students selected for jobs in the particular organization

•Display the total number of students selected in each company.

•Display the average pay package of the 10 students

Calculate the execution time of each of the above processes using wtime.

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

#define NUM_STUDENTS 10

// Define a structure for student
typedef struct {
    char name[50];
    int reg_no;
    float pay_package;
    char company[50];
} Student;

// Array to hold student data
Student students[NUM_STUDENTS];

// Function to get student data
```

```
void get_student_data() {
    for (int i = 0; i < NUM_STUDENTS; i++) {
        printf("Enter details for student %d\n", i + 1);
        printf("Name: ");
        scanf("%s", students[i].name);
        printf("Register Number: ");
        scanf("%d", &students[i].reg_no);
        printf("Pay Package: ");
        scanf("%f", &students[i].pay_package);
        printf("Company (Amazon/Google/Shell/Intel): ");
        scanf("%s", students[i].company);
    }
}

// Function to display total students per company
void display_total_students_per_company() {
    int count_amazon = 0, count_google = 0, count_shell = 0, count_intel = 0;

    #pragma omp parallel
    {
        #pragma omp for
        for (int i = 0; i < NUM_STUDENTS; i++) {
            if (strcmp(students[i].company, "Amazon") == 0) {
                #pragma omp atomic
                count_amazon++;
            } else if (strcmp(students[i].company, "Google") == 0) {
                #pragma omp atomic
                count_google++;
            } else if (strcmp(students[i].company, "Shell") == 0) {
                #pragma omp atomic
                count_shell++;
            } else if (strcmp(students[i].company, "Intel") == 0) {
                #pragma omp atomic
                count_intel++;
            }
        }
    }

    printf("Total students per company:\n");
    printf("Amazon: %d\n", count_amazon);
    printf("Google: %d\n", count_google);
    printf("Shell: %d\n", count_shell);
    printf("Intel: %d\n", count_intel);
}
```

```
// Function to display average pay package
void display_average_pay_package() {
    float total_pay = 0.0;

    #pragma omp parallel
    {
        #pragma omp for reduction(+:total_pay)
        for (int i = 0; i < NUM_STUDENTS; i++) {
            total_pay += students[i].pay_package;
        }
    }

    float average_pay = total_pay / NUM_STUDENTS;
    printf("Average pay package: %.2f\n", average_pay);
}

int main() {
    double start_time, end_time;

    start_time = omp_get_wtime();
    get_student_data();
    end_time = omp_get_wtime();
    printf("Time taken for input student data: %f seconds\n", end_time - start_time);

    // Parallel region with sections
    #pragma omp parallel sections
    {
        #pragma omp section
        {
            double section_start_time = omp_get_wtime();
            display_total_students_per_company();
            double section_end_time = omp_get_wtime();
            printf("Time taken for displaying total students per company: %f seconds\n",
section_end_time - section_start_time);
        }

        #pragma omp section
        {
            double section_start_time = omp_get_wtime();
            display_average_pay_package();
            double section_end_time = omp_get_wtime();
            printf("Time taken for calculating average pay package: %f seconds\n",
section_end_time - section_start_time);
```

```
        }
    }

    return 0;
}
```

```
C:\Users\Jagta\OneDrive\Desktop\VIT SEM1\CAO>a.exe
Enter details for student 1
Name: mk
Register Number: 8
Pay Package: 12
Company (Amazon/Google/Shell/Intel): Amazon
Enter details for student 2
Name: kk
Register Number: 2
Pay Package: 23
Company (Amazon/Google/Shell/Intel): Google
Enter details for student 3
Name: we
Register Number: 3
Pay Package: 45
Company (Amazon/Google/Shell/Intel): Shell
Enter details for student 4
Name: jk
Register Number: 890
Pay Package: 43
Company (Amazon/Google/Shell/Intel): Intel
Enter details for student 5
Name: op
Register Number: 80
Pay Package: 22
Company (Amazon/Google/Shell/Intel): Google
```

