

**School of Computer Science and Engineering (SCOPE)**

**M.Tech –CSE**, **AI & ML, Data Analytics**

**Computer Architecture and Organisation- MCSE503L**

**LAB RECORD**

**Prepared By**

**Selvakumar G – 22MAI1004**

**Submitted To**

**Dr. Maheswari R**

**VIT CHENNAI**

**Vandalur - Kelambakkam Road**

**Chennai – 600127**

T**ABLE OF CONTENTS**

|  |  |  |  |
| --- | --- | --- | --- |
| S. no | Date | Title of the Experiment | Page |
| 1. |  |  |  |
| 2 |  |  |  |
| 3 |  |  |  |
| 4 |  |  |  |
| 5 |  |  |  |
| 6 |  |  |  |
| 7 |  |  |  |
| 8 |  |  |  |
| 9 |  |  |  |
| 10 |  |  |  |
| 11 |  |  |  |
| 12 |  |  |  |

**EXPERIMENT 1**

**OMP program to print thread number**

**AIM:**

To write a OMP program to print the current thread number.

**PROGRAM:**

#include<stdio.h>

#include<omp.h>

void main()

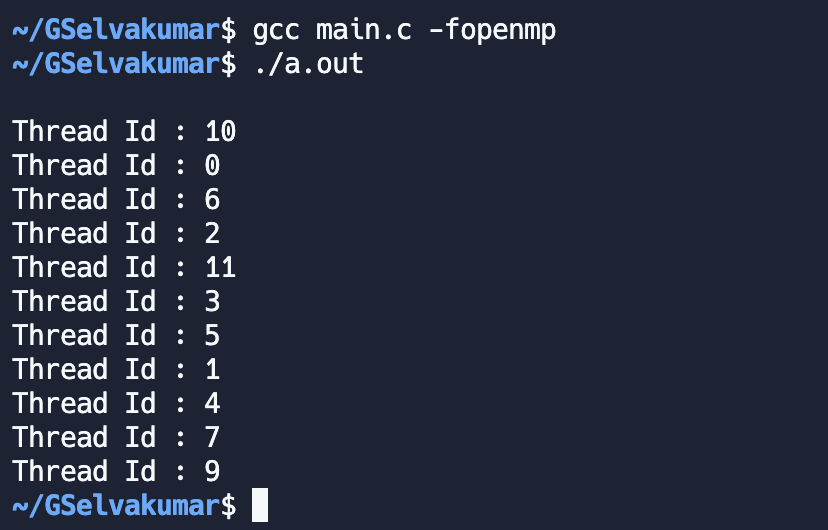
{

#pragma omp parallel

printf("\nThread Id : %d", omp\_get\_thread\_num());

}

**OUTPUT**

****

**EXPERIMENT 2**

**OMP program to print thread count**

**AIM:**

To write a OMP program to print the total number of threads processing the parallel region.

**PROGRAM:**

#include<stdio.h>

#include<omp.h>

void main()

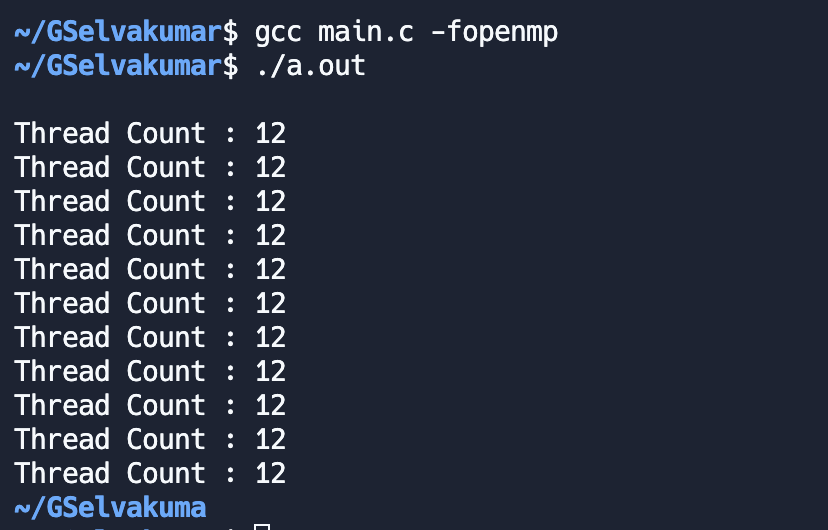
{

#pragma omp parallel

printf("\nThread Count : %d", omp\_get\_num\_threads());

}

**OUTPUT**



**EXPERIMENT 3**

**OMP program to calculate BMI**

**AIM:**

To write a OMP program to calculate BMI of the user based on his height and weight.

**PROGRAM:**

#include <stdio.h>

#include <omp.h>

void main()

{

int ht, wt;

printf("\nEnter the height : ");

scanf("%d", &ht);

printf("\nEnter the weight : ");

scanf("%d", &wt);

#pragma omp parallel

{

int tid = omp\_get\_thread\_num();

int bmi = (wt \* 703) / (ht \* ht);

printf("\nThread Id : %d Your BMI is : %d", tid, bmi);

if(tid == 0)

{

printf("\nNumber of Processors : %d", omp\_get\_num\_procs());

}

}

}

**OUTPUT**

Text

Description automatically generated

**EXPERIMENT 4**

**OMP program to find the sum of array**

**AIM:**

To write a OMP program to calculate the sum of the elements of the two input arrays.

**PROGRAM:**

#include <omp.h>

#include <stdio.h>

int main()

{

   int sum[10],a[10],b[10];

   for(int i=0;i<10;i++)

   {

       a[i]=i;

       b[i]=i;

   }

   #pragma open parallel for

   for(int i=0;i<9;i++)

   {

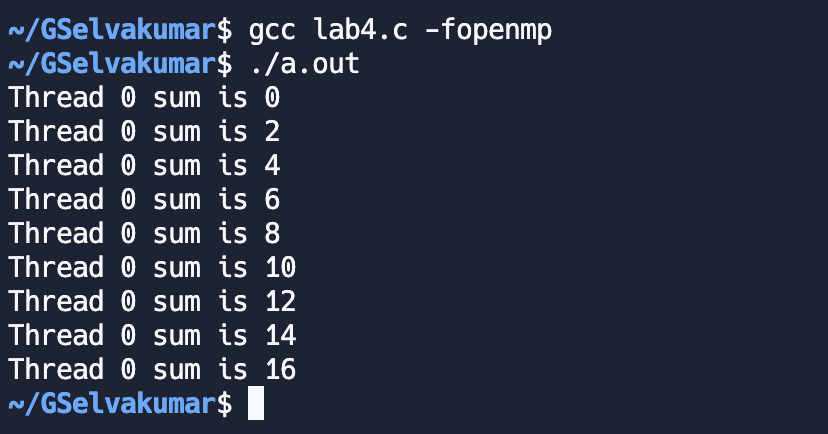
       sum[i]=a[i]+b[i];

       printf("Thread %d sum is %d\n",omp\_get\_thread\_num(),sum[i]);

   }

}

**OUTPUT**



**EXPERIMENT 5**

**OMP program to calculate Simple Interest**

**AIM:**

To write a OMP program to calculate Simple Interest for the given principal amount, interest rate and the period of investment.

**PROGRAM:**

#include <stdio.h>

#include <omp.h>

void main()

{

int p, r, t;

printf("\nEnter Principle Amount : ");

scanf("%d", &p);

printf("\nEnter Interest rate : ");

scanf("%d", &r);

printf("\nEnter the time period : ");

scanf("%d", &t);

#pragma omp parallel

{

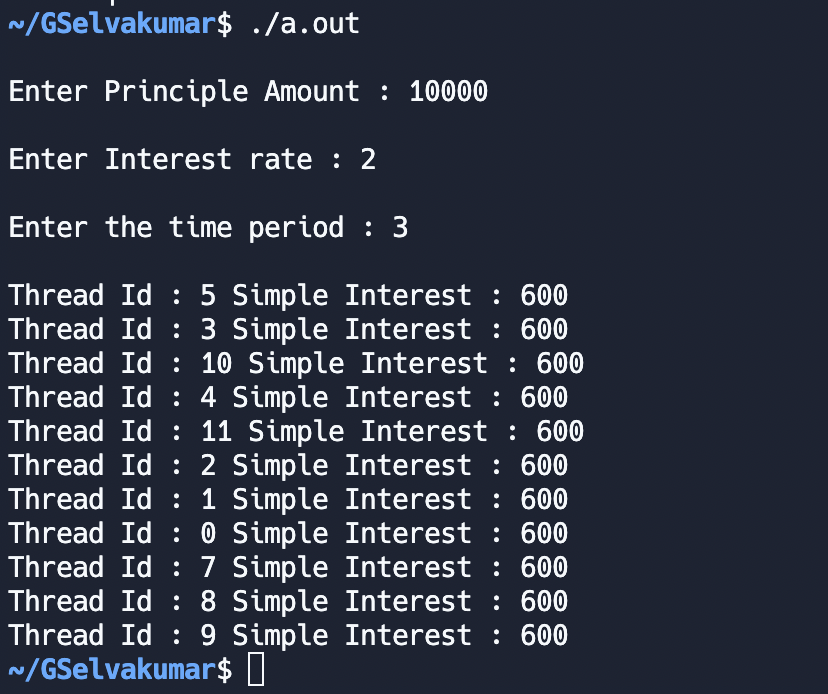
int si = (p \* r \* t) / 100;

printf("\nThread Id : %d Simple Interest : %d", omp\_get\_thread\_num(), si);

}

}

**OUTPUT**



**EXPERIMENT 6**

**OMP program to Perform Arithmetic Operations using Sections**

**AIM:**

To write a OMP program to explore the Sections by perform arithmetic operation such as add, multiply using omp section construct.

