

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

# M.Tech –CSE,AI & ML, Data Analytics

**Computer Architecture and Organisation-**

**MCSE503L LAB RECORD**

**Prepared By**

## Atiendirya Verma 22MCS1009

**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. | 29/9/22 | To get the Thread number using openMP | 3 |
| 2 | 28/9/22 | Parallel region constructs | 6 |
| 3 | 5/10/22 | Parallel Sections construct | 8 |
| 4 | 8/10/22 | Date and Time function | 12 |
| 5 | 12/10/22 | Private and shared space | 17 |
| 6 | 19/10/22 | Static, dynamic, and Guided scheduling | 19 |
| 7 | 16/10/22 | Ordered, unordered scheduling and locks | 23 |
| 8 | 23/11/22 | Open mp Barrier and File operations | 27 |
| 9 | 30/11/22 | Matrix Multiplication | 32 |
| 10 | 7/12/22 | Vtune Profile Matrix multiplication | 35 |
| 11 | 14/12/22 | Vtune Profile Matrix-vector multiplication | 39 |
| 12 | 21/12/22 | Vtune Profile of Minimum Spanning Tree using Kruskal Algorithm | 42 |
| 13 | 11/01/23 | CUDA programming | 50 |
| 14 | 18/01/23 | Performing vector arithmetic operation in CUDA | 52 |

## EXPERIMENT 1

**<<Parallel processing in OpenMP>>**

**AIM:** To get the thread number using OpenMP

**PROGRAM:**

#include<omp.h>

#include<stdio.h> int main()

{

#pragma omp parallel

printf("Hello World ! %d \n",

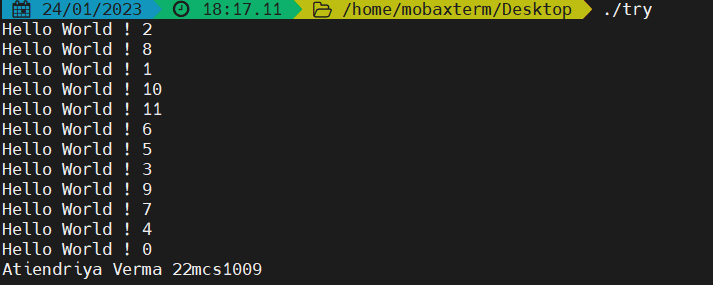
omp\_get\_thread\_num());

printf(“Atiendriya Verma 22mcs1009);

return 0;

}

### SCREEN SHOTS OF OUTPUT

****

**EXPERIMENT 1**

## <<Parallel processing in OpenMP>>

**AIM:** To print the number of the threads using OpenMP

**PROGRAM:**

#include<omp.h>

#include<stdio.h>

int main()

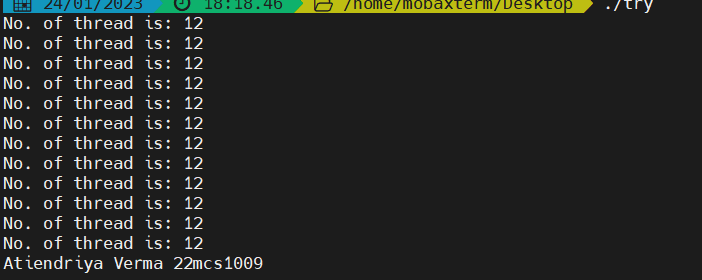
{

#pragma omp parallel printf("No. of thread is: %d \n", omp\_get\_num\_threads());

return 0;

}

### SCREEN SHOTS OF OUTPUT



EXERCISE PROBLEM

To write a program on OpenMP to calculate the body mass index

(BMI), check for the master thread, and display the processor number.

Program

#include<stdio.h>

#include<omp.h>

int main()

{

int h,w,tid,BMI,th;

printf("Enter the value of height and Weight ");

scanf("%d%d",&h,&w);

BMI =((w\*703)/(h\*h));

printf("\nBMI is %d",BMI);

#pragma omp parallel

{

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

th =omp\_get\_num\_procs();

printf("\nMy thread is %d:",tid);

if(tid==0)

{

printf("\nI am The Master Thread");

printf("\nNumber Of thread is %d:",omp\_get\_num\_threads());

}

printf("\nNumber Of Processer Available %d:",th);

}

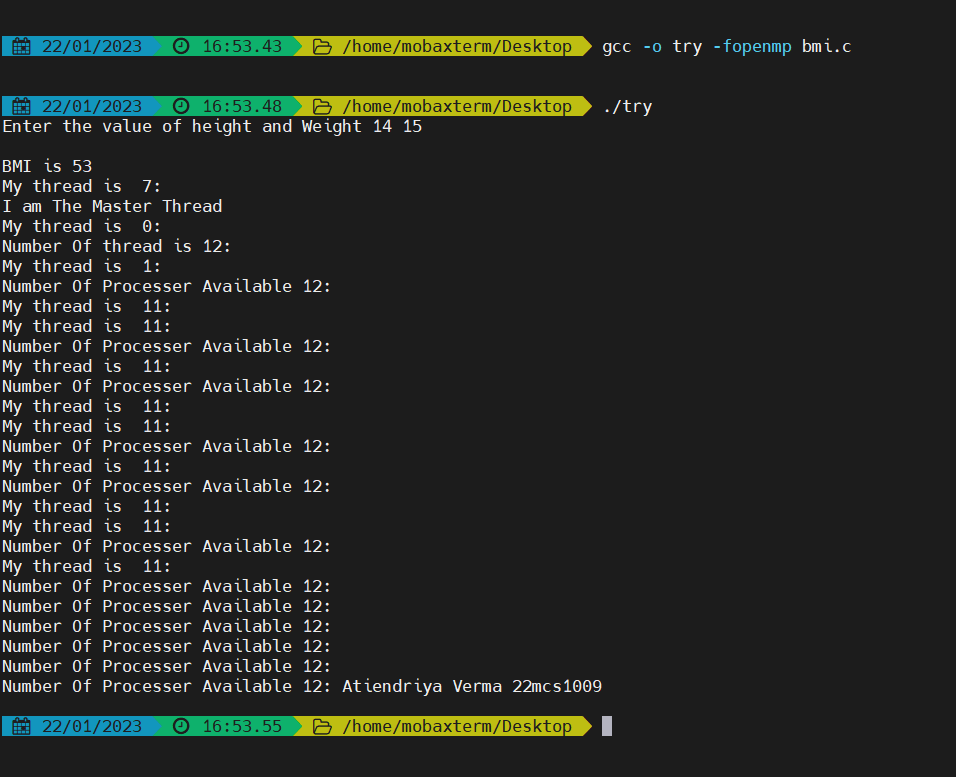
printf(“\n Atiendriya Verma 22MCS1009”);

return (0);

}

### 

### SCREEN SHOTS OF OUTPUT



## EXPERIMENT 2

**<<****Parallel region constructs>>**

**AIM:** To write a program for the sum of the array

**PROGRAM:**

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

int main()

{

int a[5],b[5],sum[5],mult[5];

int i;

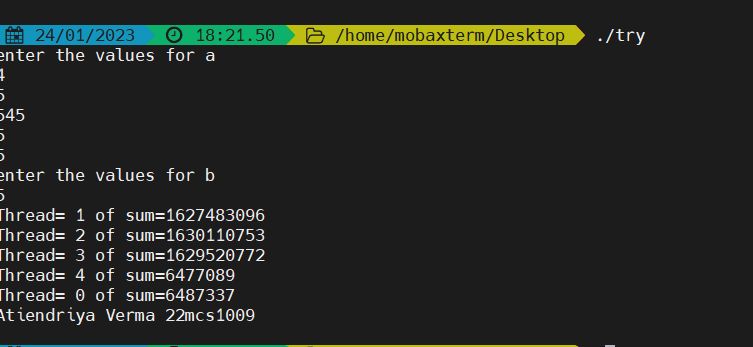
printf("enter the values for a\n"); for(i=0;i<5;i++) scanf("%d",&a[i]); printf("enter the values for b\n"); for(i=0;i<5;i++) scanf("%d",&b[i]); #pragma omp parallel for for(int i=0;i<5;i++){ sum[i]=a[i]+b[i]; printf("Thread= %d of sum=%d \n",omp\_get\_thread\_num(),sum[i]); }

printf(“Atiendriya Verma 22mcs1009);

return 0;

}

### SCREEN SHOTS OF OUTPUT



EXERCISE PROBLEM

To write a program for simple interest calculation

#### Program

#include<omp.h> #include<stdio.h> int main(){ int tid,si,p,r,t;

printf("enter the principle amount: \n"); printf("enter the rate: \n"); printf("enter the time period: \n");

scanf("%d%d%d",&p,&r,&t);

#pragma omp parallel

{

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

si=(p\*r\*t)/100; printf("my thread is %d\n",tid); if(tid==0) printf("the master thread is %d with si=%d\n",tid,si); printf("the thread is %d with simple interest=%d\n",tid,si);

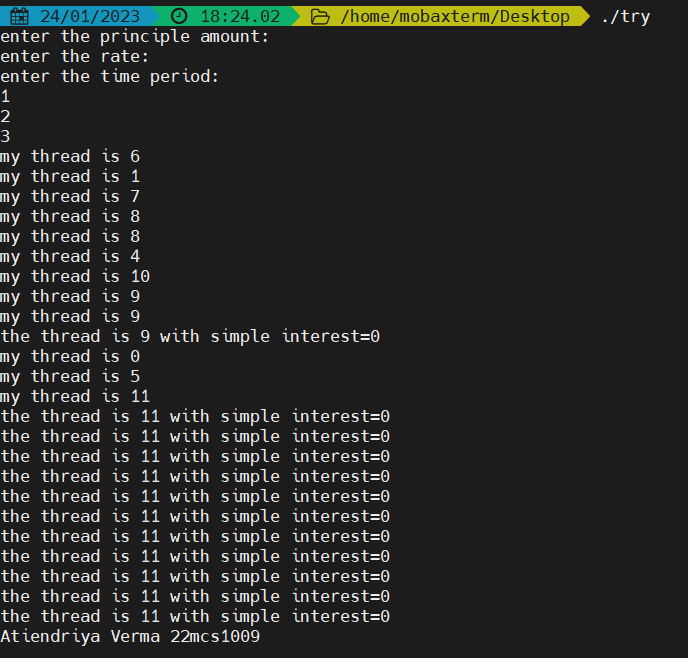
}

Printf(“Atiendriya Verma22MCS1009);

return 0;

}

### SCREEN SHOTS OF OUTPUT



**EXPERIMENT 3**

## <<Parallel sections in OpenMP>>

**AIM:** To write a program for doing different operations in parallel sections

**PROGRAM:**

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

int main()

{

int a[5],b[5],c[5],d[5],e[5],i; for(i=0;i<5;i++)

{

a[i]=i\*2; b[i]=i+10;

}

#pragma omp parallel sections

{

//sum of two array #pragma omp section

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

{c[i]=a[i]+b[i]; printf("the thread number is: %d and ADD of two array is %d \n", omp\_get\_thread\_num(), c[i]);