```
Enter details for student 6
Name: ii
Register Number: 90
Pay Package: 32
Company (Amazon/Google/Shell/Intel): Shell
Enter details for student 7
Name: oo
Register Number: 05
Pay Package: 21
Company (Amazon/Google/Shell/Intel): Amazon
Enter details for student 8
Name: pp
Register Number: 44
Pay Package: 29
Company (Amazon/Google/Shell/Intel): Intel
Enter details for student 9
Name: ooppw
Register Number: 82
Pay Package: 33
Company (Amazon/Google/Shell/Intel): Google
Enter details for student 10
Name: sj
Register Number: 82
Pay Package: 44
Company (Amazon/Google/Shell/Intel): Shell
Time taken for input student data: 120.487000 seconds
Average pay package: 30.40
Time taken for calculating average pay package: 0.000000 seconds
Total students per company:
Amazon: 2
Google: 3
Shell: 3
Intel: 2
Time taken for displaying total students per company: 0.006000 seconds
```

CAO Lab Assignment 4

Name : Mahesh Jagtap

Reg no. : 24MCS1017

Problem statement: A new aeroplane service company announces a new scheme for ticket reservation as its opening ceremony offers. It runs a 100-seat plane with 3 services a day. The booking scheme is as follows:

- ➔ For the first 20 passengers 40% of the original ticket cost;
- ➔ For the next 20 passengers, it provides a 30% discount on the original ticket cost;
- ➔ For the next 30 passengers it gives 25% off the original ticket cost;
- ➔ For the remaining passengers, it gives a 10% off the original ticket cost. Assume that all the tickets are sold for each service.

Write an OpenMP C program to calculate the amount earned by the company in a month. Identify the master thread, display the processor number, and use sections appropriately. For each loop calculate its wall time.

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

#define ORIGINAL_TICKET_COST 1000  // Assume the original ticket cost is 1000
#define SEATS 100
#define SERVICES_PER_DAY 3
#define DAYS_IN_MONTH 30

int main() {
   int i, total_earnings = 0;
   double start_time, end_time;
   omp_set_num_threads(4); // Set the number of threads
   start_time = omp_get_wtime();

   #pragma omp parallel sections
   {
      #pragma omp section
      {
         int earnings = 0;
         for (i = 0; i < 20; i++) {
            earnings += ORIGINAL_TICKET_COST * 0.6; // 40% off
         }
         #pragma omp critical
         total_earnings += earnings;

         printf("Section 1: Master thread: %d, Processor: %d, Earnings: %d\n",
omp_get_thread_num(), omp_get_num_procs(), earnings);
```

```
        }

        #pragma omp section
        {
            int earnings = 0;
            for (i = 20; i < 40; i++) {
                earnings += ORIGINAL_TICKET_COST * 0.7; // 30% off
            }
            #pragma omp critical
            total_earnings += earnings;

            printf("Section 2: Master thread: %d, Processor: %d, Earnings: %d\n",
omp_get_thread_num(), omp_get_num_procs(), earnings);
        }

        #pragma omp section
        {
            int earnings = 0;
            for (i = 40; i < 70; i++) {
                earnings += ORIGINAL_TICKET_COST * 0.75; // 25% off
            }
            #pragma omp critical
            total_earnings += earnings;

            printf("Section 3: Master thread: %d, Processor: %d, Earnings: %d\n",
omp_get_thread_num(), omp_get_num_procs(), earnings);
        }

        #pragma omp section
        {
            int earnings = 0;
            for (i = 70; i < 100; i++) {
                earnings += ORIGINAL_TICKET_COST * 0.9; // 10% off
            }
            #pragma omp critical
            total_earnings += earnings;

            printf("Section 4: Master thread: %d, Processor: %d, Earnings: %d\n",
omp_get_thread_num(), omp_get_num_procs(), earnings);
        }
    }

    end_time = omp_get_wtime();

    total_earnings *= SERVICES_PER_DAY * DAYS_IN_MONTH; // Calculate for a month

    printf("Total earnings for the company in a month: %d\n", total_earnings);
    printf("Time taken for computation: %f seconds\n", end_time - start_time);
```

```
    return 0;
}
```

```
exam1@oslab-VirtualBox:~/Desktop$ gcc -fopenmp lab4.c
exam1@oslab-VirtualBox:~/Desktop$ ./a.out
Section 2: Master thread: 1, Processor: 2, Earnings: 14000
Section 4: Master thread: 1, Processor: 2, Earnings: 27000
Section 3: Master thread: 0, Processor: 2, Earnings: 22500
Section 1: Master thread: 2, Processor: 2, Earnings: 12000
Total earnings for the company in a month: 6795000
Time taken for computation: 0.000276 seconds
exam1@oslab-VirtualBox:~/Desktop$
```

CAO Lab Assignment 5

Name : Mahesh Jagtap

Reg no. : 24MCS1017

Title:Private shared variables

Problem statement:

Design a math application:
It accepts an integer number as input and outputs whether it is a rational number, perfect number, or prime number.

Design a parallel program for the same.