**PROGRAM:**

#include <stdio.h>

#include <omp.h>

void main()

{

int n = 2;

int a[n], b[n], c[n];

for (int i = 0; i < n; i++)

{

a[i] = 2 \* i;

b[i] = 10 + i;

}

#pragma omp parallel sections

{

#pragma omp section

{

for (int i = 0; i < n; i++)

{

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

printf("\nThread Id : %d Add : %d", omp\_get\_thread\_num(), c[i]);

}

}

#pragma omp section

{

for (int i = 0; i < n; i++)

{

c[i] = a[i] \* b[i];

printf("\nThread Id : %d Mul : %d", omp\_get\_thread\_num(), c[i]);

}

}

#pragma omp section

{

for (int i = 0; i < n; i++)

{

c[i] = a[i] - b[i];

printf("\nThread Id : %d Sub : %d", omp\_get\_thread\_num(), c[i]);

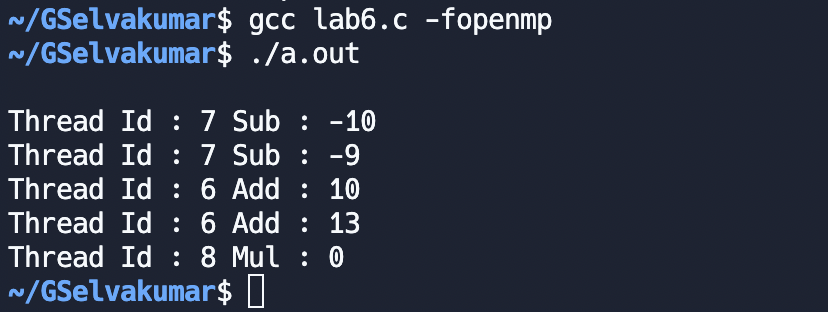
}

}

}

}

**OUTPUT**

****

**EXPERIMENT 7**

**OMP program to find eligible candidates for election**

**AIM:**

To write a OMP program to find eligible candidates for election based on their age (age < 18 and age >= 16).

**PROGRAM:**

#include <stdio.h>

#include <omp.h>

void main()

{

int age[5];

printf("\nEnter the age :");

for (int i = 0; i < 5; i++)

{

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

}

#pragma omp parallel

{

int tid = omp\_get\_thread\_num();

if (tid == 0)

{

for (int i = 0; i < 5; i++)

{

if (age[i] < 16 || age[i] >= 18)

{

printf("\nThread Id : %d Age : %d is not eligible", tid, age[i]);

}

}

}

else if (tid == 1)

{

for (int i = 0; i < 5; i++)

{

if (age[i] >= 16 && age[i] < 18)

{

printf("\nThread Id : %d Age : %d is eligible", tid, age[i]);

}

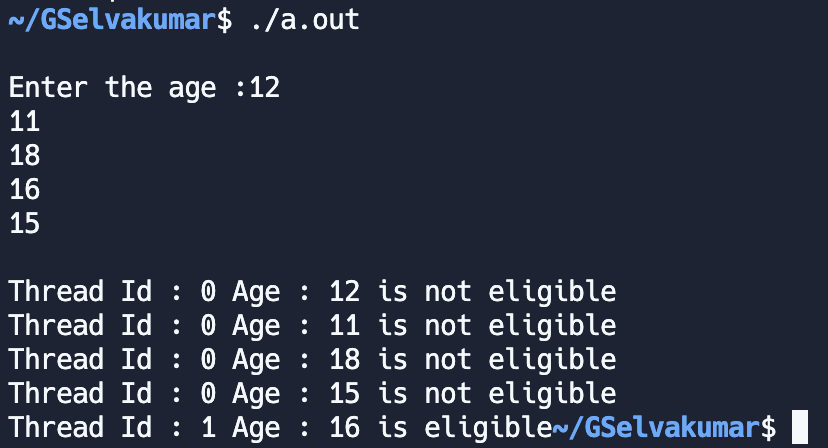
}

}

}

}

**OUTPUT**

****

**EXPERIMENT 8**

**OMP program to print time stamp**

**AIM:**

To write a OMP program to print the current time stamp of the threads.

**PROGRAM:**

#include <stdio.h>

#include <omp.h>

#include <time.h>

void main()

{

#pragma omp parallel

{

int tid = omp\_get\_thread\_num();

time\_t t;

if (tid == 0)

{

printf("\nThread Id : %d Time : %s", tid, ctime(&t));

}

else

{

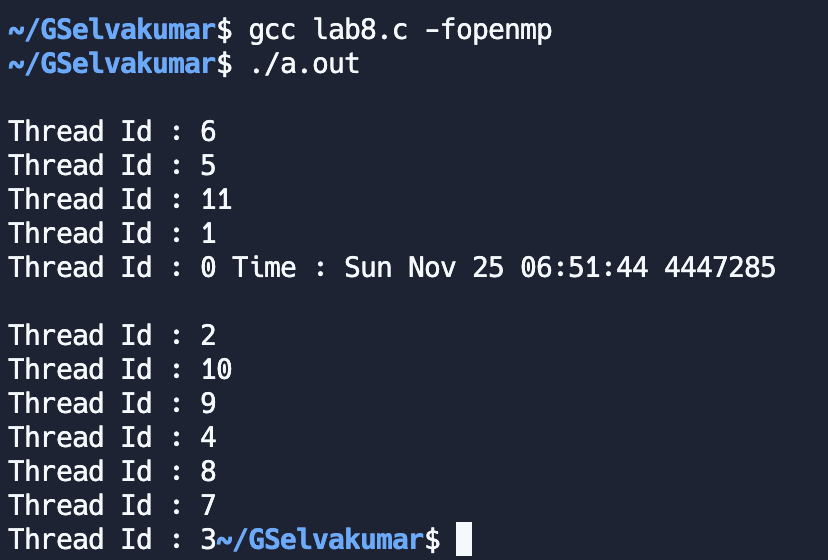
printf("\nThread Id : %d", tid);

}

}

}

**OUTPUT**

****

**EXPERIMENT 9**

**OMP program to find start, end and elapsed time**

**AIM:**

To write a OMP program to calculate start, end and elapsed thread for a thread.

**PROGRAM:**

#include <stdio.h>

#include <omp.h>

#include <time.h>

void main()

{

int n = 2;

int a[n], b[n], c[n];

double start = omp\_get\_wtime();

printf("\nStart time : %f", start);

for (int i = 0; i < n; i++)

{

a[i] = 2 \* i;

b[i] = 10 + i;

}

#pragma omp parallel sections

{

#pragma omp section

{

for (int i = 0; i < n; i++)

{

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

printf("\nThread Id : %d Add : %d", omp\_get\_thread\_num(), c[i]);

}

}

#pragma omp section

{

for (int i = 0; i < n; i++)

{

c[i] = a[i] \* b[i];

printf("\nThread Id : %d Mul : %d", omp\_get\_thread\_num(), c[i]);

}

}

#pragma omp section

{

for (int i = 0; i < n; i++)

{

c[i] = a[i] - b[i];

printf("\nThread Id : %d Sub : %d", omp\_get\_thread\_num(), c[i]);

}

}

}

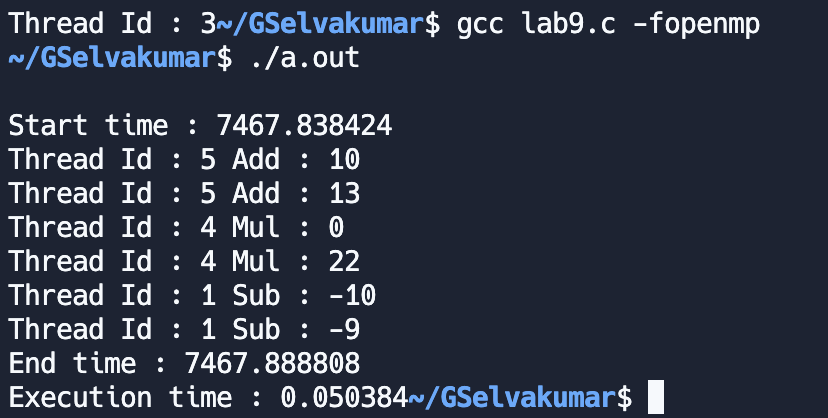
double end = omp\_get\_wtime();

printf("\nEnd time : %f", end);

printf("\nExecution time : %f", end - start);