}

//multiplication of two array #pragma omp section

for(int i=0;i<5;i++) { d[i]=a[i]\*b[i]; printf("the thread number is: %d and MULT of two array is %d\n",omp\_get\_thread\_num(),d[i]);

}

//substraction of two array #pragma omp section

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

{

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

printf("the thread number is: %d and SUB of two array is %d\n",omp\_get\_thread\_num(),e[i]);

}

}

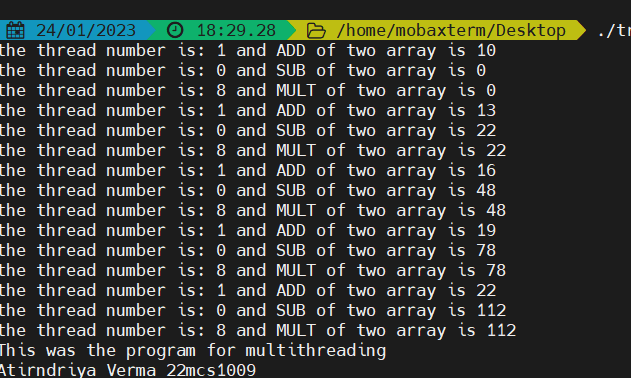
printf("This was the program for multithreading\n");

printf(“Atiendriya Verma 22mcs1009”);

return 0;

}

### SCREEN SHOTS OF OUTPUT



**Election commission program**

**EXERCISE PROBLEM**

Use thread 1 to print eligible people and thread 0 for printing not eligible candidate

### Program

#include<omp.h> #include<stdio.h> int main() { int i,tid; int age[10]; for(i=0;i<=3;i++)

{

printf("enter age of three applicants: \n"); scanf("%d",&age[i]);

}

#pragma omp parallel

{

int tid=omp\_get\_thread\_num(); if(tid==1) {

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

{

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

{

printf("Thread=%d and ELIGIBLE APPLICANT is %d years old\n",tid,age[i]);

}

}

}

else if(tid==0)

{

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

{

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

{

printf("Thread=%d and APLLICANT NOT ELIGIBLE is %d years old\n",tid,age[i]);

}

}

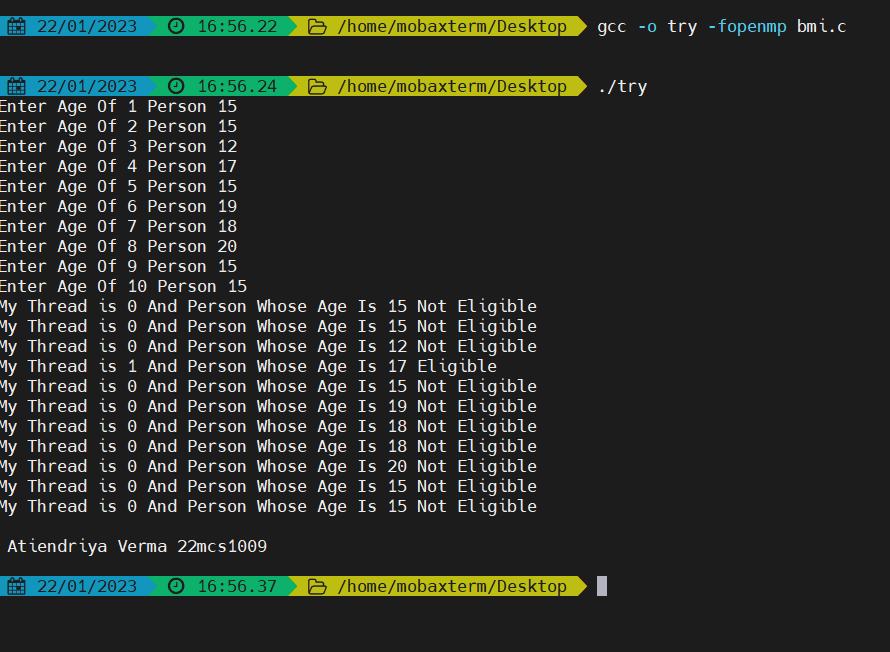
}

Printf(“Atiendriya Verma 22mcs1009”);

} return 0;

}

### SCREEN SHOTS OF OUTPUT



**EXPERIMENT 4**

## <<Date and time function>>

**AIM:** To print day and time using the time function

**PROGRAM:**

#include<stdio.h>

#include<omp.h> #include<time.h> int main() { int tid; time\_t t;

#pragma omp parallel {tid=omp\_get\_thread\_num(); printf("thread =%d \n",tid); if(tid==0)

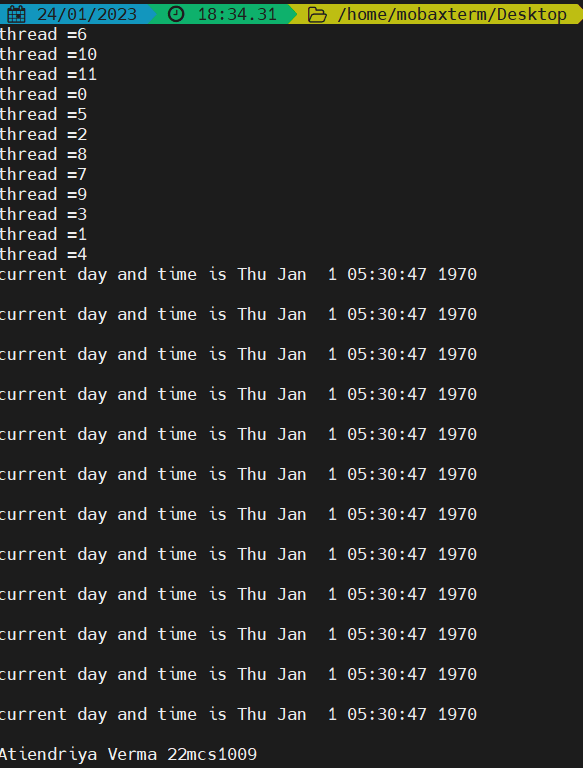
printf("I am the master thread %d, and current day and time stored is %s \n",tid,ctime(&t)); printf("current day and time is %s \n",ctime(&t));}

printf(“Atiendriya Verma 22mcs1009”):

return 0;

}

### SCREEN SHOTS OF OUTPUT



AIM: To print the start time, end time, and execution time of a program

Program

#include<stdio.h>

#include<omp.h> #include<time.h> int main() {

double start,end,etime; int a[5],b[5],c[5],d[5],e[5]; int i;

for(i=0;i<5;i++) {a[i]=i\*2; b[i]=i+10;

}

//start time of program start=omp\_get\_wtime();

printf("the start world clock time is %f\n",start);

#pragma omp parallel sections

{

//sum of two array #pragma omp section for(int i=0;i<5;i++) {c[i]=a[i]+b[i];

printf("the thread number is: %d and ADD of two array is %d\n",omp\_get\_thread\_num(),c[i]); } //multiplication of two array #pragma omp section for(int i=0;i<5;i++) { d[i]=a[i]\*b[i];

printf("the thread number is: %d and MULT of two array is %d\n",omp\_get\_thread\_num(),d[i]);}

//subtraction of two array #pragma omp section for(int i=0;i<5;i++) { e[i]=a[i]\*b[i];

printf("the thread number is: %d and SUB of two array is %d\n",omp\_get\_thread\_num(),e[i]); }

}

//end time of program end=omp\_get\_wtime();

printf("the end world clock time is %f\n",end);

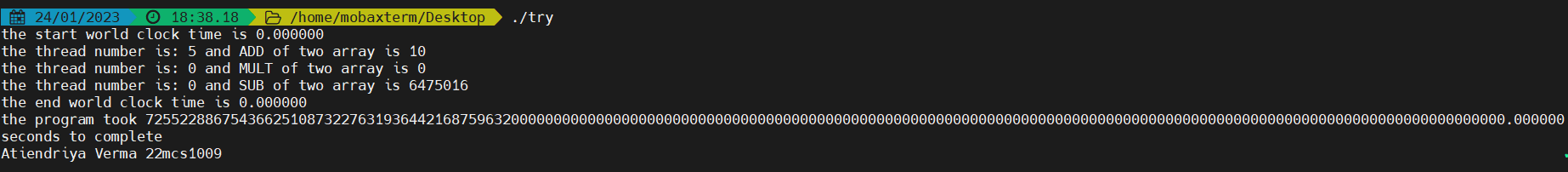
//exection time etime=end-start;

printf("the program took %f seconds to complete\n",etime);

printf(“Atiendriya Verma 22mcs1009);

return 0;

}

SCREEN SHOTS OF OUTPUT 

**EXERCISE PROBLEM**

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 task in parallel and display the result with thread id. Use separate sections to perform each operation. Get as input the name, register number, and the pay package of the student selected for a job in the particular organization. Display the total number of students selected in each company. Display the average pay Code:

#include<stdio.h>

#include<omp.h>

#include<string.h> #include<time.h> struct student

{

char name[50]; int pack; int reg\_no; int company; } studentGroup[10]; int main() {

int a,g,s,it,total=0;

char \*companies[] ={"Amazone", "Google", "Shell", "Intel"};

printf(" select the company: \n 1:Amazone \n 2:Google \n 3:Shell \n 4:Intel \n"); for(int i=0;i<3;i++)

{

printf("enter name of the students: \n"); scanf("%s",(char\*) &studentGroup[i].name); printf("enter your registraion number: \n"); scanf("%d", &studentGroup[i].reg\_no); printf("enter the pay chk for the student: \n"); scanf("%d",&studentGroup[i].company);

} float start,end,etime; start= omp\_get\_wtime();

printf("the start world clock time is: %f \n",start); for(int k=0;k<3;k++)

{

if(studentGroup[k].company==1) a+=1;

else if(studentGroup[k].company==2) g+=1;

else if(studentGroup[k].company==3) s+=1;

else if(studentGroup[k].company==1) it+=1; }

printf("total number of studnts selected => \n"); printf("Amazone: %d \n",a); printf("Google: %d \n",g); printf("Shell: %d \n",s); printf("Intel: %d \n",it);

#pragma omp parallel sections

{

#pragma omp section

{ for(int j=0; j<3;j++) total += studentGroup[j].pack;

printf("Average pay check of students in is: %f\n",total/10);

}

#pragma omp section

{printf("number of processor is: %d\n", omp\_get\_num\_procs());

} }

end= omp\_get\_wtime();

printf("the end world clock time is: %f \n",end); etime =end-start;