- Use sections for every operation.
- Incorporate `private` for rational number, `LastPrivate` for perfect number, and `FirstPrivate` for prime number.
- Limit the number of threads to 3.

```
#include <stdio.h>
#include <math.h>
#include <omp.h>
int isRational(int n) {
   return (n > 0);
}

int isPerfect(int n) {
int sum = 0;
   for(int i = 1; i < n; i++){
          if(n% i == 0)
         sum = sum + i;
   }
   if(sum == n)
      return 1;
   else
      return 0;

}

int isPrime(int n) {
  int  i, flag = 0;
```

```
 if (n <=1)
        flag = 1;

 for (i = 2; i <= n / 2; ++i) {
        if (n % i == 0) {
                flag = 1;
                break;
        }
 }

 if (flag == 0)
        return 1;
 else
        return 0;
}



int main() {
   int number;
   printf("Enter an integer number: ");
   scanf("%d", &number);

   int rational, perfect, prime;

   omp_set_num_threads(3);

   #pragma omp parallel sections private(rational) lastprivate(perfect)
firstprivate(prime)
   {
      #pragma omp section
      {
         rational = isRational(number);
         if (rational)
            printf("The number %d is rational.\n", number);
         else
                printf("The number %d is not a rational.\n", number);

      }

      #pragma omp section
      {
         perfect = isPerfect(number);
         if (perfect)
```

```
            printf("The number %d is a perfect number.\n", number);
            else
                printf("The number %d is not a perfect number.\n", number);
        }

        #pragma omp section
        {
          prime = isPrime(number);
          if (prime)
            printf("The number %d is a prime number.\n", number);
            else
                printf("The number %d is not a prime number.\n", number);
        }
      }

      return 0;
    }
```

```
exam1@oslab-VirtualBox:~$ gcc -fopenmp CaoLab5.c
exam1@oslab-VirtualBox:~$ ./a.out
Enter an integer number: 28
The number 28 is rational.
The number 28 is a perfect number.
The number 28 is not a prime number.
exam1@oslab-VirtualBox:~$ ./a.out
Enter an integer number: 17
The number 17 is rational.
The number 17 is not a perfect number.
The number 17 is a prime number.
```

```
exam1@oslab-VirtualBox:~$ ./a.out
Enter an integer number: -89
The number -89 is not a rational.
The number -89 is not a perfect number.
The number -89 is not a prime number.
exam1@oslab-VirtualBox:~$
```