}

**OUTPUT**

****

**EXPERIMENT 10**

**OMP program for managing VIT placement cell**

**AIM:**

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

**PROGRAM:**

#include<stdio.h>

#include<omp.h>

#include<string.h>

struct student

{

char name[5];

char company[5];

int regNo;

int pay;

};

void main()

{

struct student students[5];

printf("\nEnter Student details : ");

for(int i = 0; i < 5; i++)

{

printf("\nEnter Name of Student %d", i+1);

scanf("%s", students[i].name);

printf("\nEnter Company of Student %d", i+1);

scanf("%s", students[i].company);

printf("\nEnter Registor Number of Student %d", i+1);

scanf("%d", &students[i].regNo);

printf("\nEnter Pay of the Student %d", i+1);

scanf("%d", &students[i].pay);

}

#pragma omp parallel sections

{

#pragma omp section

{

int tid = omp\_get\_thread\_num();

int count[4];

double start = omp\_get\_wtime();

for(int i = 0; i < 4; i++)

{

if(strcmp(students[i].company, "Amazon") == 0)

{

count[0]++;

}

else if(strcmp(students[i].company, "Google") == 0)

{

count[1]++;

}

else if(strcmp(students[i].company, "Shell") == 0)

{

count[2]++;

}

else if(strcmp(students[i].company, "Intel") == 0)

{

count[3]++;

}

}

printf("\nThread Id : %d No of students placed in Amazon : %d", tid, count[0]);

printf("\nThread Id : %d No of students placed in Google : %d", tid, count[1]);

printf("\nThread Id : %d No of students placed in Shell : %d", tid, count[2]);

printf("\nThread Id : %d No of students placed in Intel : %d", tid, count[3]);

double end = omp\_get\_wtime();

printf("\nTotal time taken by Thread : %d is %f", tid, end-start);

}

#pragma omp section

{

double start = omp\_get\_wtime();

int tid = omp\_get\_thread\_num();

int sum = 0;

for(int i = 0; i < 5; i++)

{

sum += students[i].pay;

}

printf("\nThread Id : %d Average Pay : %d", tid, sum/5);

double end = omp\_get\_wtime();

printf("\nTotal time taken by Thread : %d is %f", tid, end - start);

}

#pragma omp section

{

double start = omp\_get\_wtime();

printf("\nThread Id : %d Number of Processors : %d", omp\_get\_thread\_num(), omp\_get\_num\_procs());

double end = omp\_get\_wtime();

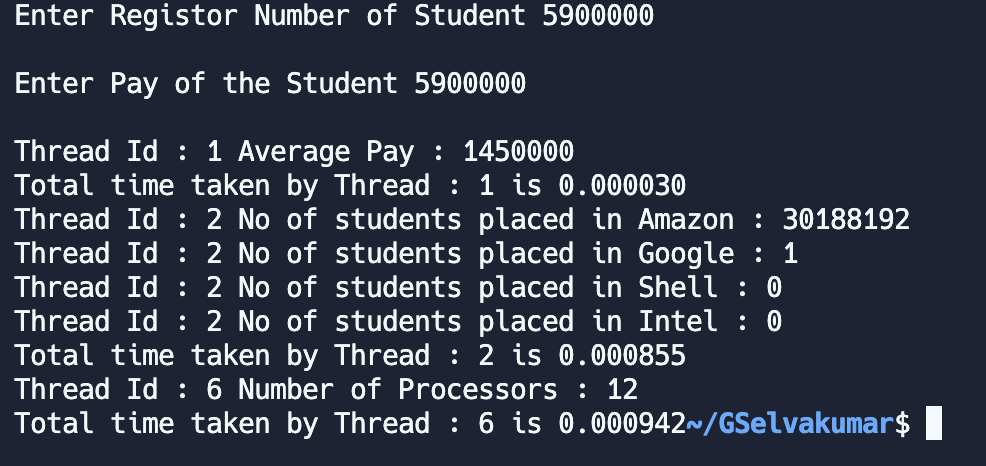
printf("\nTotal time taken by Thread : %d is %f", omp\_get\_thread\_num(), end - start);

}

}

}

**OUTPUT**

****

**EXPERIMENT 11**

**OMP program to use Private variables**

**AIM:**

To write a OMP program to use Private variables in a parallel region.

**PROGRAM:**

#include <stdio.h>

#include <omp.h>

void main()

{

int x = 9;

#pragma omp parallel for private(x)

for (int i = 0; i < 10; i++)

{

x = x + i;

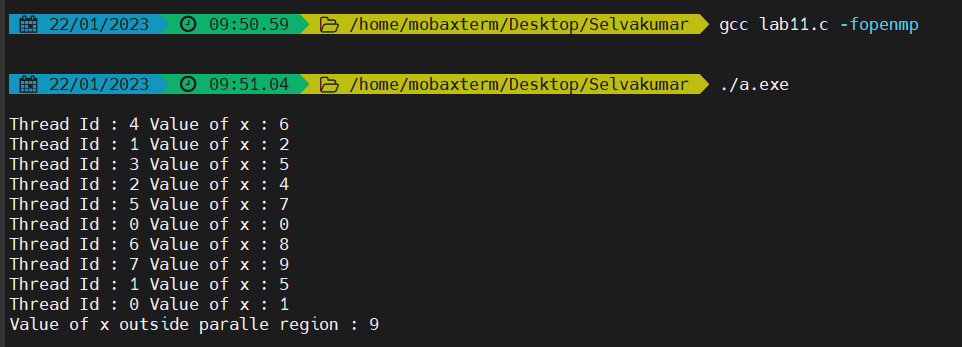
printf("\nThread Id : %d Value of x : %d", omp\_get\_thread\_num(), x);

}

printf("\nValue of x outside paralle region : %d", x);

}

**OUTPUT**

****

**EXPERIMENT 12**

**OMP program to restrict number of threads used**

**AIM:**

To write a OMP program to restrict the number of threads used to process a parallel region.

**PROGRAM:**

#include <stdio.h>

#include <omp.h>

void main()

{

int x = 9;

#pragma omp parallel for private(x) num\_threads(2)

for (int i = 0; i < 10; i++)

{

x = x + i;

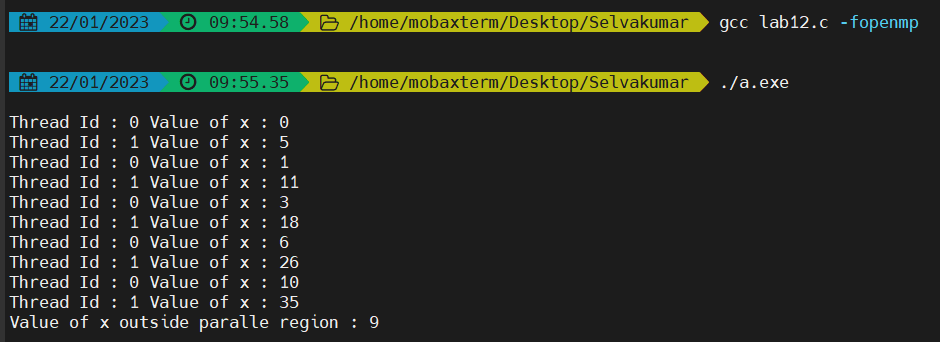
printf("\nThread Id : %d Value of x : %d", omp\_get\_thread\_num(), x);

}

printf("\nValue of x outside paralle region : %d", x);

}

**OUTPUT**

****

**EXPERIMENT 13**

**OMP program to use LastPrivate**

**AIM:**

To write a OMP program to use LastPrivate construct to retain the last iteration value.

**PROGRAM:**

#include<stdio.h>

#include<omp.h>

void main()

{

int x = 10;

printf("\nValue of x before entering parallel region : %d", x);

#pragma omp parallel for lastprivate(x)

for(int i = 0; i < 6; i++){

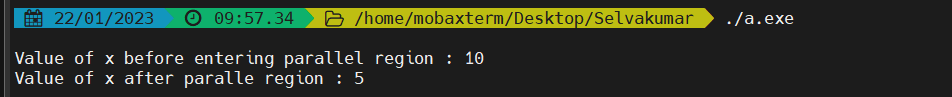
x = i;

}

printf("\nValue of x after paralle region : %d", x);

}

**OUTPUT**

****

**EXPERIMENT 14**

**OMP program to use FristPrivate**

**AIM:**

To write a OMP program to use FirstPrivate construct.