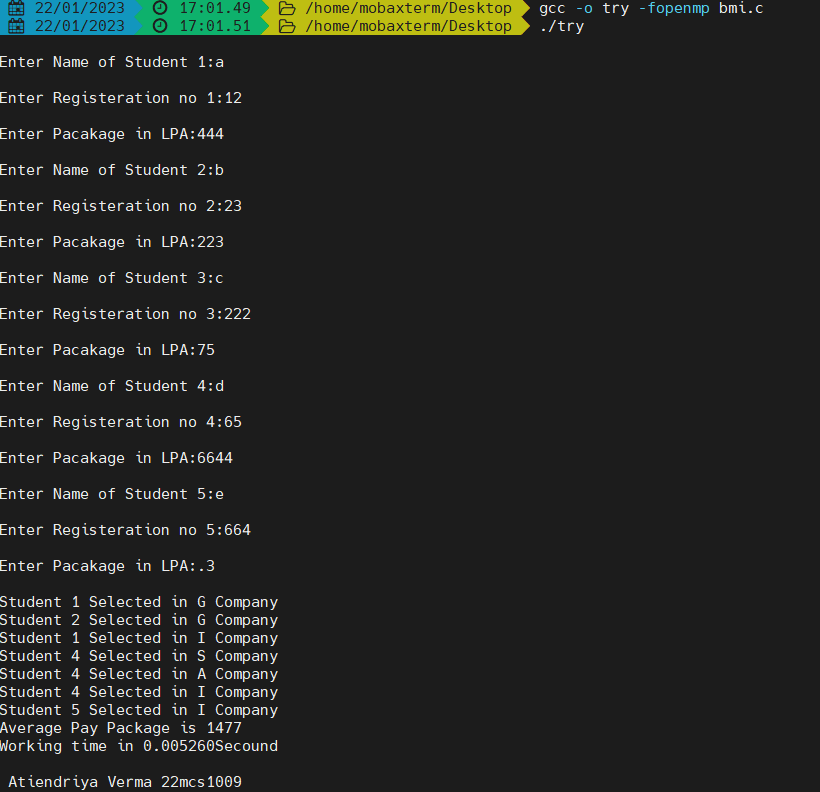
printf("time of execution is %f seconds \n",etime);

printf(“Atiendriya Verma 22mcs1009”):

return 0;

}

### SCREEN SHOTS OF OUTPUT



**EXPERIMENT 5**

## <<Private and shared space>>

**AIM:** To print the variable value present in private and shared space

**PROGRAM:**

#include<omp.h> #include<stdio.h> int main()

{ int x=9;

#pragma omp parallel for private(x)

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

{ x=i;

printf("value of x in parallel section under private space is %d and thread is %d \n", x,omp\_get\_thread\_num());

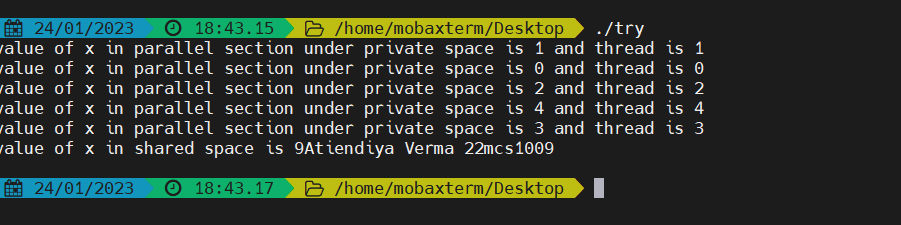
}

printf("value of x in shared space is %d", x);

printf(“Atiendriya Verma 22mcs1009”);

}

### SCREEN SHOTS OF OUTPUT

****

**AIM: To limit the number of threads**

### Program

#include<omp.h> #include<stdio.h> int main()

{ int x=9;

#pragma omp parallel for private(x) num\_threads(3)

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

{ x=i;

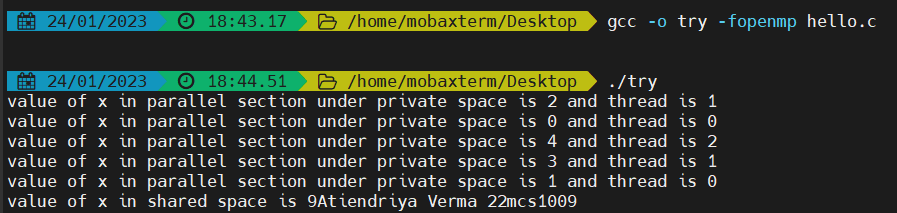
printf("value of x in parallel section under private space is %d and thread is %d \n", x,omp\_get\_thread\_num());

}

printf("value of x in shared space is %d", x);

}

### SCREEN SHOTS OF OUTPUT

****

**AIM: To retain the last iteration value**

**Program**

#include<omp.h> #include<stdio.h> int main() { int x=9;

#pragma omp parallel for lastprivate(x) for(int i=0;i<5;i++)

{

printf("value of x before assignment is %d\n",x); x=i;

printf("value of x in parallel section under private space is %d and thread is %d \n", x,omp\_get\_thread\_num());

}

printf("value of x after assignment is %d\n",x);

printf("value of x in shared space is %d\n", x);

printf(“Atiendriya Verma 22mcs1009”);

}

### SCREEN SHOTS OF OUTPUT

A picture containing graphical user interface

Description automatically generated

**AIM: To take first assigned value on private space**

**Program**

#include<omp.h> #include<stdio.h> int main() { int x=9;

#pragma omp parallel for firstprivate(x) for(int i=0;i<5;i++)

{

printf("value of x before assignment is %d\n",x); x=i;

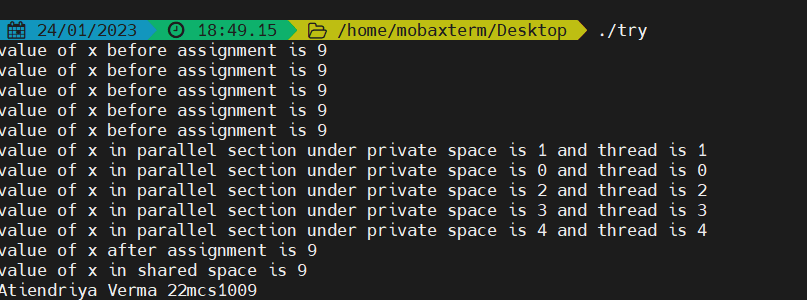
printf("value of x in parallel section under private space is %d and thread is %d \n", x,omp\_get\_thread\_num());

}

printf("value of x after assignment is %d\n",x); printf("value of x in shared space is %d\n", x);

}

### SCREEN SHOTS OF OUTPUT



**EXERCISE PROBLEM:**

**Design a math application To accept an integer as input and output whether it is a rational number, or perfect number, or a prime number**

Code:

#include<omp.h>

#include<stdio.h>

#include<math.h> #include<bool.h> bool primeNo(int n)

{int i,flag=0; if(n==0 || n==1) flag=1;

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

{

if(n%i==0) //if n is divisible by i then its not a prime number

{

flag=1; break;

}

}

//prime number program

if(flag==0) return true; else return false;

}

bool perfectNo(int n) {//perfect number program int i,rem,sum=0; for(i=1;i<n;i++)

{

rem=n%i; //finding the multiples of number if(rem==0) sum=sum+i;

}

if(sum==n&&n!=1)

return true; else return false;

}

bool rationalNo(int p, int q)

{//rational number

int i,p,q;

printf("enter the numerator and denominator value: \n"); scanf("%d \t %d",&p,&q);

if(q!=0) return true;

}

int main()

{

//accepting and interger

int n;

printf("enter a positive integer value: \n"); scanf("%d",&n); bool rationalNo, primeNo, perfectNo;

#pragma omp parallel section num\_threads(3) private(rationalNo) firstprivate(primeNo) lastprivate(perfectNo)

{

#pragma omp section {rationalNo = TRUE; printf("It's a rational number \n");

}

#pragma omp section {is\_primeNo = primeNo(n); if(is\_primeNo) printf("It's a prime number \n"); else printf("It's not a prime number \n");

}

#pragma omp section {is\_perfectNo = perfectNo(n); if(is\_perfectNo) printf("It's a perfect number \n"); else printf("It's not a perfect number \n");

}

}

}

**OUTPUT-** Enter a value 10

The number is rational

The number is not perfect

The number is not prime



**EXPERIMENT 6**

## <<Static, dynamic, and guided scheduling>>

**AIM:** Write a program for static scheduling using OpenMP

**PROGRAM:**

#include<omp.h> #include<stdio.h> int main() {

int i;

#pragma omp parallel num\_threads(3)

{

#pragma omp for schedule(static,3) for(int i=0;i<20;i++)

printf("iteration %d run by thread %d\n",i,omp\_get\_thread\_num());

} return 0;

}

### SCREEN SHOTS OF OUTPUT

Text

Description automatically generated

AIM: Write a program for dynamic scheduling using OpenMP

Program

#include<omp.h> #include<stdio.h> int main() {

int i;

#pragma omp parallel num\_threads(3)

{

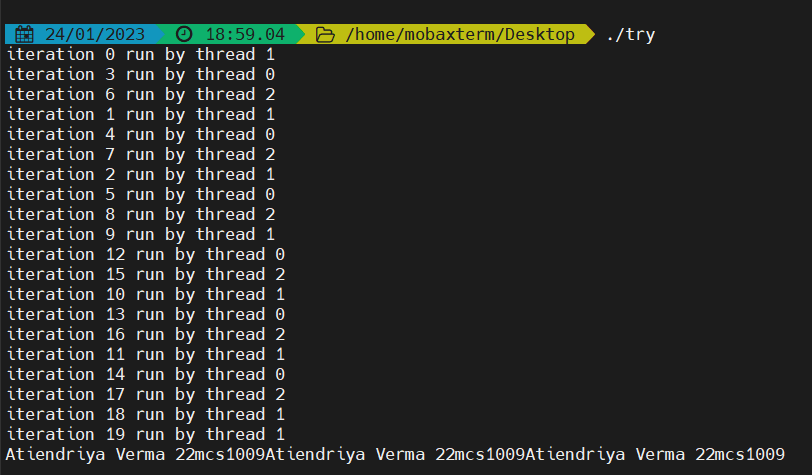
#pragma omp for schedule(dynamic,3) for(int i=0;i<20;i++)

printf("iteration %d run by thread %d\n",i,omp\_get\_thread\_num());

} return 0;

}

### SCREEN SHOTS OF OUTPUT



AIM: Write a program for guided scheduling using OpenMP

Program

#include<omp.h> #include<stdio.h> int main() {

int i;

#pragma omp parallel num\_threads(3)

{

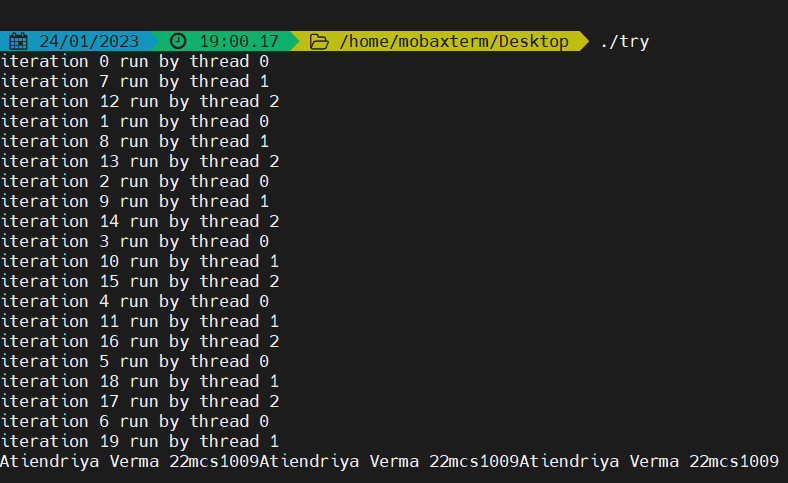
#pragma omp for schedule(guided,3) for(int i=0;i<20;i++)

printf("iteration %d run by thread %d\n",i,omp\_get\_thread\_num());

} return 0;

}

### SCREEN SHOTS OF OUTPUT



**Aim**- The quality checking unit in the toy modeling unit has incremental counter and counts the tested toy from 0 to 256. Once the counter reaches the max value all tested toy will transferred to dispatching unit in which this counter decrements from the maximum of 256 and reaches to zero. Use last private to get max value.