CAO Lab Assignment 6

Name : Mahesh Jagtap

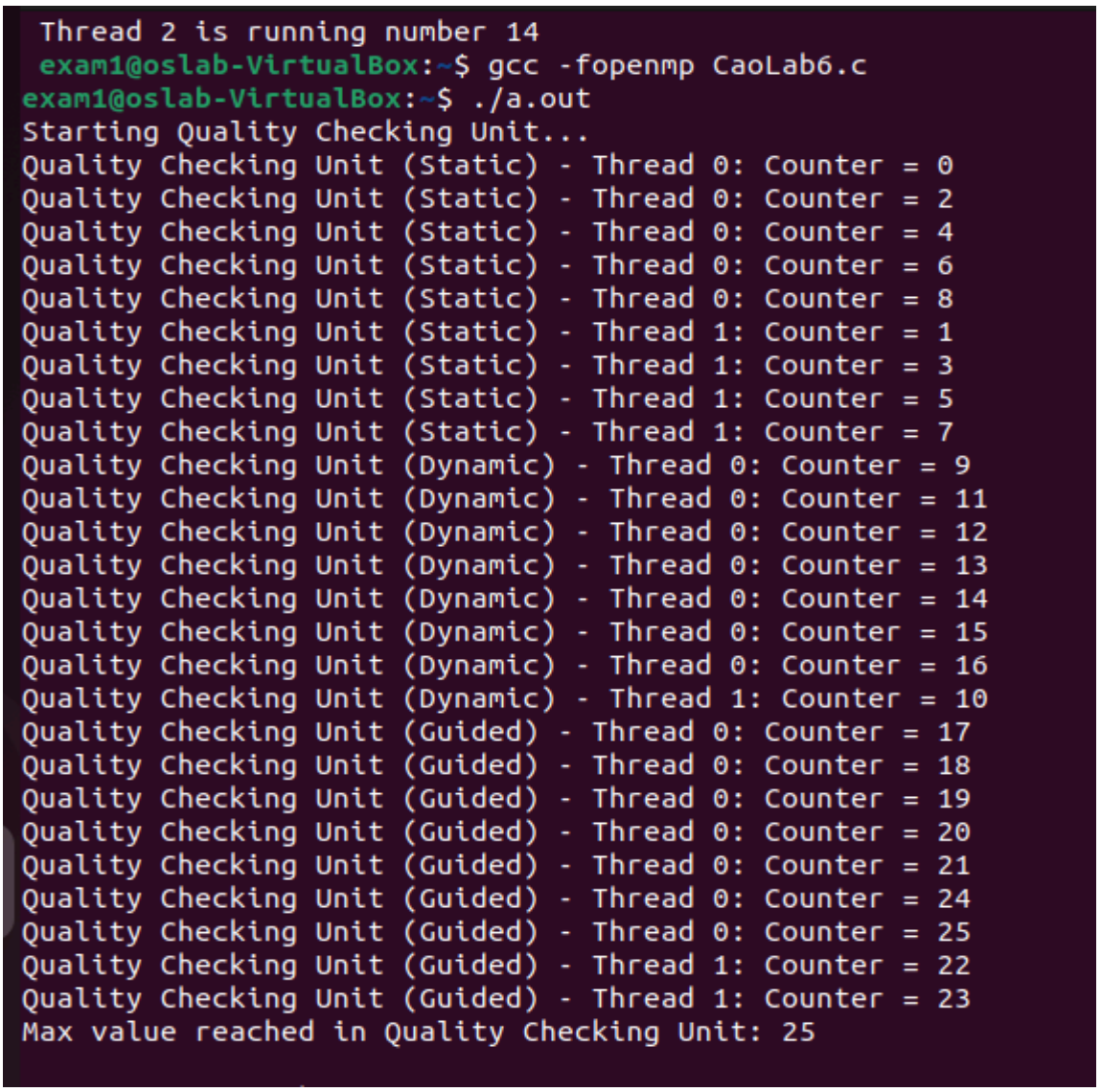
Reg no. : 24MCS1017

Problem statement:

The quality checking unit in the toy modeling unit has an incremental counter and counts the tested toy from 0 to 25 . Once the counter reaches the max value all tested toys will be transferred to the dispatching unit in which this counter decrements from the maximum of 25 and reaches to zero. Use last private to get max value. Write an OpenMp program to perform the above scenario using all 3 scheduling concepts.

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

#define MAX_COUNT 25

void quality_checking_unit() {
	int counter = 0;

	#pragma omp parallel for schedule(static, 1) lastprivate(counter)
	for (int i = 0; i <= MAX_COUNT / 3; i++) {
	counter = i;
	printf("Quality Checking Unit (Static) - Thread %d: Counter = %d\n",
omp_get_thread_num(), counter);
	}

	#pragma omp parallel for schedule(dynamic, 1) lastprivate(counter)
	for (int i = MAX_COUNT / 3 + 1; i <= 2 * MAX_COUNT / 3; i++) {
	counter = i;
	printf("Quality Checking Unit (Dynamic) - Thread %d: Counter = %d\n",
omp_get_thread_num(), counter);
	}

	#pragma omp parallel for schedule(guided, 1) lastprivate(counter)
	for (int i = 2 * MAX_COUNT / 3 + 1; i <= MAX_COUNT; i++) {
	counter = i;
	printf("Quality Checking Unit (Guided) - Thread %d: Counter = %d\n",
omp_get_thread_num(), counter);
	}

	printf("Max value reached in Quality Checking Unit: %d\n", counter);
}

void dispatching_unit() {
```

```
    int counter = MAX_COUNT;

    #pragma omp parallel for schedule(static, 1) lastprivate(counter)
    for (int i = MAX_COUNT; i >= 2 * MAX_COUNT / 3 + 1; i--) {
    counter = i;
    printf("Dispatching Unit (Static) - Thread %d: Counter = %d\n", omp_get_thread_num(),
counter);
    }


    #pragma omp parallel for schedule(dynamic, 1) lastprivate(counter)
    for (int i = 2 * MAX_COUNT / 3; i >= MAX_COUNT / 3 + 1; i--) {
    counter = i;
    printf("Dispatching Unit (Dynamic) - Thread %d: Counter = %d\n", omp_get_thread_num(),
counter);
    }

    #pragma omp parallel for schedule(guided, 1) lastprivate(counter)
    for (int i = MAX_COUNT / 3; i >= 0; i--) {
    counter = i;
    printf("Dispatching Unit (Guided) - Thread %d: Counter = %d\n", omp_get_thread_num(),
counter);
    }

    printf("Min value reached in Dispatching Unit: %d\n", counter);
}

int main() {
    printf("Starting Quality Checking Unit...\n");
    quality_checking_unit();

    printf("\nStarting Dispatching Unit...\n");
    dispatching_unit();