**PROGRAM:**

#include<stdio.h>

#include<omp.h>

void main()

{

int x = 10;

printf("\nValue of x before entering parallel region : %d", x);

#pragma omp parallel for firstprivate(x)

for(int i = 0; i < 6; i++){

printf("\nThread Id : %d Value of x inside parallel region : %d",omp\_get\_thread\_num(), x);

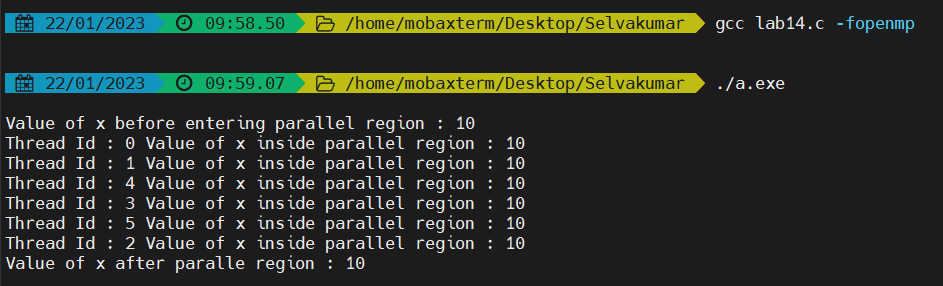
x = i;

}

printf("\nValue of x after paralle region : %d", x);

}

**OUTPUT**

****

**EXPERIMENT 15**

**OMP program for Math Application**

**AIM:**

To write a OMP program for math application, to identify if the given number is rational, prime of perfect number

**PROGRAM:**

#include <stdio.h>

#include <omp.h>

int isPerfectNum(int n)

{

int sum = 0;

for (int i = 1; i <= n / 2; i++)

{

if (n % i == 0)

{

sum += i;

}

}

if (sum == n)

return 1;

return 0;

}

int isPrimeNum(int n)

{

if(n == 2) return 1;

for (int i = 2; i <= n / 2; i++)

{

if (n % i == 0)

return 0;

}

return 1;

}

void main()

{

int n;

int isRational = 0, isPrime = 0, isPerfect = 0;

printf("\nEnter the Number : ");

scanf("%d", &n);

#pragma omp parallel sections private(isRational) firstprivate(isPrime) lastprivate(isPerfect) num\_threads(3)

{

#pragma omp section

{

isRational = 1;

int tid = omp\_get\_thread\_num();

if(isRational)

{

printf("\nThread Id : %d Num : %d is Rational", tid, n);

}

else

{

printf("\nThread Id : %d Num : %d is not Rational", tid, n);

}

}

#pragma omp section

{

isPerfect = isPerfectNum(n);

int tid = omp\_get\_thread\_num();

if(isPerfect)

{

printf("\nThread Id : %d Num : %d is Perfect", tid, n);

}

else

{

printf("\nThread Id : %d Num : %d is not Perfect", tid, n);

}

}

#pragma omp section

{

isPrime = isPrimeNum(n);

int tid = omp\_get\_thread\_num();

if(isPrime)

{

printf("\nThread Id : %d Num : %d is Prime", tid, n);

}

else

{

printf("\nThread Id : %d Num : %d is not Prime", tid, n);

}

}

}

}

**OUTPUT**

**Text

Description automatically generated**

**EXPERIMENT 16**

**OMP program to perform Static scheduling**

**AIM:**

To write a OMP program to perform static scheduling.

**PROGRAM:**

#include<stdio.h>

#include<omp.h>

void main()

{

#pragma omp parallel for schedule(static, 2)

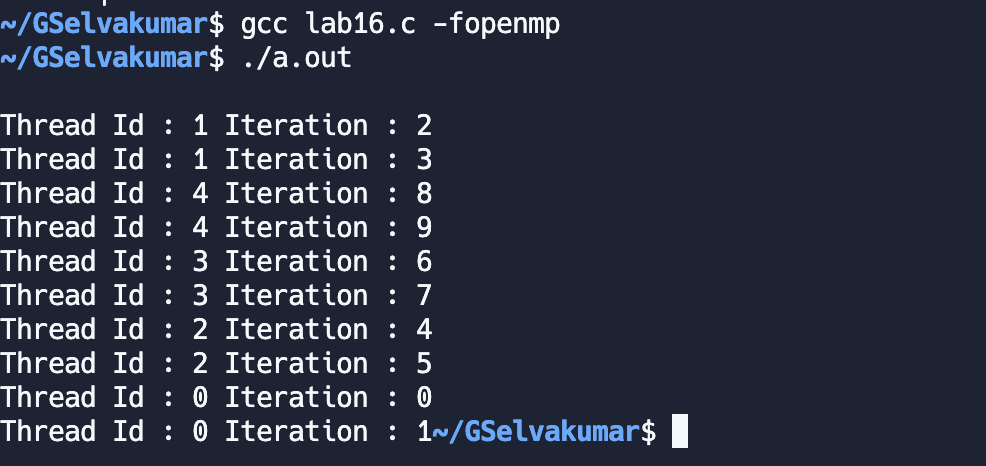
for(int i = 0; i < 10; i++){

printf("\nThread Id : %d Iteration : %d", omp\_get\_thread\_num(), i);

}

}

**OUTPUT**

****

**EXPERIMENT 17**

**OMP program to perform Dynamic Scheduling**

**AIM:**

To write a OMP program to perform Dynamic Scheduling.

**PROGRAM:**

#include<stdio.h>

#include<omp.h>

void main()

{

#pragma omp parallel for schedule(dynamic, 2)

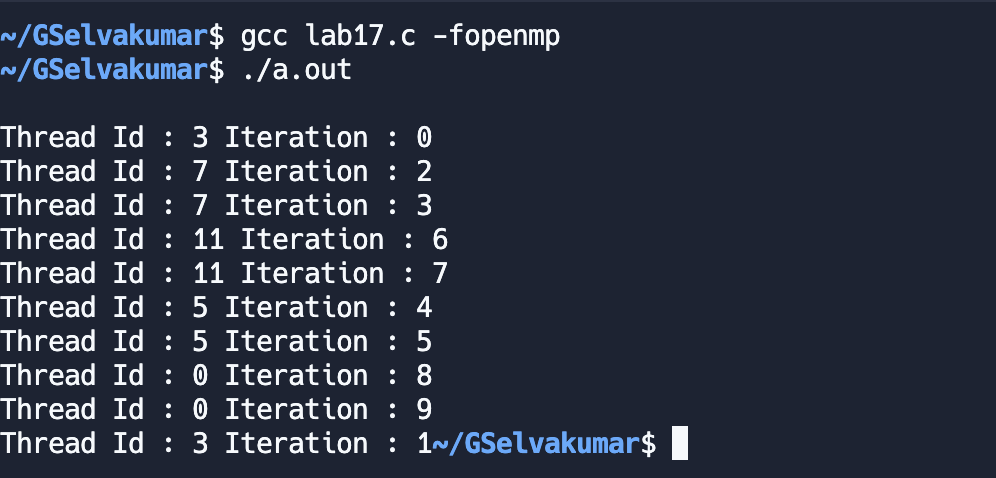
for(int i = 0; i < 10; i++){

printf("\nThread Id : %d Iteration : %d", omp\_get\_thread\_num(), i);

}

}

**OUTPUT**

****

**EXPERIMENT 18**

**OMP program to perform Guided Scheduling**

**AIM:**

To write a OMP program to perform Guided Scheduling.

**PROGRAM:**

#include<stdio.h>

#include<omp.h>

void main()

{

#pragma omp parallel for schedule(guided, 2)

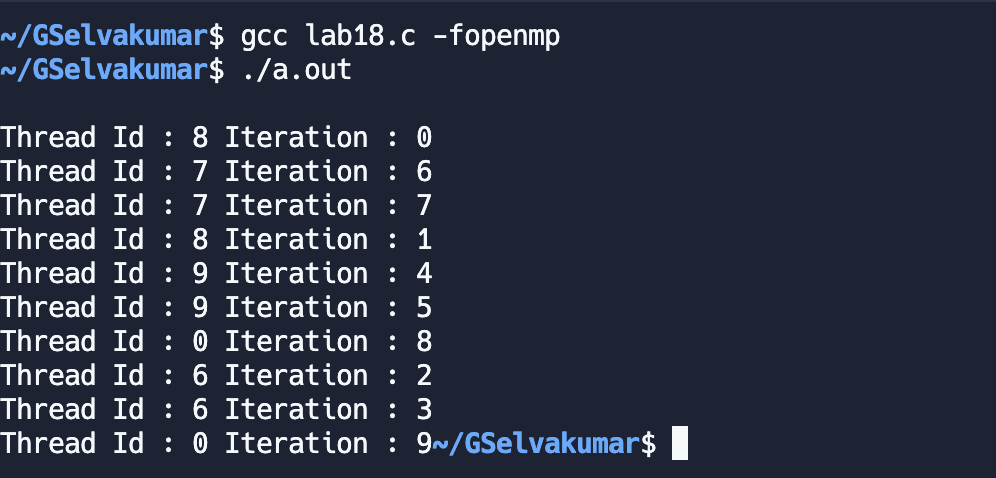
for(int i = 0; i < 10; i++){

printf("\nThread Id : %d Iteration : %d", omp\_get\_thread\_num(), i);

}

}

**OUTPUT**

****

**EXPERIMENT 19**

**OMP program for Toy Modelling Company**

**AIM:**

To write a OMP program for the quality checking unit in the toy modelling unit which has an incremental counter and counts the tested toy from 0 to 256. Once the counter reaches the max value all tested toys will be transferred to the dispatching unit in which this counter decrement from the maximum of 256 and reaches zero. Using Last private to get the max value and all three-scheduling concepts.