**Program:**

#include<stdio.h>

#include<omp.h>

void main(){

int counter = 0;

#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);

}

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

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

{

counter = 256 - i;

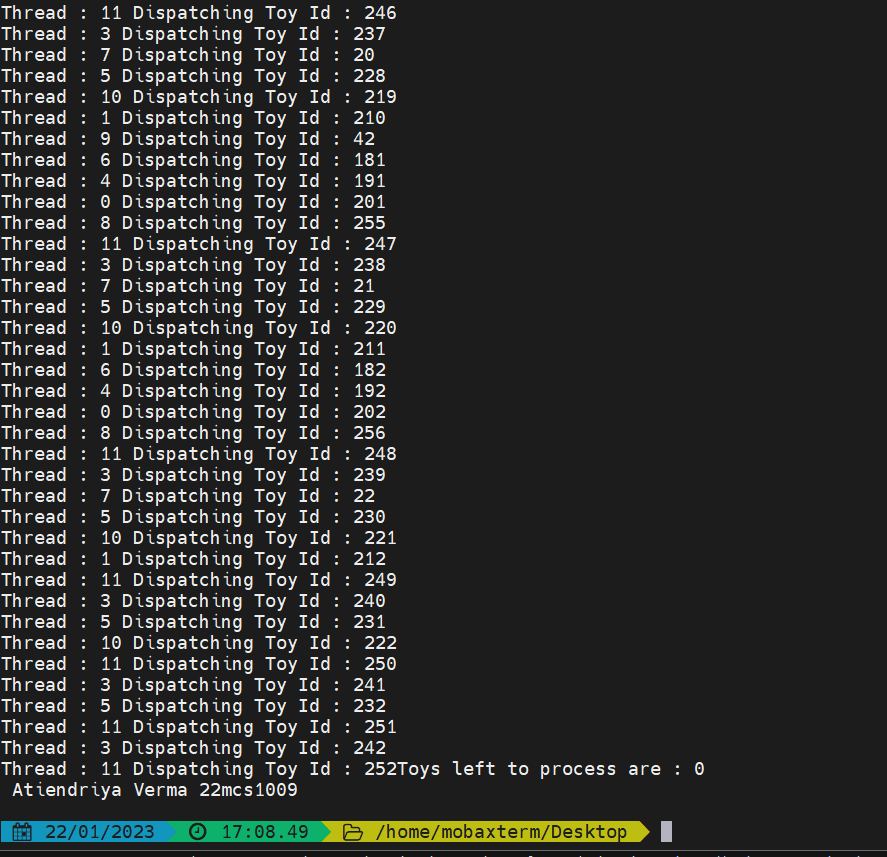
printf("\nThread : %d Dispatching Toy Id : %d", omp\_get\_thread\_num(), i);

}

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

}

**SCREEN SHOTS OF OUTPUT**

****

**EXPERIMENT 7**

## <<Ordered, unordered scheduling, and locks>>

**AIM:** To write a program for printing threads in an ordered fashion

**PROGRAM:**

#include<omp.h> #include<stdio.h> int main()

{

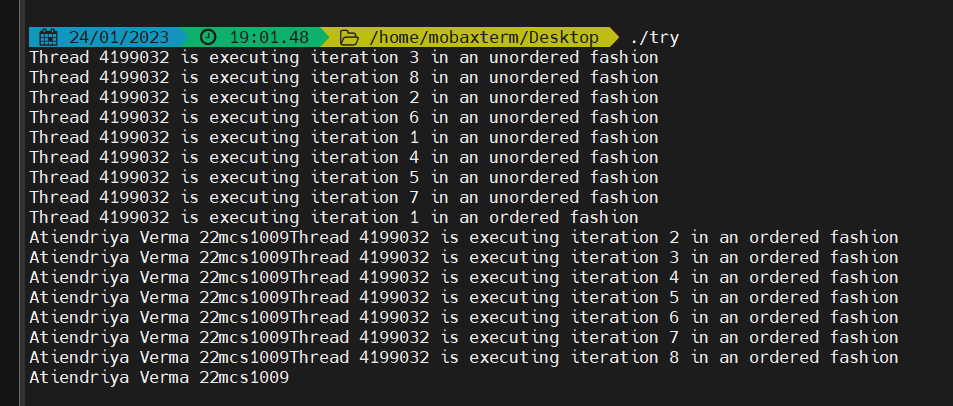
#pragma omp parallel for ordered for(int i=1;i<=8;i++) {

printf("Thread %d is executing iteration %d in an unordered fashion\n", omp\_get\_thread\_num,i);

#pragma omp ordered

printf("Thread %d is executing iteration %d in an ordered fashion\n", omp\_get\_thread\_num,i); } }

### SCREEN SHOTS OF OUTPUT



### AIM: To write a program for using locks in shared space

Program

#include<omp.h> #include<stdio.h> int main() { int id,i;

omp\_lock\_t mylock; omp\_init\_lock(&mylock); #pragma omp parallel

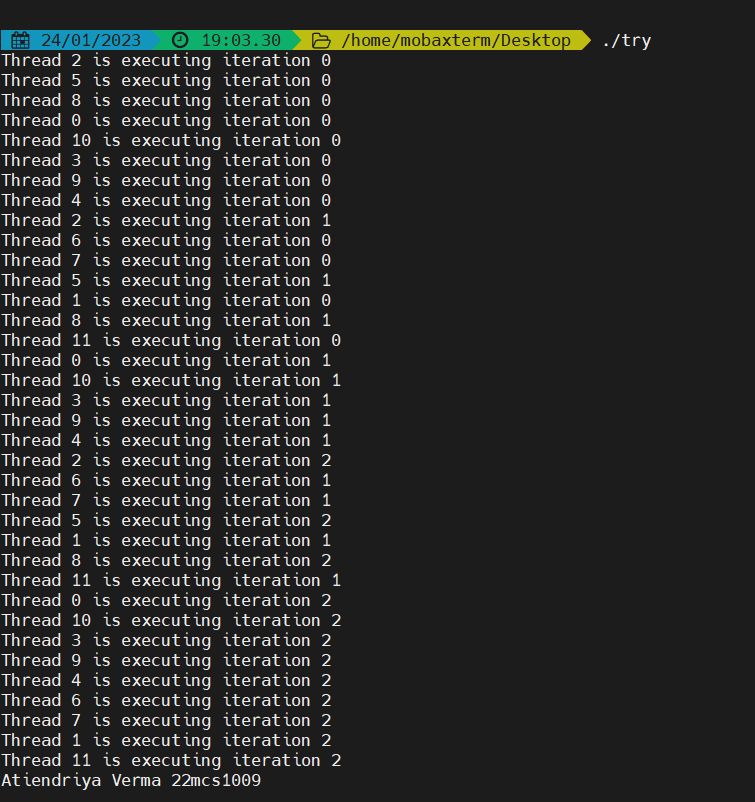
{ id=omp\_get\_thread\_num(); #pragma omp parallel for for(int i=0;i<3;i++) { omp\_set\_lock(&mylock);

printf("Thread %d is executing iteration %d\n", id,i); omp\_unset\_lock(&mylock); }

} omp\_destroy\_lock(&mylock);

}

### SCREEN SHOTS OF OUTPUT



**EXERCISE PROBLEM:** Design a parallel program to print ‘n’ even and odd numbers in a sequential fashion of threads.

Program:

#include<omp.h> #include<stdio.h> int main() { int n;

printf("enter the number : \n"); scanf("%d",&n);

#pragma omp parallel for ordered for(int i=0;i<=n;i++) { if(i%2==0){ #pragma omp ordered

printf("Thread %d is executing even number %d in an ordered fashion\n", omp\_get\_thread\_num(),i); } else{

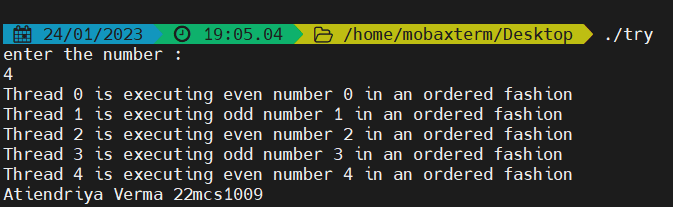
#pragma omp ordered

printf("Thread %d is executing odd number %d in an ordered fashion\n", omp\_get\_thread\_num(),i);

} } return 0;

}

### SCREEN SHOTS OF OUTPUT



1. A contest is being held for technoVIT. Students can register, if they want, they can unregister. Registered students (registration number:9,3,2..) is stored in an array. Only one student can register or unregister at a time.

Program:

#include <stdio.h>

#include <omp.h>

#include <unistd.h>

int registration[1000];

omp\_lock\_t lock1,lock2;

void main()

{

int num, iterate = 1;

omp\_init\_lock(&lock1);

omp\_init\_lock(&lock2);

while (iterate)

{

printf("Enter 1 for registering,2 for unregistering and 3 for viewing details and 0 to stop.\n");

scanf("%d", &num);

if (num == 0)

{

iterate = 0;

break;

}

{

#pragma omp parallel sections

{

#pragma omp section

{

if (num == 1)

{

omp\_set\_lock(&lock1);

int id;

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

scanf("%d", &id);

sleep(2);

registration[id] = 1;

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

omp\_unset\_lock(&lock1);

}

}

#pragma omp section

{

if (num == 2)

{

omp\_set\_lock(&lock2);

int id;

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

scanf("%d", &id);

sleep(2);

registration[id] = 0;

printf("\n UnRegistered : %d\n", id);

omp\_unset\_lock(&lock2);

}

}

}

if (num == 3)

{

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

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

{

if (registration[i] == 1)