    return 0;
}
```


Thread 2 is running number 14
exam1@oslab-VirtualBox:~$ gcc -fopenmp CaoLab6.c
exam1@oslab-VirtualBox:~$ ./a.out
Starting Quality Checking Unit...
Quality Checking Unit (Static) - Thread 0: Counter = 0
Quality Checking Unit (Static) - Thread 0: Counter = 2
Quality Checking Unit (Static) - Thread 0: Counter = 4
Quality Checking Unit (Static) - Thread 0: Counter = 6
Quality Checking Unit (Static) - Thread 0: Counter = 8
Quality Checking Unit (Static) - Thread 1: Counter = 1
Quality Checking Unit (Static) - Thread 1: Counter = 3
Quality Checking Unit (Static) - Thread 1: Counter = 5
Quality Checking Unit (Static) - Thread 1: Counter = 7
Quality Checking Unit (Dynamic) - Thread 0: Counter = 9
Quality Checking Unit (Dynamic) - Thread 0: Counter = 11
Quality Checking Unit (Dynamic) - Thread 0: Counter = 12
Quality Checking Unit (Dynamic) - Thread 0: Counter = 13
Quality Checking Unit (Dynamic) - Thread 0: Counter = 14
Quality Checking Unit (Dynamic) - Thread 0: Counter = 15
Quality Checking Unit (Dynamic) - Thread 0: Counter = 16
Quality Checking Unit (Dynamic) - Thread 1: Counter = 10
Quality Checking Unit (Guided) - Thread 0: Counter = 17
Quality Checking Unit (Guided) - Thread 0: Counter = 18
Quality Checking Unit (Guided) - Thread 0: Counter = 19
Quality Checking Unit (Guided) - Thread 0: Counter = 20
Quality Checking Unit (Guided) - Thread 0: Counter = 21
Quality Checking Unit (Guided) - Thread 0: Counter = 24
Quality Checking Unit (Guided) - Thread 0: Counter = 25
Quality Checking Unit (Guided) - Thread 1: Counter = 22
Quality Checking Unit (Guided) - Thread 1: Counter = 23
Max value reached in Quality Checking Unit: 25

```
 Thread 2 is running number 14
 exam1@oslab-VirtualBox:~$ gcc -fopenmp CaoLab6.c
exam1@oslab-VirtualBox:~$ ./a.out
Starting Quality Checking Unit...
Quality Checking Unit (Static) - Thread 0: Counter = 0
Quality Checking Unit (Static) - Thread 0: Counter = 2
Quality Checking Unit (Static) - Thread 0: Counter = 4
Quality Checking Unit (Static) - Thread 0: Counter = 6
Quality Checking Unit (Static) - Thread 0: Counter = 8
Quality Checking Unit (Static) - Thread 1: Counter = 1
Quality Checking Unit (Static) - Thread 1: Counter = 3
Quality Checking Unit (Static) - Thread 1: Counter = 5
Quality Checking Unit (Static) - Thread 1: Counter = 7
Quality Checking Unit (Dynamic) - Thread 0: Counter = 9
Quality Checking Unit (Dynamic) - Thread 0: Counter = 11
Quality Checking Unit (Dynamic) - Thread 0: Counter = 12
Quality Checking Unit (Dynamic) - Thread 0: Counter = 13
Quality Checking Unit (Dynamic) - Thread 0: Counter = 14
Quality Checking Unit (Dynamic) - Thread 0: Counter = 15
Quality Checking Unit (Dynamic) - Thread 0: Counter = 16
Quality Checking Unit (Dynamic) - Thread 1: Counter = 10
Quality Checking Unit (Guided) - Thread 0: Counter = 17
Quality Checking Unit (Guided) - Thread 0: Counter = 18
Quality Checking Unit (Guided) - Thread 0: Counter = 19
Quality Checking Unit (Guided) - Thread 0: Counter = 20
Quality Checking Unit (Guided) - Thread 0: Counter = 21
Quality Checking Unit (Guided) - Thread 0: Counter = 24
Quality Checking Unit (Guided) - Thread 0: Counter = 25
Quality Checking Unit (Guided) - Thread 1: Counter = 22
Quality Checking Unit (Guided) - Thread 1: Counter = 23
Max value reached in Quality Checking Unit: 25
```

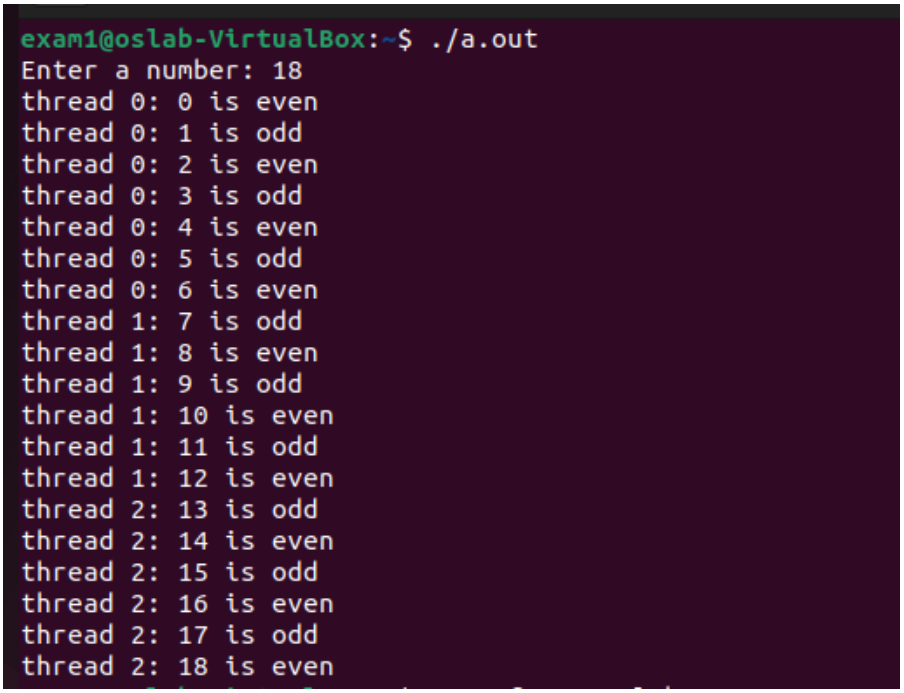
CAO Lab Assignment 7

Name : Mahesh Jagtap

Reg no. : 24MCS1017

A] Design a parallel program to print ‘n’ even and odd numbers in sequential fashion of threads.use ordered.

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

int main() {
        // Set the number of threads
        omp_set_num_threads(3);
        int value;
        printf("Enter a number: ");
        scanf("%d", &value);

        #pragma omp parallel for ordered
        for (int j = 0; j <=value; j++) {
        #pragma omp ordered
        if (j % 2 == 0) {
        printf("thread %d: %d is even \n", omp_get_thread_num(), j);
        } else {
        printf("thread %d: %d is odd \n", omp_get_thread_num(), j);
        }
        }

        return 0;
}
```