**PROGRAM:**

#include<stdio.h>

#include<omp.h>

void main(){

int counter = 0;

//count the tested toys

#pragma omp parallel for schedule(static, 10) lastprivate(counter)

for (int i = 1; i <= 256; i++)

{

counter = i;

printf("\nThread : %d Testing Toy Id : %d", omp\_get\_thread\_num(), i);

}

// //dispatch the tested toys

// #pragma omp parallel for schedule(dynamic, 10) lastprivate(counter)

// for(int i = 1; i <= 256; i++)

// {

// counter = 256 - i;

// printf("\nThread : %d Dispatching Toy Id : %d", omp\_get\_thread\_num(), i);

// }

//dispatch the tested toys

#pragma omp parallel for schedule(guided, 10) lastprivate(counter)

for(int i = 1; i <= 256; i++)

{

counter = 256 - i;

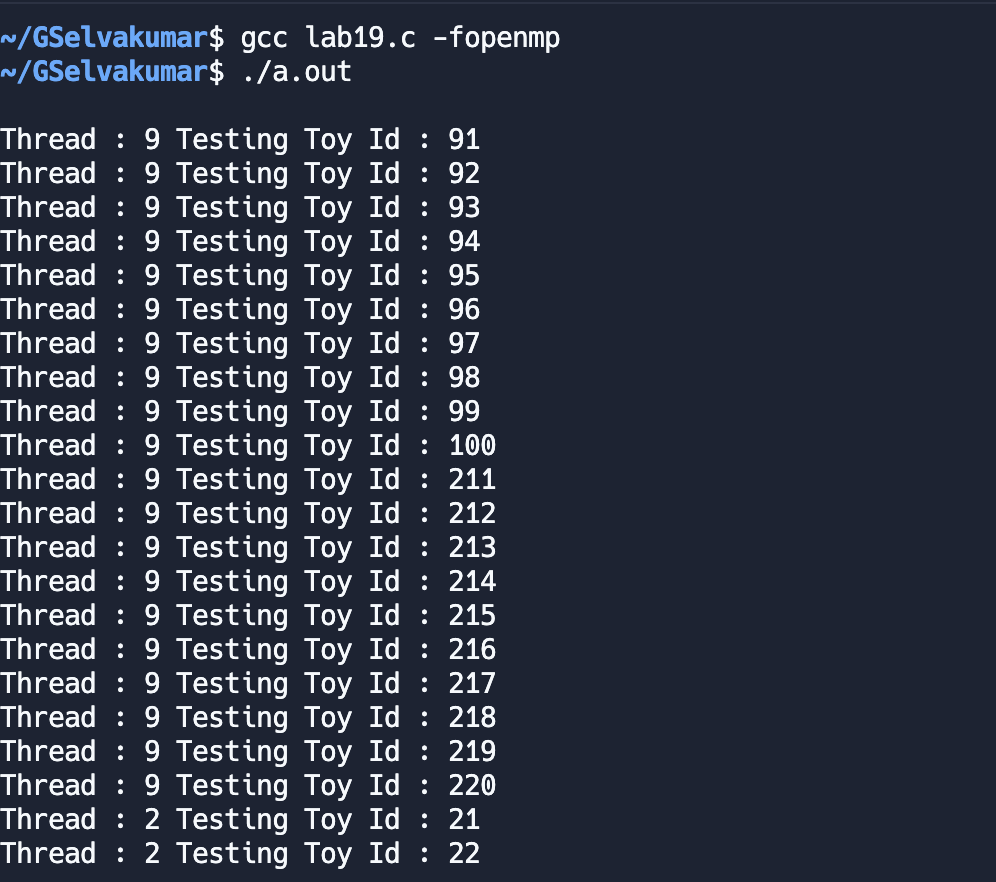
printf("\nThread : %d Dispatching Toy Id : %d", omp\_get\_thread\_num(), i);

}

printf("Toys left to process are : %d", counter);

}

**OUTPUT**

****

**EXPERIMENT 20**

**OMP program to execute parallel region in ordered fashion**

**AIM:**

To write a OMP program to print region in ordered fashion.

**PROGRAM:**

#include<stdio.h>

#include<omp.h>

void main()

{

#pragma omp parallel for ordered

for(int i = 0; i < 5; i++){

printf("\nThread Id : %d Un\_Ordered Iteration : %d", omp\_get\_thread\_num(), i);

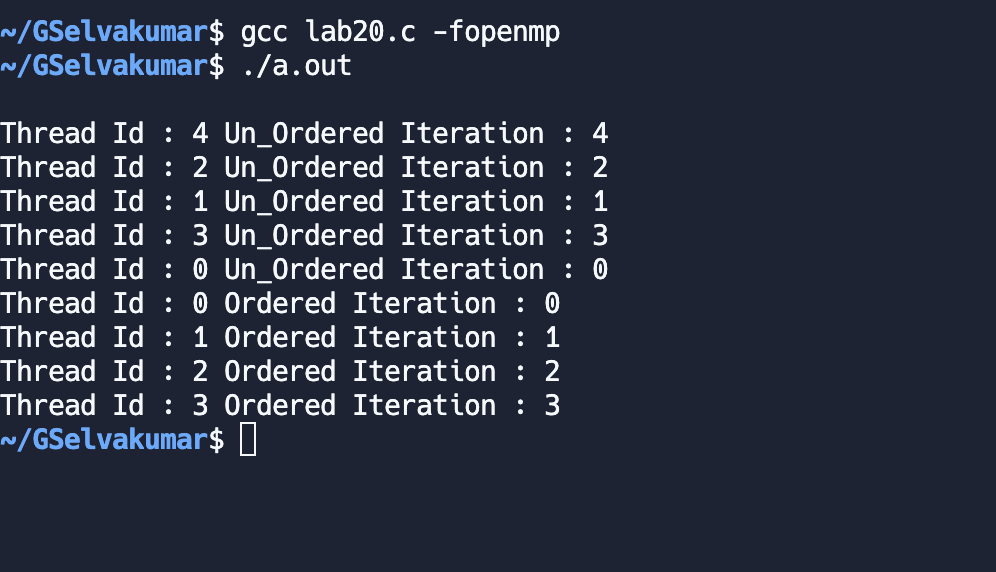
#pragma omp ordered

printf("\nThread Id : %d Ordered Iteration : %d", omp\_get\_thread\_num(), i);

}

}

**OUTPUT**

****

**EXPERIMENT 21**

**OMP program to use locks**

**AIM:**

To write a OMP program using locks.

**PROGRAM:**

#include <omp.h>

#include <stdio.h>

int main()

{

int id, i;

omp\_lock\_t mylock;

omp\_init\_lock(&mylock);

#pragma omp parallel

{

id = omp\_get\_thread\_num();

#pragma omp parallel for

for (int i = 0; i < 3; i++)

{

omp\_set\_lock(&mylock);

printf("Thread %d is executing iteration %d\n", id, i);

omp\_unset\_lock(&mylock);