{

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

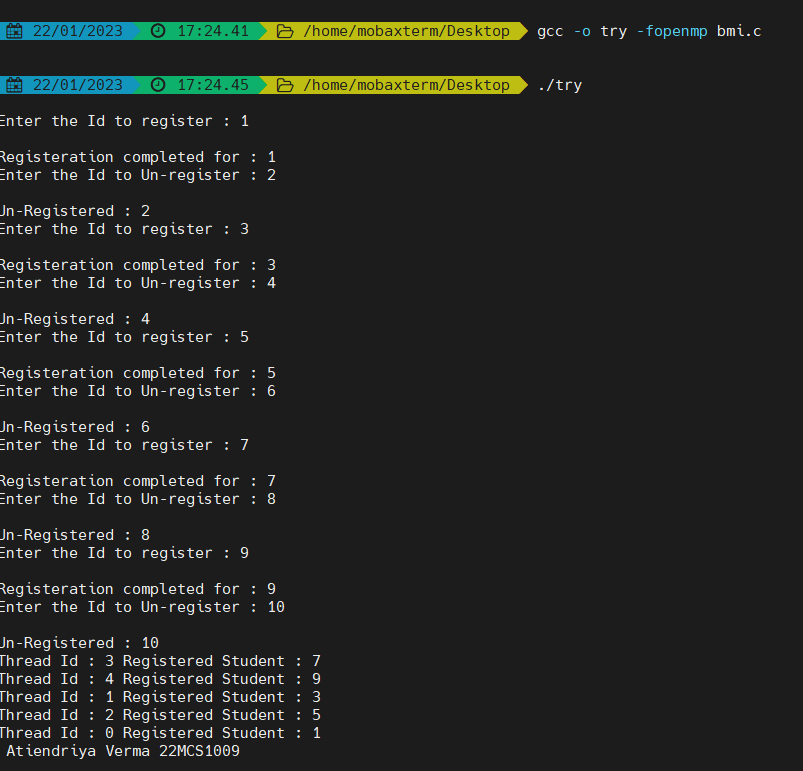
}

}

}

}

**SCREEN SHOTS OF OUTPUT**



## EXPERIMENT 8

**<<****Open MP barrier and File operations>>**

**AIM:** To write a program for printing threads in an ordered fashion

**PROGRAM:**

#include<stdio.h> int main() { char name[50]; int marks,i,num;

printf("enter the num of students: \n"); scanf("%d",&num); FILE \*fptr;

fptr =fopen("C:/Users/HP/OneDrive/Desktop/BDA\_SEM1/COA/lab/stu.txt","w"); if(fptr==NULL) printf("ERROR\n"); else

{

for(i=0;i<num;++i){ printf(" student %d \t enter name: \n",i+1); scanf("%s",name); printf("enter marks: \n"); scanf("%d",&marks);

fprintf(fptr," in name:%s marks=%d", name,marks);

} } fclose(fptr);

}

Text, timeline

Description automatically generated

**Output:**

enter the num of students:

2

student 1 enter name: Shubham enter marks: 31

student 2 enter name: Harshad enter marks: 32

AIM: To open a file using OpenMP with a thread barrier Program:

#include<stdio.h> #include<omp.h> int main() { char name[50]; int num\_threads, tid;

FILE \*fptr;

Fptr=fopen("C:/Users/HP/OneDrive/Desktop/BDA\_SEM1/COA/lab/stu.txt","w");

if(fptr==NULL) printf("ERROR\n");

#pragma omp parallel

{

int tid=omp\_get\_thread\_num(); if(tid==0)

num\_threads=omp\_get\_num\_threads();

#pragma omp barrier

fprintf(fptr,"hello world! from thread %d of %d /n", tid, num\_threads);

} fclose(fptr);

}

### Output

hello world! from thread 2 of 8 hello world! from thread 3 of 8 hello world! from thread 4 of 8 hello world! from thread 1 of 8 hello world! from thread 6 of 8 hello world! from thread 0 of 8 hello world! from thread 5 of 8 hello world! from thread 7 of 8

EXERCISE PROBLEM

AIM: Write a parallel program using OpenMP to implement the following series ½+1/4+1/8….. find the sum of the series and print it along with thread id and last value in the series of given “N” values. Print output in file “series.txt”.

Program:

#include<stdio.h>

#include<omp.h>

#include<math.h> #include<stdlib.h> int main()

{

int tid, num\_threads,n; printf("enter the num : \n");

scanf("%d",&n);

double i, sum = 1.0;

#pragma omp parallel for lastprivate(sum)

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

sum+=pow((double)(0.5),(double)(i)); FILE \*fptr;

fptr =fopen("C:/Users/HP/OneDrive/Desktop/BDA\_SEM1/COA/lab/series.txt","w"); if(fptr==NULL) printf("ERROR\n");

#pragma omp parallel

{

int tid=omp\_get\_thread\_num(); if(tid==0)

num\_threads=omp\_get\_num\_threads();

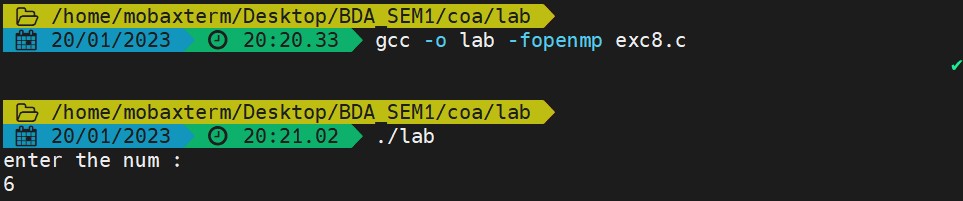
#pragma omp barrier

fprintf(fptr,"sum %.2f from thread %d of %d /n",sum, tid);

} fclose(fptr);

}

**Output:**

sum 0.02 from thread 3 of 0 sum 0.02 from thread 1 of 0 sum 0.02 from thread 7 of 0 sum 0.02 from thread 5 of 0 sum 0.02 from thread 4 of 0 sum 0.02 from thread 2 of 0 sum 0.02 from thread 6 of 0

**EXPERIMENT 9**

## << Matrix multiplication >>

**AIM:** write a parallel program using OpenMP to do matrix multiplication

**PROGRAM:**

#include<stdio.h>

#include<stdlib.h>

#include<omp.h>

#include<sys/time.h> #define n 1000 int a[n][n]; int b[n][n]; int c[n][n]; int main() {

int i,j,k;

struct timeval tv1,tv2; struct timezone tz; double elapsed;

omp\_set\_num\_threads(omp\_get\_num\_procs()); for(i=0;i<3;i++) for(j=0;j<3;j++){ a[i][j]=2; b[i][j]=2;} gettimeofday(&tv1,&tz);

#pragma omp parallel for private(i,j,k) shared(a,b,c) for(i=0;i<3;++i)

for(j=0;j<3;++j) for(k=0;k<3;++k) c[i][j] += a[i][k]\*b[k][j]; gettimeofday(&tv2,&tz);

elapsed=(double)(tv2.tv\_sec-tv1.tv\_sec)+(double)(tv2.tv\_usec-tv1.tv\_usec)\*(1.e-6); printf("elapsed time =%.6f seconds \n",elapsed); for(i=0;i<3;i++)

{

for(j=0;j<3;j++) printf("%d\t",c[i][j]); printf("\n");

}

}

### SCREEN SHOTS OF OUTPUT

Text

Description automatically generated

## <<Title: matrix-vector multiplication>>

**EXERCISE PROBLEM**

**Aim: write a parallel program using OpenMP to do matrix-vector multiplication** Program:

#include<stdio.h>

#include<stdlib.h>

#include<omp.h>

#include<sys/time.h> #define n 1000 int a[n][n]; int b[n][n]; int c[n][n]; int main() {

int i,j,k;

struct timeval tv1,tv2; //under sys/time file struct timezone tz; //ist time zone double elapsed;

omp\_set\_num\_threads(omp\_get\_num\_procs());

//a[i][j] 3\*3 identity matrix for(i=0;i<3;i++) for(j=0;j<3;j++){ a[i][j]=2; b[i][j]=2;}

//b[i][j] 3\*1 identity matrix for(i=0;i<3;i++) for(j=0;j<1;j++){ b[i][j]=2;} gettimeofday(&tv1,&tz); //program starting time #pragma omp parallel for private(i,j,k) shared(a,b,c) for(i=0;i<3;++i) for(j=0;j<3;++j) for(k=0;k<3;++k) c[i][j] += a[i][k]\*b[k][j];

gettimeofday(&tv2,&tz); //program end time

//calculating elapsed time

elapsed=(double)(tv2.tv\_sec-tv1.tv\_sec)+(double)(tv2.tv\_usec-tv1.tv\_usec)\*(1.e-6); printf("elapsed time =%.6f seconds \n",elapsed); for(i=0;i<3;i++){ for(j=0;j<1;j++) printf("%d\t",c[i][j]); printf("\n");}

}

### SCREEN SHOTS OF OUTPUT

Text

Description automatically generated

**EXPERIMENT 10**

## << Vtune Profile Matrix multiplication >>

**AIM:** write a parallel program using OpenMP to do matrix multiplication and evaluate the performance with the Vtune profiler

**PROGRAM:**

#include <iostream>

#include <omp.h>

#include <chrono>

#include <iomanip> #include <Windows.h> using namespace std; #define N 2000 int A[N][N]; int B[N][N]; int C[N][N]; int main() {

int i, j, k;

double elapsed;

omp\_set\_num\_threads(omp\_get\_num\_procs());

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

{

A[i][j] = 2;

B[i][j] = 2;

}

unsigned int start = GetTickCount();

#pragma omp parallel for private(i,j,k) shared(A,B,C) for (i = 0; i < N; ++i) { for (j = 0; j < N; ++j) { for (k = 0; k < N; ++k) {

C[i][j] += A[i][k] \* B[k][j];

}

}

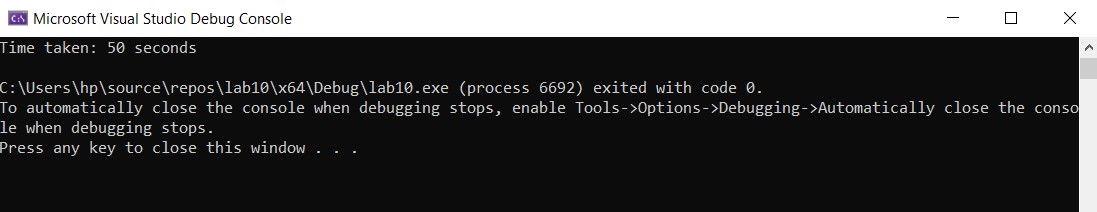
}

unsigned int end = GetTickCount(); cout.setf(ios::fixed);

cout << "Time taken: " << setprecision(3) << (end - start)/1000 << " seconds" << endl;