B] A contest is being held for TechnoVIT. Students can register, if they want, they can unregister. Registered students (registration numbers:9,3,2…) are stored in an array. Only one student can register or unregister at a time. But they can view the registered list without any constraint. Design a parallel program with the help of locks. Use ordered in any part of the code.

Code:

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


#define MAX_STUDENTS 100


// Global variables
int registered[MAX_STUDENTS]; // Array to store registered students
int num_registered = 0; // Number of registered students
omp_lock_t lock; // Lock for synchronization


// Function to register a student
void registerStudent(int student_id) {
    omp_set_lock(&lock); // Acquire the lock to perform registration
    registered[num_registered] = student_id;
    num_registered++;
```

```
    printf("Student %d registered.\n", student_id);
    omp_unset_lock(&lock); // Release the lock
}

// Function to unregister a student
void unregisterStudent(int student_id) {
    omp_set_lock(&lock); // Acquire the lock to perform unregistration
    int found = 0;
    for (int i = 0; i < num_registered; i++) {
        if (registered[i] == student_id) {
            // Remove the student from the list by shifting elements
            for (int j = i; j < num_registered - 1; j++) {
                registered[j] = registered[j + 1];
            }
            num_registered--;
            found = 1;
            printf("Student %d unregistered.\n", student_id);
            break;
        }
    }
    if (!found) {
        printf("Student %d not found in the registered list.\n", student_id);
    }
    omp_unset_lock(&lock); // Release the lock
}

// Function to display the registered list
void displayRegisteredList() {
    omp_set_lock(&lock); // Acquire the lock to display the list
    printf("Registered students: ");
    for (int i = 0; i < num_registered; i++) {
        printf("%d ", registered[i]);
    }
    printf("\n");
    omp_unset_lock(&lock); // Release the lock
}

int main() {
    int choice, student_id;

    omp_init_lock(&lock); // Initialize the lock

    while (1) {
```

```
        printf("Enter your choice:\n");
        printf("1. Register\n");
        printf("2. Unregister\n");
        printf("3. Display registered list\n");
        printf("4. Exit\n");
        scanf("%d", &choice);


        if (choice == 1) {
            printf("Enter student ID to register: ");
            scanf("%d", &student_id);
            registerStudent(student_id);
        } else if (choice == 2) {
            printf("Enter student ID to unregister: ");
            scanf("%d", &student_id);
            unregisterStudent(student_id);
        } else if (choice == 3) {
            displayRegisteredList();
        } else if (choice == 4) {
            break;
        } else {
            printf("Invalid choice. Try again.\n");
        }
    }


    omp_destroy_lock(&lock); // Destroy the lock


    return 0;
}
```

```
exam1@oslab-VirtualBox:~$ gcc -fopenmp lab7b.c
exam1@oslab-VirtualBox:~$ ./a.out
Enter your choice:
1. Register
2. Unregister
3. Display registered list
4. Exit
1
Enter student ID to register: 234
Student 234 registered.
Enter your choice:
1. Register
2. Unregister
3. Display registered list
4. Exit
1
Enter student ID to register: 45
Student 45 registered.
Enter your choice:
1. Register
2. Unregister
3. Display registered list
4. Exit
```

```
Registered students: 234 45
Enter your choice:
1. Register
2. Unregister
3. Display registered list
4. Exit
2
Enter student ID to unregister: 37
Student 37 not found in the registered list.
Enter your choice:
1. Register
2. Unregister
3. Display registered list
4. Exit
2
Enter student ID to unregister: 234
Student 234 unregistered.
Enter your choice:
1. Register
2. Unregister
3. Display registered list
4. Exit
3
Registered students: 45
```

CAO Lab Assignment 8

Name : Mahesh Jagtap
Reg no. : 24MCS1017

Title: Barrier_Series

Problem statement:

Write a parallel program using OpenMP to implement the following series,
1/2 + 1/4 + 1/8 + ...
Find the sum of the series and print it along with the thread id and the last value in the series for the given "N" value.

Incorporate barrier, scheduling, ordered constructs of OpenMP.

Print the output in a file "series.txt"