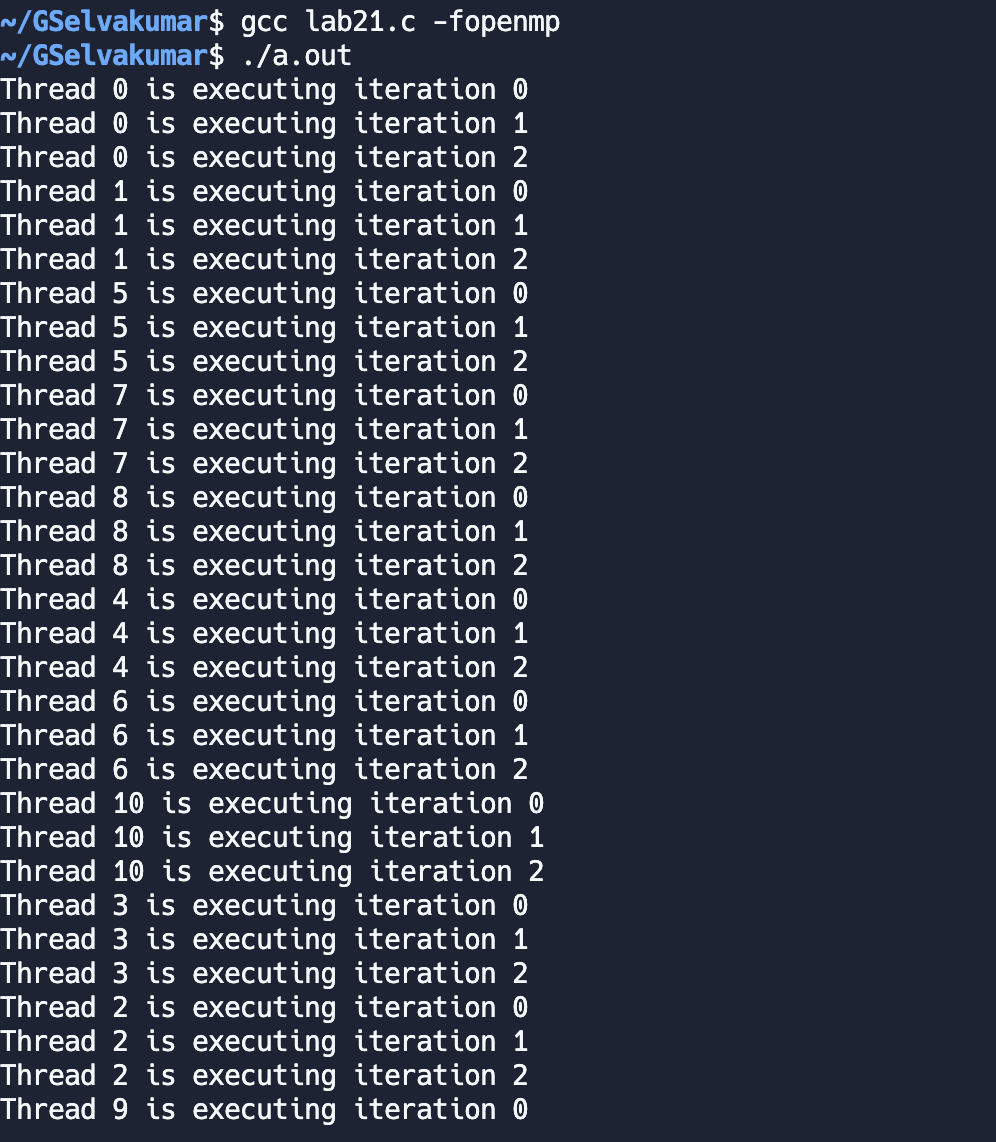
}

}

omp\_destroy\_lock(&mylock);

}

**OUTPUT**

****

**EXPERIMENT 22**

**OMP program using locks to print n even and n odd numbers**

**AIM:**

To write a OMP program to print n odd and n even numbers using locks.

**PROGRAM:**

#include<stdio.h>

#include<omp.h>

void main()

{

int n;

printf("\nEnter the value of n : ");

scanf("%d", &n);

#pragma omp parallel for ordered

for(int i = 1; i <= n \* 2; i+=2){

#pragma omp ordered

printf("\nThread Id : %d Odd number : %d", omp\_get\_thread\_num(), i);

}

#pragma omp parallel for ordered

for(int i = 2; i <= n \* 2; i+=2){

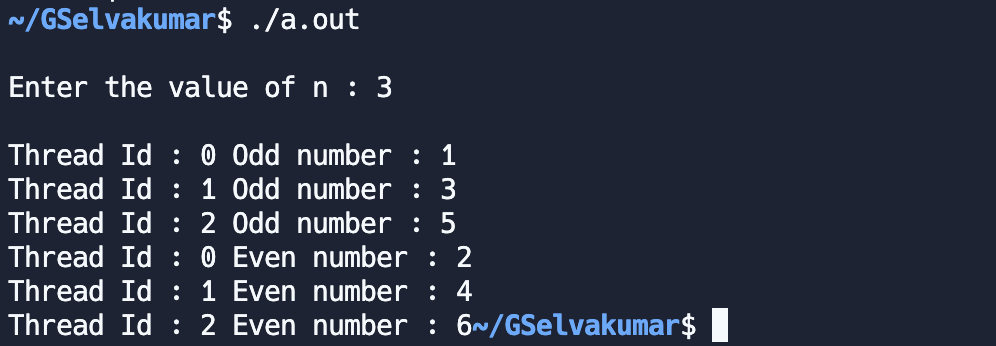
#pragma omp ordered

printf("\nThread Id : %d Even number : %d", omp\_get\_thread\_num(), i);

}

}

**OUTPUT**

****

**EXPERIMENT 23**

**OMP program for TechnoVIT**

**AIM:**

To write a OMP program for TechnoVIT, where students can register, if they want, they can unregister. Registered students (registration numbers: 9,3,2…) are stored in an array. Only one can register or unregister at a time. But they can view the registered list without any constraint. Parallel program has to be designed with the help of locks incorporating ordered, sections, and scheduling.

**PROGRAM:**

#include <stdio.h>

#include <omp.h>

#include <unistd.h>

int registeration[1000];

omp\_lock\_t lock;

void main()

{

omp\_init\_lock(&lock);

#pragma omp parallel sections

{

// Registration parallel section

#pragma omp section

{

for (int i = 0; i < 5; i++)

{

omp\_set\_lock(&lock);

int id;

printf("\nEnter the Id to register : ");

scanf("%d", &id);

sleep(2);

registeration[id] = 1;

printf("\nRegisteration completed for : %d", id);

omp\_unset\_lock(&lock);

}

}

// Un-Register parallel section

#pragma omp section

{

for (int i = 0; i < 5; i++)

{

omp\_set\_lock(&lock);

int id;

printf("\nEnter the Id to Un-register : ");

scanf("%d", &id);

sleep(2);

registeration[id] = 0;

printf("\nUn-Registered : %d", id);

omp\_unset\_lock(&lock);

}

}

}

// Registered student details

#pragma omp parallel for schedule(static, 2) ordered

for (int i = 1; i < 1000; i++)

{

if (registeration[i] == 1)

{

printf("\nThread Id : %d Registered Student : %d", omp\_get\_thread\_num(), i);

}

}

}

**OUTPUT**

**Text

Description automatically generated**

**EXPERIMENT 24**

**OMP program using Barrier and File Operations**

**AIM:**

To write a OMP program to using file operations using Barrier synchronization construct.

**PROGRAM:**

#include<stdio.h>

#include<omp.h>

void main()

{

FILE \*fptr;

fptr = fopen("test.txt", "w");

#pragma omp parallel

{

int tid = omp\_get\_thread\_num();

int total = omp\_get\_num\_threads();

if(tid == 0){

printf("\nThread Id : %d", tid);

}

else{

for(int i = 0; i < 1000; i++);

printf("\nThread Id : %d", tid);

}

#pragma omp barrier

fprintf(fptr, "Hello World from thread Id : %d of %d\n", tid, total);

}

fclose(fptr);

}

**OUTPUT**

****

**EXPERIMENT 25**

**OMP program to find sum series using Barrier**

**AIM:**

To write a OMP program calculate ½ + ¼ + …. using barrier.

**PROGRAM:**

#include<stdio.h>

#include<math.h>

#include<omp.h>

void main()

{

int n;

printf("\nEnter the value of N : ");

scanf("%d", &n);

FILE \*fptr;

fptr = fopen("test.txt", "w");

#pragma omp parallel

{

float ans = 0;

int tid = omp\_get\_thread\_num();

float d = 2;

#pragma omp parallel for schedule(static, 2) ordered

for(int i = 1; i <= n; i++){

ans += ((float)1/d);

d = d \* (float)2;

}

#pragma omp barrier

fprintf(fptr, "Thread Id : %d Sum is : %f Last Value : %f\n", tid, ans, (float)1/(d / 2));

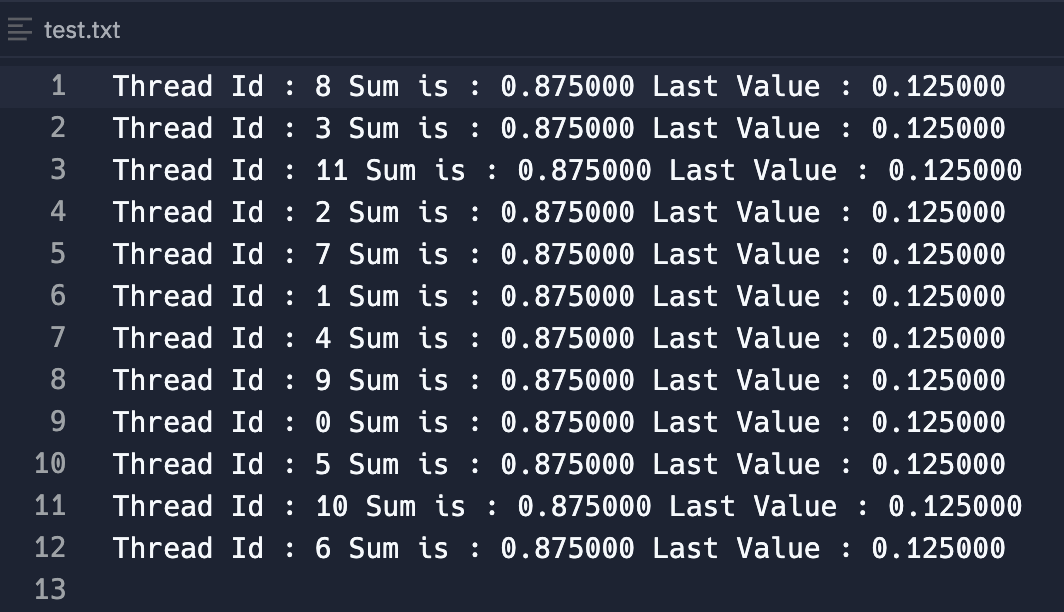
}

fclose(fptr);

}

**OUTPUT**

****

****

**EXPERIMENT 26**

**OMP program for Matrix-Matrix multiplication**

**AIM:**

To write a OMP program to perform Matrix-Matrix multiplication.