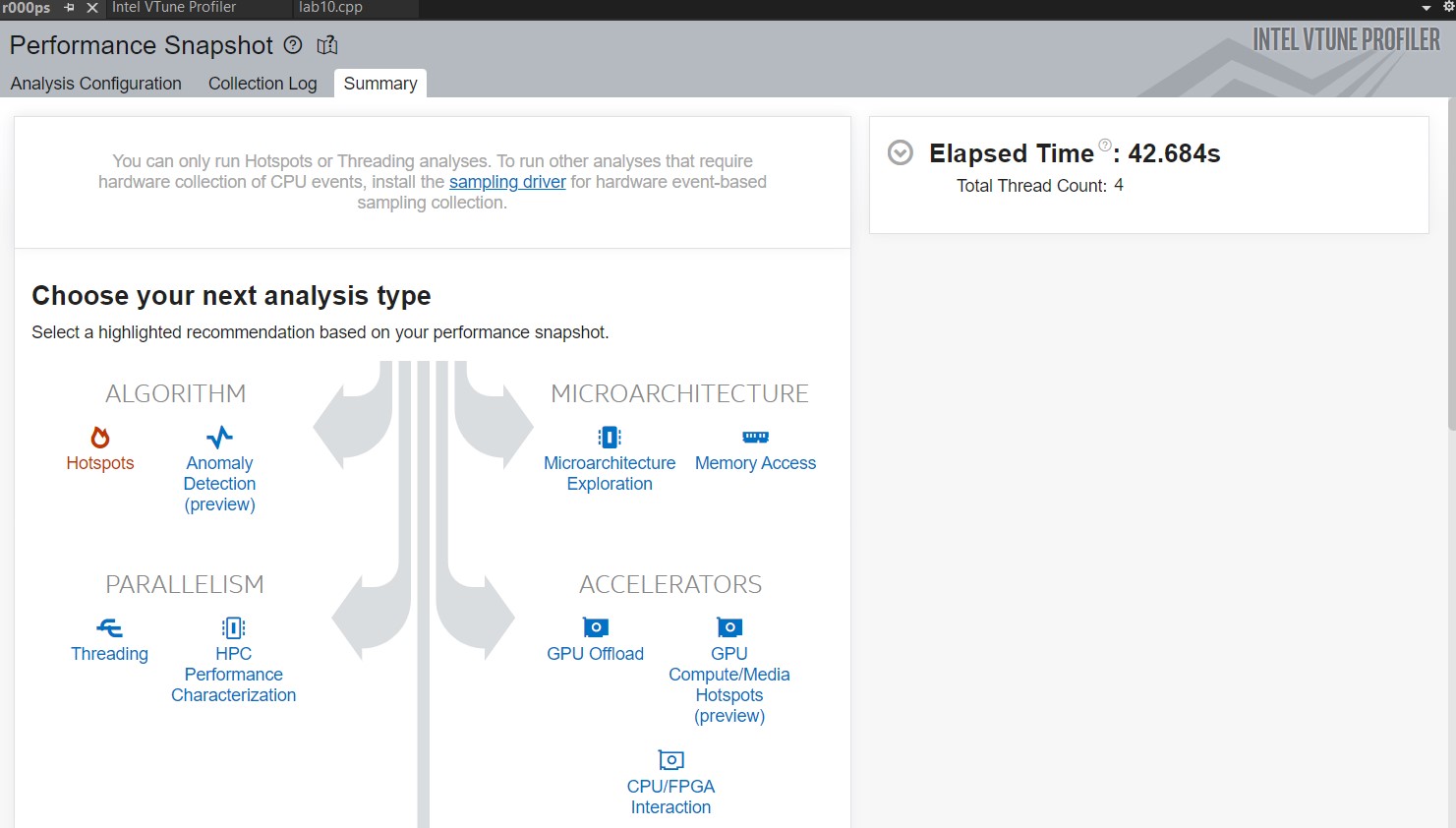
}

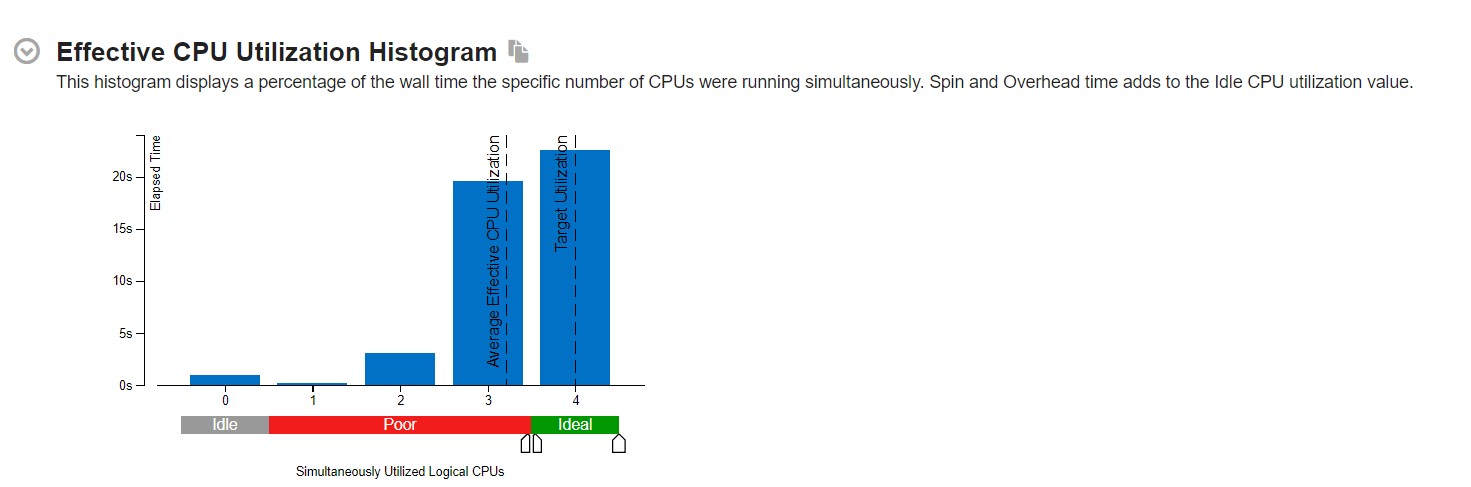
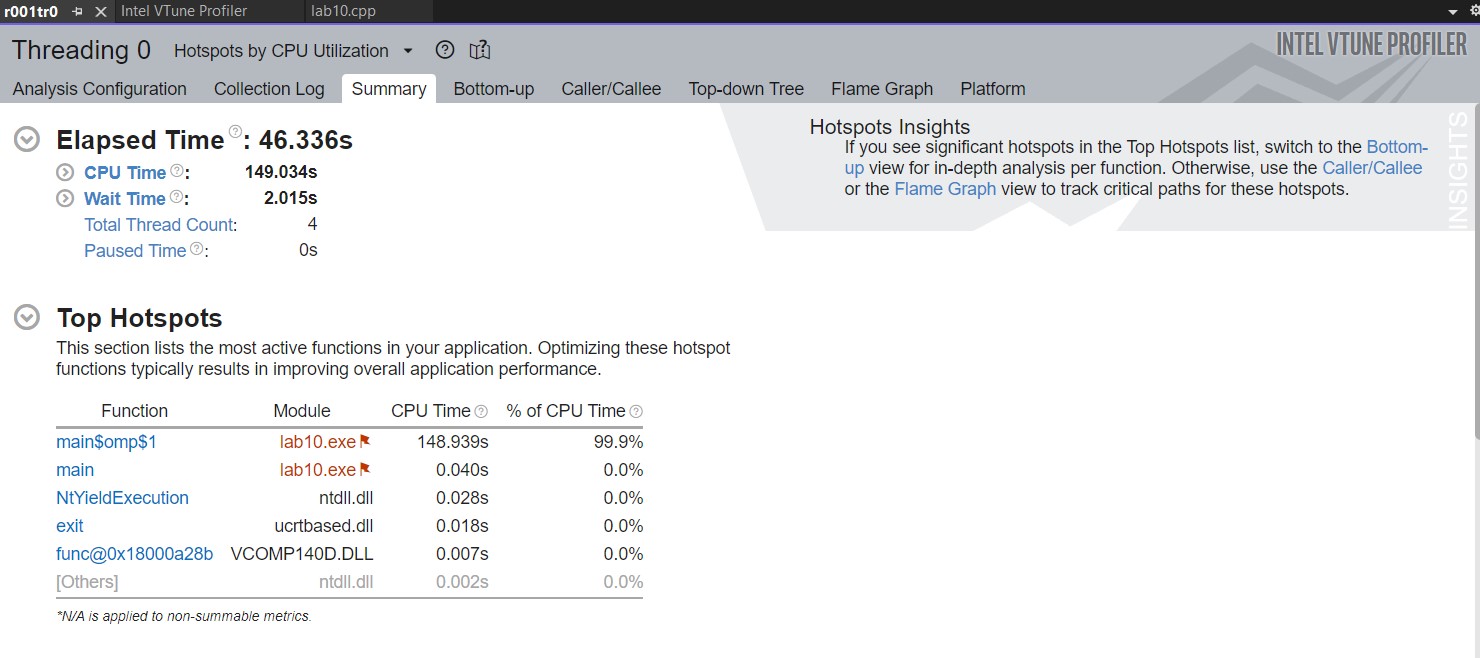
### SCREEN SHOTS OF OUTPUT



The time taken by the system is 42.684 s

Vtune profiler procedure for configuring analysis output:





**EXPERIMENT 11**

## << Vtune Profile Matrix-vector multiplication >>

**AIM:** write a parallel program to do matrix-vector multiplication, and evaluate the performance with the Vtune profiler.

**PROGRAM:**

#include<iostream>

#include<omp.h>

#include<chrono>

#include<iomanip> #include<Windows.h> using namespace std; #define N 2000

int A[N][N]; int B[N][N]; int C[N][N]; int main()

{ int i, j, k;

double elapsed;

omp\_set\_num\_threads(omp\_get\_num\_procs());

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

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

A[i][j] = 2;

if (j==0)

B[i][j] = 2; }

unsigned int start = GetTickCount();

#pragma omp parallel for private(i,j,k) shared(A,B,C) for (i = 0; i < N; ++i) {

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

C[i][j] += A[i][k] \* B[k][j]; }

} }

unsigned int end = GetTickCount(); cout.setf(ios::fixed); cout << "Time taken: " << setprecision(3) << (end - start) / 1000 << " seconds" << endl;

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

for (j= 0; j< 3; j++) {

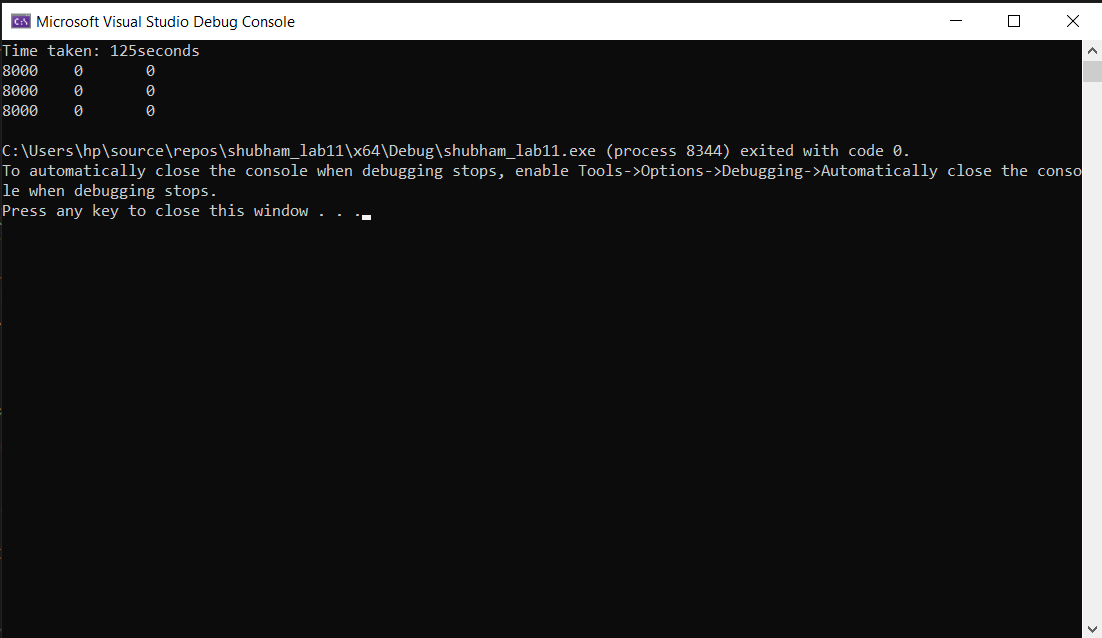
printf("%d\t",C[i][j]);