```
#include <stdio.h>
#include <omp.h>
int main()
{
   int N;
   double sum = 0.0, term;
   FILE *fptr;

   // Prompt user for the number of terms
   printf("Enter the number of terms (N): ");
   scanf("%d", &N);

   // Open file for writing output
   fptr = fopen("series.txt", "w");
   if (fptr == NULL)
   {
      printf("Error opening file!\n");
      return 1;
   }
#pragma omp parallel shared(sum) private(term)
   {
      int tid = omp_get_thread_num();

#pragma omp for schedule(static) reduction(+ : sum) ordered
      for (int i = 1; i <= N; i++)
      {
         // Calculate the term of the series
```

```
            term = 1.0 / (1 << i); // Equivalent to 1.0 / (2^i)

        #pragma omp ordered
                {
                   // Print thread info, term, and partial sum to file and console in the order of
        execution
                   printf("Thread %d - Term %d: %f\n", tid, i, term);
                   fprintf(fptr, "Thread %d - Term %d: %f\n", tid, i, term);
                }

                sum += term;
            }
        // Ensure all threads have completed their updates to `sum`
        #pragma omp barrier

        // Only one thread prints the final output for the sum and last term
        #pragma omp single
            {
                printf("Sum of series: %f\n", sum);
                printf("Last term in series: %f\n", term);

                fprintf(fptr, "Sum of series: %f\n", sum);
                fprintf(fptr, "Last term in series: %f\n", term);
            }
        }
        // Close the file
        fclose(fptr);

        printf("Results written to series.txt\n");
        return 0;
    }
```

Output:

```
C:\Users\Jagta\OneDrive\Desktop\VIT SEM1\CAO>a.exe
Enter the number of terms (N): 5
Thread 0 - Term 1: 0.500000
Thread 1 - Term 2: 0.250000
Thread 2 - Term 3: 0.125000
Thread 3 - Term 4: 0.062500
Thread 4 - Term 5: 0.031250
Sum of series: 0.968750
Last term in series: 0.031250
Results written to series.txt
```

CAO Lab Assignment 9

Name : Mahesh Jagtap

Reg no. : 24MCS1017

Title: Matrix Operation

Problem statement:

Write a parallel program using OpenMP to Matrix-Vector multiplication

Code:

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

#define ROWS 2   // Number of rows in the matrix
#define COLS 3   // Number of columns in the matrix (and size of the vector)

int main() {
   int matrix[ROWS][COLS], vector[COLS], result[ROWS] = {0};

   // Taking matrix input from the user
   printf("Enter elements of %dx%d matrix:\n", ROWS, COLS);
   for (int i = 0; i < ROWS; i++) {
      for (int j = 0; j < COLS; j++) {
         printf("Matrix[%d][%d]: ", i, j);
         scanf("%d", &matrix[i][j]);
      }
   }

   // Taking vector input from the user
   printf("Enter elements of vector of size %d:\n", COLS);
   for (int i = 0; i < COLS; i++) {
      printf("Vector[%d]: ", i);
      scanf("%d", &vector[i]);
   }

   // Start time measurement
   double start_time = omp_get_wtime();

   // Matrix-vector multiplication in parallel
   #pragma omp parallel for
   for (int i = 0; i < ROWS; i++) {
      for (int j = 0; j < COLS; j++) {
         result[i] += matrix[i][j] * vector[j];
      }
   }

   // End time measurement
   double end_time = omp_get_wtime();
```

```
    // Display the result
    printf("Resultant vector:\n");
    for (int i = 0; i < ROWS; i++) {
        printf("%d\n", result[i]);
    }

    // Display the time taken
    printf("Time taken for matrix-vector multiplication: %f seconds\n", end_time - start_time);

    return 0;
}
```

Output:

```
C:\Users\Jagta\OneDrive\Desktop\VIT SEM1\CAO>a.exe
Enter elements of 2x3 matrix:
Matrix[0][0]: 4
Matrix[0][1]: 9
Matrix[0][2]: 7
Matrix[1][0]: 5
Matrix[1][1]: 8
Matrix[1][2]: 7
Enter elements of vector of size 3:
Vector[0]: 3
Vector[1]: 2
Vector[2]: 1
Resultant vector:
37
38
Time taken for matrix-vector multiplication: 0.001000 seconds
```

CAO Lab Assignment 10

Name : Mahesh Jagtap

Reg no. : 24MCS1017

Problem statement:

Write a parallel program using OpenMP to Matrix multiplication

Code:

```
#include <stdio.h>
#include <omp.h>
#define N 3  // Size of the matrices (N x N)
int main() {
   int A[N][N], B[N][N], C[N][N] = {0};
   // Taking input for Matrix A
   printf("Enter elements of %dx%d matrix A:\n", N, N);
   for (int i = 0; i < N; i++) {
      for (int j = 0; j < N; j++) {
         printf("A[%d][%d]: ", i, j);
         scanf("%d", &A[i][j]);
      }
   }
   // Taking input for Matrix B

   printf("Enter elements of %dx%d matrix B:\n", N, N);

   for (int i = 0; i < N; i++) {
      for (int j = 0; j < N; j++) {
         printf("B[%d][%d]: ", i, j);

         scanf("%d", &B[i][j]);

      }

   }

   // Start time measurement
   double start_time = omp_get_wtime();

   // Matrix multiplication in parallel
   #pragma omp parallel for
   for (int i = 0; i < N; i++) {
      for (int j = 0; j < N; j++) {
         C[i][j] = 0;
         for (int k = 0; k < N; k++) {
            C[i][j] += A[i][k] * B[k][j];
```

```
            }
        }
    }

    // End time measurement
    double end_time = omp_get_wtime();

    // Display the result
    printf("Resultant matrix C:\n");
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            printf("%d ", C[i][j]);
        }
        printf("\n");
    }


    // Display the time taken
    printf("Time taken for matrix multiplication: %f seconds\n", end_time - start_time);

    return 0;
}
```

Output:

```
C:\Users\Jagta\OneDrive\Desktop\VIT SEM1\CAO>a.exe
Enter elements of 3x3 matrix A:
A[0][0]: 1
A[0][1]: 4
A[0][2]: 5
A[1][0]: 2
A[1][1]: 3
A[1][2]: 4
A[2][0]: 8
A[2][1]: 7
A[2][2]: 9
Enter elements of 3x3 matrix B:
B[0][0]: 5
B[0][1]: 4
B[0][2]: 7
B[1][0]: 6
B[1][1]: 3
B[1][2]: 1
B[2][0]: 0
B[2][1]: 0
B[2][2]: 3
Resultant matrix C:
29 16 26
28 17 29
82 53 90
Time taken for matrix multiplication: 0.002000 seconds
```

CAO Lab Assignment 11

Name : Mahesh Jagtap

Reg no. : 24MCS1017

Title: Quick_Sort

Question:

Develop a program to analyse the parallel quick sort .

Code:

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

#define N 10  // Size of the array

// Function to swap two elements
void swap(int *a, int *b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}


// Partition function for QuickSort
int partition(int arr[], int low, int high) {
    int pivot = arr[high];  // Pivot element
    int i = low - 1;

    for (int j = low; j < high; j++) {
        if (arr[j] <= pivot) {
            i++;
            swap(&arr[i], &arr[j]);
        }
    }
    swap(&arr[i + 1], &arr[high]);
    return i + 1;
}

// Parallel QuickSort function
void quicksort(int arr[], int low, int high) {
    if (low < high) {
        // Partition the array
        int pi = partition(arr, low, high);

        // Parallelize recursive calls
```

```
        #pragma omp parallel sections
        {
            #pragma omp section
            quicksort(arr, low, pi - 1);

            #pragma omp section
            quicksort(arr, pi + 1, high);
        }
    }
}

int main() {
    int arr[N];
    // Taking array input from the user
    printf("Enter %d elements for sorting:\n", N);
    for (int i = 0; i < N; i++) {
        printf("Element[%d]: ", i);
        scanf("%d", &arr[i]);
    }
    // Start time measurement
    double start_time = omp_get_wtime();

    // Call parallel QuickSort
    #pragma omp parallel
    {
        #pragma omp single
        quicksort(arr, 0, N - 1);
    }

    // End time measurement
    double end_time = omp_get_wtime();

    // Display sorted array
    printf("Sorted array:\n");
    for (int i = 0; i < N; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");

    // Display the time taken
    printf("Time taken for parallel QuickSort: %f seconds\n", end_time - start_time);

    return 0;
}
```

Output:

```
C:\Users\Jagta\OneDrive\Desktop\VIT SEM1\CAO>gcc -fopenmp assi11.c

C:\Users\Jagta\OneDrive\Desktop\VIT SEM1\CAO>a.exe
Enter 10 elements for sorting:
Element[0]: 4
Element[1]: 7
Element[2]: 9
Element[3]: 43
Element[4]: 2
Element[5]: 8
Element[6]: 31
Element[7]: 9
Element[8]: 32
Element[9]: 83
Sorted array:
2 4 7 8 9 9 31 32 43 83
Time taken for parallel QuickSort: 0.003000 seconds
```