**PROGRAM:**

#include<stdio.h>

#include<omp.h>

#include<sys/time.h>

#define N 10

int a[N][N], b[N][N], c[N][N];

void main()

{

int i, j, k;

int n = N;

struct timeval start, end;

for(int i = 0; i < n; i++){

for(int j = 0; j < n; j++){

a[i][j] = 2;

b[i][j] = 2;

}

}

gettimeofday(&start, NULL);

#pragma omp parallel for private(i,j,k) shared(a,b,c)

for(int i = 0; i < n; i++){

for(int j = 0; j < n; j++){

for(int k = 0; k < n; k++){

c[i][j] += a[i][k] \* b[k][j];

}

}

}

gettimeofday(&end, NULL);

double elapsed = (double)(end.tv\_sec-start.tv\_sec)+(double)(end.tv\_usec-start.tv\_usec)\*1.e-6;

printf("\nElapsed time : %lf\n", elapsed);

for(int i = 0; i < n; i++){

for(int j = 0; j < n; j++){

printf("%d\t", c[i][j]);

}

printf("\n");

}

}

**OUTPUT**

**A picture containing table

Description automatically generated**

**EXPERIMENT 27**

**OMP program for Matrix-Vector multiplication**

**AIM:** To write a OMP program to perform Matrix-Vector multiplication.

**PROGRAM:**

#include<stdio.h>

#include<omp.h>

#include<sys/time.h>

#define N 10

int a[N][N], b[N][1], c[N][1];

void main()

{

int i, j, k;

int n = N;

struct timeval start, end;

for(int i = 0; i < n; i++){

for(int j = 0; j < n; j++){

a[i][j] = 2;

}

b[i][0] = i+1;

}

gettimeofday(&start, NULL);

#pragma omp parallel for private(i,j,k) shared(a,b,c)

for(int i = 0; i < n; i++){

for(int k = 0; k < n; k++){

c[i][0] += a[i][k] \* b[k][0];

}

}

gettimeofday(&end, NULL);

double elapsed = (double)(end.tv\_sec-start.tv\_sec)+(double)(end.tv\_usec-start.tv\_usec)\*1.e-6;

printf("\nElapsed time : %lf\n", elapsed);

for(int i = 0; i < n; i++){

printf("%d\t", c[i][0]);

}

}

**OUTPUT**

**Text

Description automatically generated**

**EXPERIMENT 30**

**VTune Profiler for Matrix-Matrix multiplication**

**AIM:**

To use VTune profiler for analysing Matrix-Matrix multiplication

**PROGRAM:**

#include <iostream>

#include <omp.h>

const int N = 20;

int main()

{

long a[N][N], b[N][N], c[N][N];

for (int i = 0; i < N; i++) {

for (int j = 0; j < N; j++) {

a[i][j] = 1;

b[i][j] = 1;

c[i][j] = 0;

}

}

#pragma omp parallel for

for (int i = 0; i < N; i++) {

#pragma omp parallel for

for (int j = 0; j < N; j++) {

#pragma omp parallel for

for (int k = 0; k < N; k++) {

std::cout << omp\_get\_thread\_num();

c[i][j] += a[i][k] \* b[k][j];

}

}

}

for (int i = 0; i < N; i++) {

for (int j = 0; j < N; j++) {

std::cout << c[i][j] << " ";

}

std::cout << "\n";

}

**OUTPUT**

A picture containing graphical user interface

Description automatically generated

**EXPERIMENT 31**

**VTune Profiler for Matrix-Vector multiplication**

**AIM:**

To use VTune profiler for analysing Matrix-Vector multiplication

**PROGRAM:**

#include<stdio.h>

#include<omp.h>

#define N 100

int a[N][N], b[N][1], c[N][1];

void main()

{

int i, j, k;

int n = N;

for (int i = 0; i < n; i++) {

for (int j = 0; j < n; j++) {

a[i][j] = 2;

}

b[i][0] = i + 1;

}

#pragma omp parallel for private(i,j,k) shared(a,b,c)

for (int i = 0; i < n; i++) {

for (int k = 0; k < n; k++) {

c[i][0] += a[i][k] \* b[k][0];

}

}

for (int i = 0; i < n; i++) {

printf("%d\t", c[i][0]);

}

}

**OUTPUT**

**Graphical user interface, application

Description automatically generated**

**EXPERIMENT 32**

**VTune Profiler for Quick Sort**

**AIM:**

To use VTune profiler for analysing Quick Sort.

**PROGRAM:**

#include <omp.h>

#include <stdio.h>

#include <iostream>

#include <stdlib.h>

#include <windows.h>

int partition(int a[], int low, int high)

{

int pivot = a[low];

int i = low + 1;

int j = high;

int temp;

while (i <= j)

{

while (a[i] <= pivot)

i++;

while (a[j] > pivot)

j--;

if (i < j)

{

temp = a[i];

a[i] = a[j];

a[j] = temp;

}

}

temp = a[low];

a[low] = a[j];

a[j] = temp;

return j;

}

void quicksort(int a[], int low, int high)

{

int j;

if (low < high)

{

j = partition(a, low, high);

#pragma omp parallel sections

{

#pragma omp section

{

quicksort(a, low, j - 1);

}

#pragma omp section

{

quicksort(a, j + 1, high);

}

}

}

}

void main()

{

int a[100], n = 100, i;

for(int i = 100; i >= 1; i--)

{

a[100 - i] = i;

}

quicksort(a, 0, n - 1);

printf("Sorted elements:");

for (i = 0; i < 100; i++)

std::cout << a[i] << " ";

}

**OUTPUT**

**Graphical user interface

Description automatically generated**

**EXPERIMENT 33**

**VTune Profiler for Minimum Spanning Tree**

**AIM:**

To use VTune profiler for analysing Kruskal algorithm to find Minimum Spanning Tree.

**PROGRAM:**

#include<iostream>

#include<stdio.h>

#include<conio.h>

#include<stdlib.h>

#include<omp.h>

int i, j, k, a, b, u, v, n, ne = 1, edge1, edge2, e;

int min, mincost = 0, cost[101][101], parent[101];

int find(int i)

{

while (parent[i])

i = parent[i];

return i;

}

int uni(int i, int j)

{

if (i != j)

{

parent[j] = i;

return 1;

}

return 0;

}

void main()

{

printf("\n\n\tImplementation of Kruskal's algorithm\n\n");

printf("\nEnter the no. of vertices\n");

std::cin >> n;

printf("Enter the cost of each cell as adjacency matrix. \n");

for (i = 1; i <= n; i++)

{

for (j = 1; j <= n; j++)

{

std::cin >> cost[i][j];

if (cost[i][j] == 0)

cost[i][j] = 999;

}

}

printf("\nThe edges of Minimum Cost Spanning Tree are\n\n");

#pragma omp parallel reduction(+: mincost), private(min,a,u,v,b)

{

while (ne < n)

{

for (i = 1, min = 999; i <= n; i++)

{

for (j = 1; j <= n; j++)

{

if (cost[i][j] < min)

{

min = cost[i][j];

a = u = i;

b = v = j;

}

}

}

u = find(u);

v = find(v);

if (uni(u, v))

{

printf("\n%d edge (%d,%d) =%d\n", ne++, a, b, min);

mincost += min;

}

cost[a][b] = cost[b][a] = 999;

}

}

printf("\n\tMinimum cost = %d\n", mincost);

}

**OUTPUT**

Graphical user interface

Description automatically generated

**EXPERIMENT 34**

**Simple CUDA program**

**AIM:**

To write a simple CUDA program.