} printf("\n");

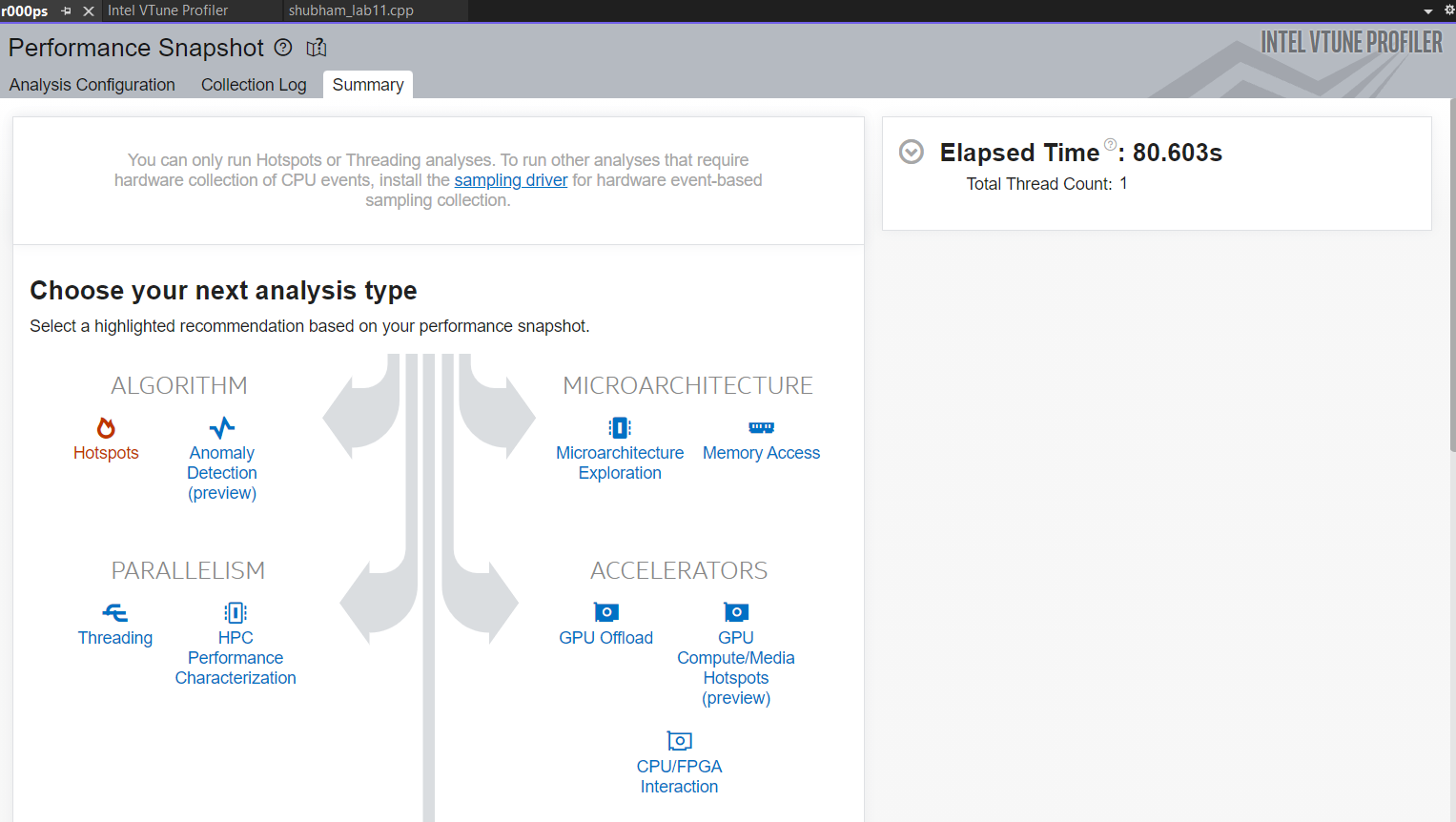
}

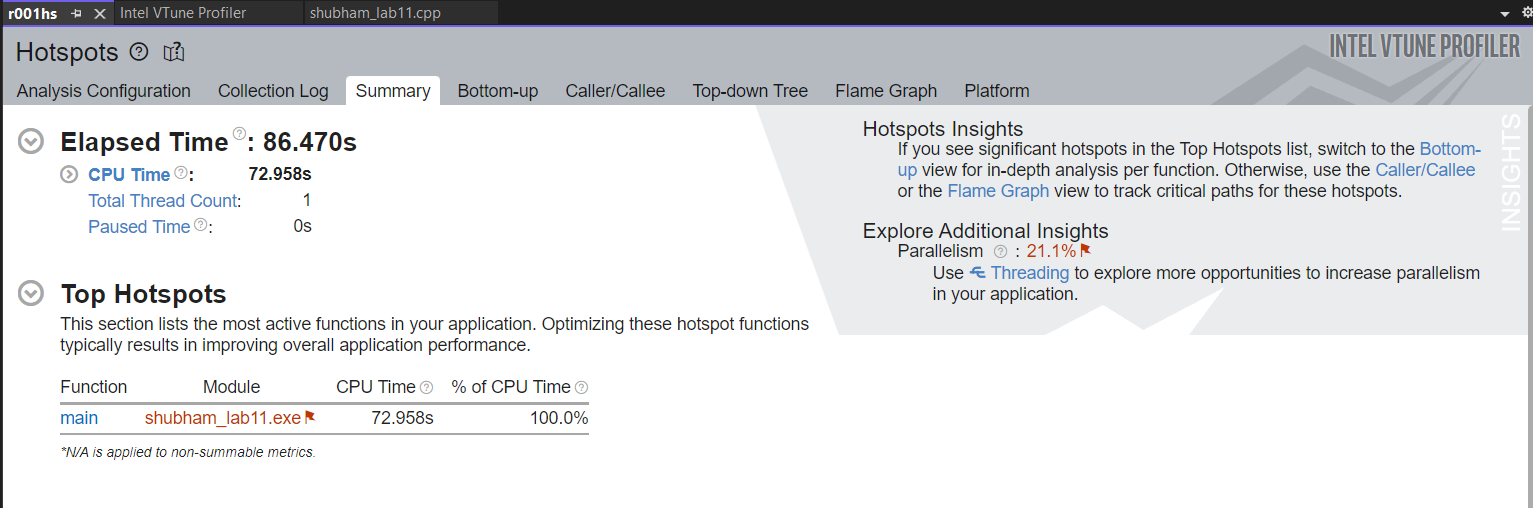
}

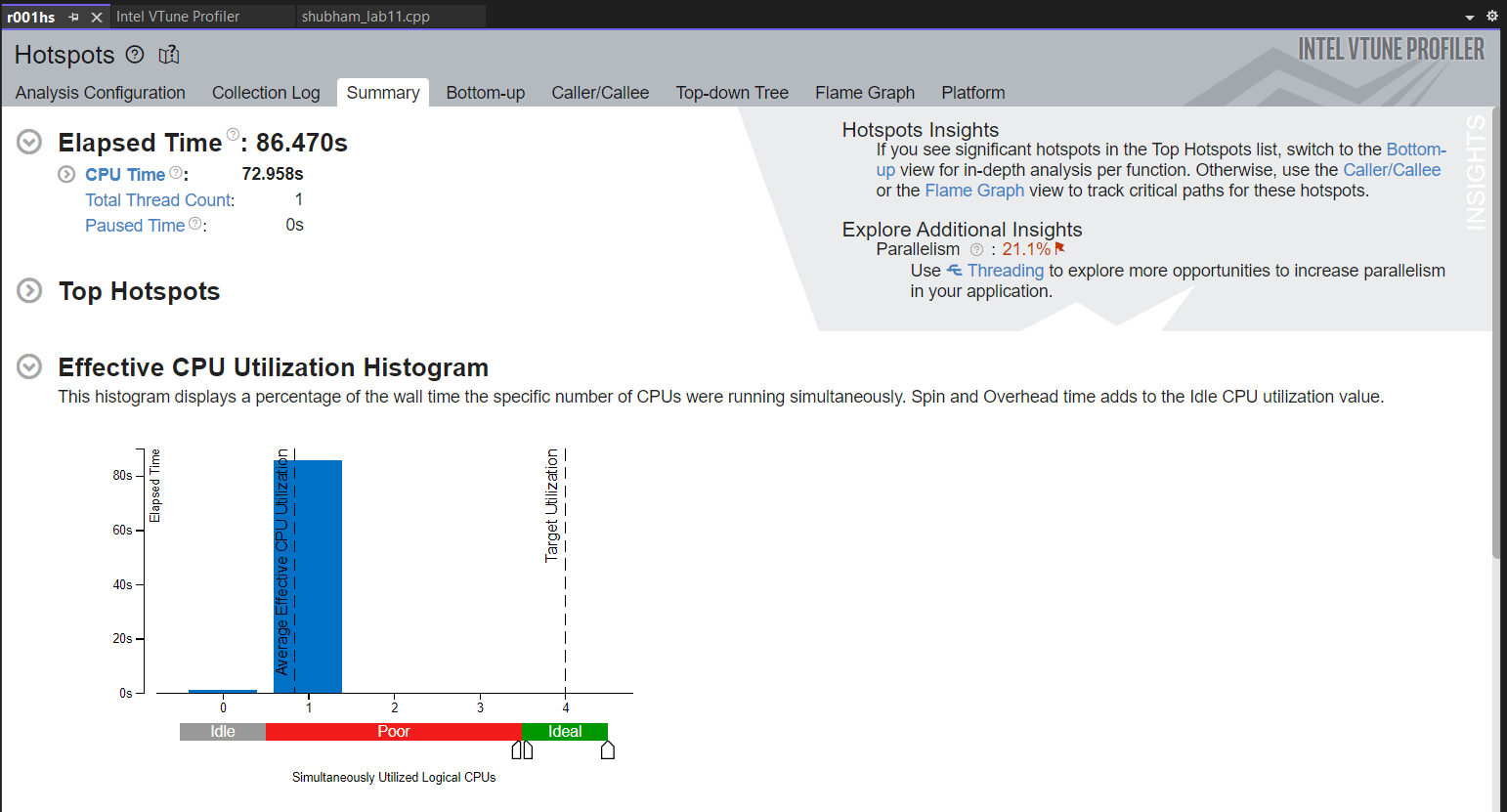
### SCREEN SHOTS OF OUTPUT



Vtune profiler procedure for configuring analysis output:







**EXPERIMENT 12**

**<<** **Vtune Profile of Minimum Spanning Tree using Kruskal Algorithm >>**

**AIM:** write a Parallel Implementation of Minimum Spanning Tree using Kruskal Algorithm and evaluate the performance with the Vtune profiler.