**PROGRAM:**

%%cu

#include <stdio.h>

\_\_global\_\_ void Hellokernel()

{

}

main()

{

Hellokernel <<<1, 1>>>();

printf("Hello Selvakumar\n");

return 0;

}

%%cu

#include <stdio.h>

\_\_global\_\_ void add(int a, int b, int \*c)

{

\*c = a + b;

}

int main(void)

{

int c;

int \*dev\_c;

cudaMalloc((void\*\*)&dev\_c, sizeof(int));

add << <1, 1 >> > (2, 7, dev\_c);

cudaMemcpy(&c, dev\_c, sizeof(int),

cudaMemcpyDeviceToHost);

printf("Selvakumar: ");

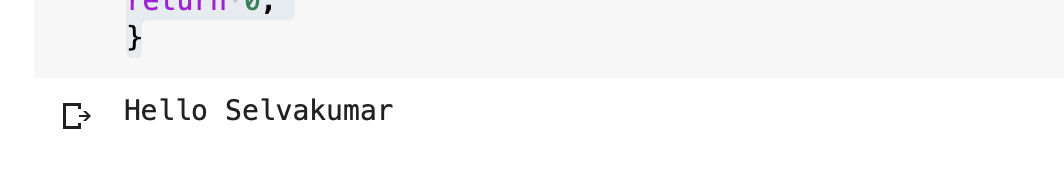
printf("2 + 7 = %d\n", c);

cudaFree(dev\_c);

return 0;

}

**OUTPUT**

****

**Graphical user interface, application

Description automatically generated**

**EXPERIMENT 35**

**CUDA program for performing Vector Operations**

**AIM:**

To write a CUDA program for performing Vector Addition and Multiplication.

**PROGRAM:**

%%cu

#include <stdio.h>

\_\_global\_\_ void vector\_add(int \*out\_d, int \*a, int \*b, int n)

{

int bx = blockIdx.x;

int by = blockIdx.y;

int tx = threadIdx.x;

int ty = threadIdx.y;

int row = by\*blockDim.y + ty;

int col = bx\*blockDim.x + tx;

int dim = gridDim.x\*blockDim.x;

int i = row\*dim + col;

out\_d[i] = a[i] + b[i];

}

int main()

{

int \*a, \*b, \*out\_d,\*out;

int \*d\_a, \*d\_b;

int N=6;

int i;

a = (int\*)malloc(sizeof(int) \* N);

b = (int\*)malloc(sizeof(int) \* N);

out = (int\*)malloc(sizeof(int) \* N);

for (i=0;i<N;i++)

{

a[i]=i;

b[i]=i\*2;

}

cudaMalloc((void\*\*)&d\_a, sizeof(int) \* N);

cudaMalloc((void\*\*)&d\_b, sizeof(int) \* N);

cudaMalloc((void\*\*)&out\_d, sizeof(int) \* N);

cudaMemcpy(d\_a, a, sizeof(int) \* N, cudaMemcpyHostToDevice);

cudaMemcpy(d\_b, b, sizeof(int) \* N, cudaMemcpyHostToDevice);

vector\_add<<<2,4>>>(out\_d, d\_a, d\_b, N);

cudaMemcpy(out, out\_d, sizeof(int) \* N, cudaMemcpyDeviceToHost);

printf("Success Selvakumar!\n");

for (i=0;i<N;i++)

{

printf("%d\n",out[i]);

}

cudaFree(d\_a);

cudaFree(d\_b);

cudaFree(out\_d);

free(a);

free(b);

free(out);

return 0;

}

%%cu

#include <stdio.h>

\_\_global\_\_ void matrixMul( int\* Pd, int\* Md, int\* Nd, int width)

{

int bx = blockIdx.x;

int by = blockIdx.y;

int tx = threadIdx.x;

int ty = threadIdx.y;

int col = by\*blockDim.y + ty;

int row = bx\*blockDim.x + tx;

int Pvalue=0;

for (int k=0;k<width;++k)

Pvalue+=Md[row\*width+k]\*Nd[k\*width+col];

Pd[row\*width+col]=Pvalue;

}

int main()

{

int \*M, \*N1, \*Md, \*Nd, \*Pd, \*P;

const int xb = 3; /\* gridDim.x \*/

const int yb = 3; /\* gridDim.y \*/

const int zb = 1; /\* gridDim.z \*/

const int xt = 3; /\* blockDim.x \*/

const int yt = 3; /\* blockDim.y \*/

const int zt = 1; /\* blockDim.z \*/

int N,width;

int i;

width=9;

N=width\*width;

M = (int\*)malloc(sizeof(int) \* N);

N1 = (int\*)malloc(sizeof(int) \* N);

P = (int\*)malloc(sizeof(int) \* N);

for (i=0;i<N;i++)

{

M[i]=i;

N1[i]=i\*2;

}

dim3 dimGrid(xb,yb,zb);

dim3 dimBlock(xt,yt,zt);

cudaMalloc((void\*\*)&Md, sizeof(int) \* N);

cudaMalloc((void\*\*)&Nd, sizeof(int) \* N);

cudaMalloc((void\*\*)&Pd, sizeof(int) \* N);

cudaMemcpy(Md, M, sizeof(int) \* N, cudaMemcpyHostToDevice);

cudaMemcpy(Nd, N1, sizeof(int) \* N, cudaMemcpyHostToDevice);

matrixMul<<<dimGrid,dimBlock>>>(Pd, Md, Nd, width);

cudaMemcpy(P, Pd, sizeof(int) \* N, cudaMemcpyDeviceToHost);

printf("Success Selvakumar!\n");

for (i=0;i<N;i++)

{

printf("%d\n",P[i]);

}

cudaFree(Md);

cudaFree(Nd);

cudaFree(Pd);

free(M);

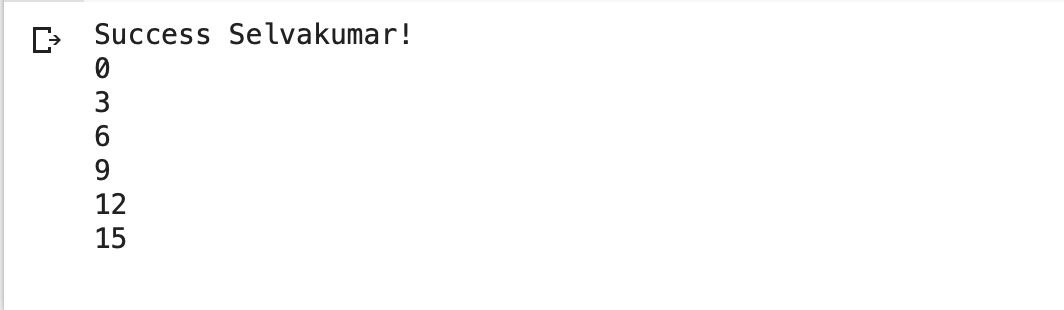
free(N1);

free(P);

return 0;

}

**OUTPUT**

****

**A picture containing text

Description automatically generated**

**EXPERIMENT 36**

**CUDA program for Matrix Fill**

**AIM:** To write a CUDA program to fill matrix.

**PROGRAM:**

%%cu

#include <stdio.h>

\_\_global\_\_ void matrixFill ( int \*x )

{

int bx = blockIdx.x;

int by = blockIdx.y;

int tx = threadIdx.x;

int ty = threadIdx.y;

int col = by\*blockDim.y + ty;

int row = bx\*blockDim.x + tx;

int dim =blockDim.x\*gridDim.x;

int i = row\*dim + col;

x[i] = i;

}

int main ( int argc, char\* argv[] )

{

const int xb = 2; /\* gridDim.x \*/

const int yb = 2; /\* gridDim.y \*/

const int zb = 1; /\* gridDim.z \*/

const int xt = 2; /\* blockDim.x \*/

const int yt = 2; /\* blockDim.y \*/

const int zt = 1; /\* blockDim.z \*/

const int n = xb\*yb\*zb\*xt\*yt\*zt;

printf("Welcome Selvakumar!\n");

printf("allocating array of length %d...\n",n);

int \*xhost = (int\*)calloc(n,sizeof(int));

for(int i=0; i<n; i++) xhost[i] = -1.0;

int \*xdevice;

size\_t sx = n\*sizeof(int);

cudaMalloc((void\*\*)&xdevice,sx);

cudaMemcpy(xdevice,xhost,sx,cudaMemcpyHostToDevice);

dim3 dimGrid(xb,yb,zb);

dim3 dimBlock(xt,yt,zt);

matrixFill<<<dimGrid,dimBlock>>>(xdevice);

cudaMemcpy(xhost,xdevice,sx,cudaMemcpyDeviceToHost);

cudaFree(xdevice);

int \*p = xhost;

for(int i1=0; i1 < xb; i1++)

for(int i2=0; i2 < yb; i2++)

for(int i3=0; i3 < zb; i3++)

for(int i4=0; i4 < xt; i4++)

for(int i5=0; i5 < yt; i5++)

for(int i6=0; i6 < zt; i6++)

printf("x[%d][%d][%d][%d][%d][%d] = %d\n",i1,i2,i3,i4,i5,i6,\*(p++));

return 0;

}

**OUTPUT**

A picture containing table

Description automatically generated