**PROGRAM:**

#include<iostream>

#include<conio.h>

#include<stdlib.h>

#include<omp.h> #include<Windows.h> using namespace std;

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);

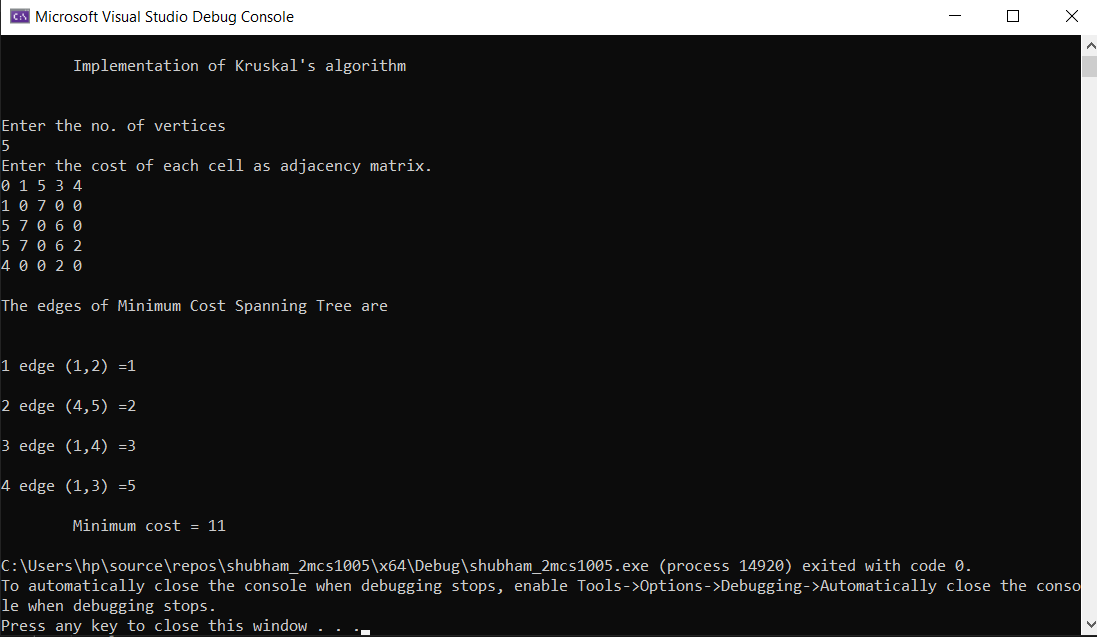
}

Adjacency matrix

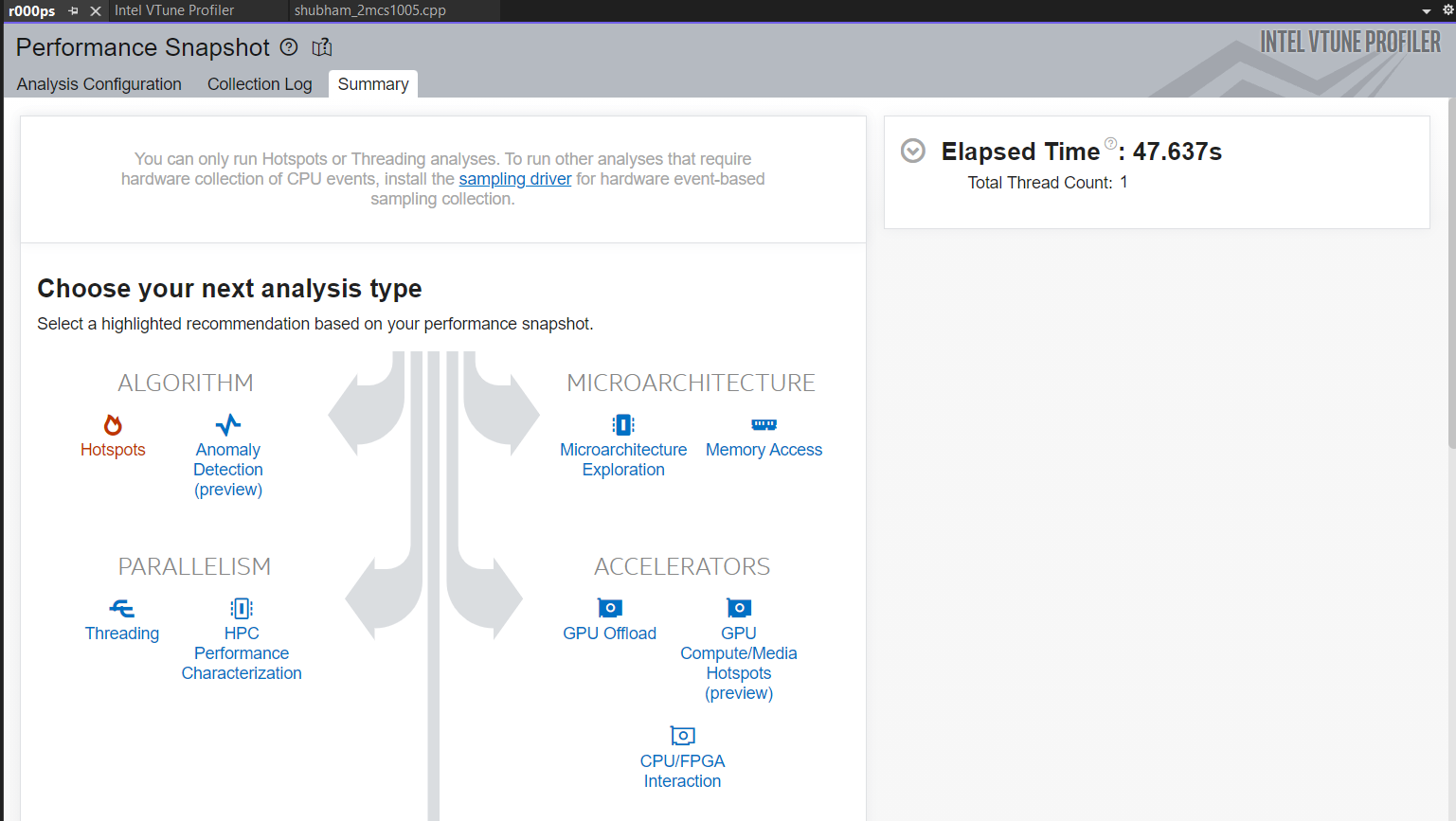
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | A | B | C | D | E |
| A | 0 | 1 | 2 | 3 | 4 |
| B | 5 | 6 | 7 | 8 | 9 |
| C | 5 | 6 | 1 | 2 | 9 |
| D | 9 | 5 | 0 | 1 | 2 |
| E | 0 | 7 | 4 | 2 | 8 |

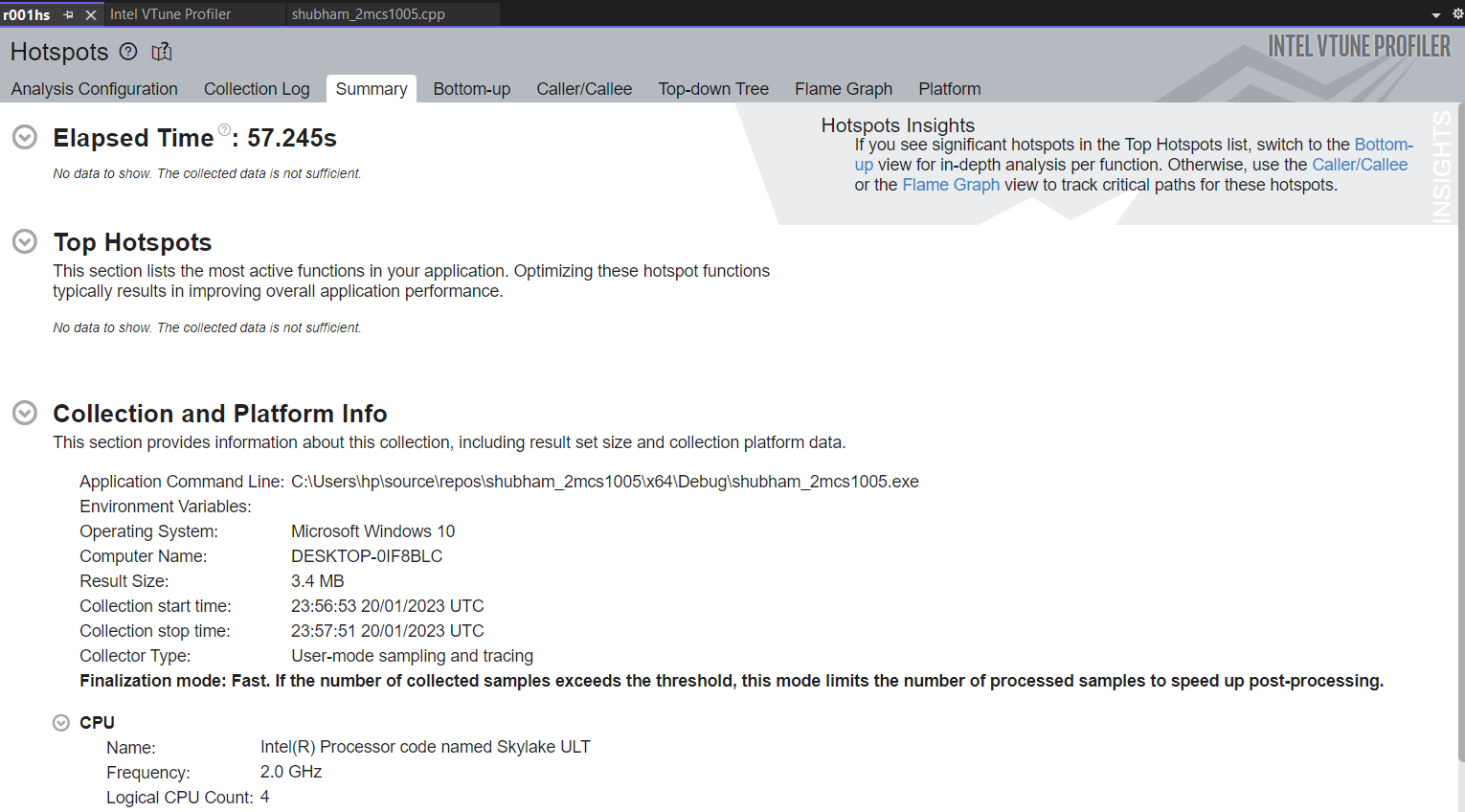
**Minimum cost was 7**

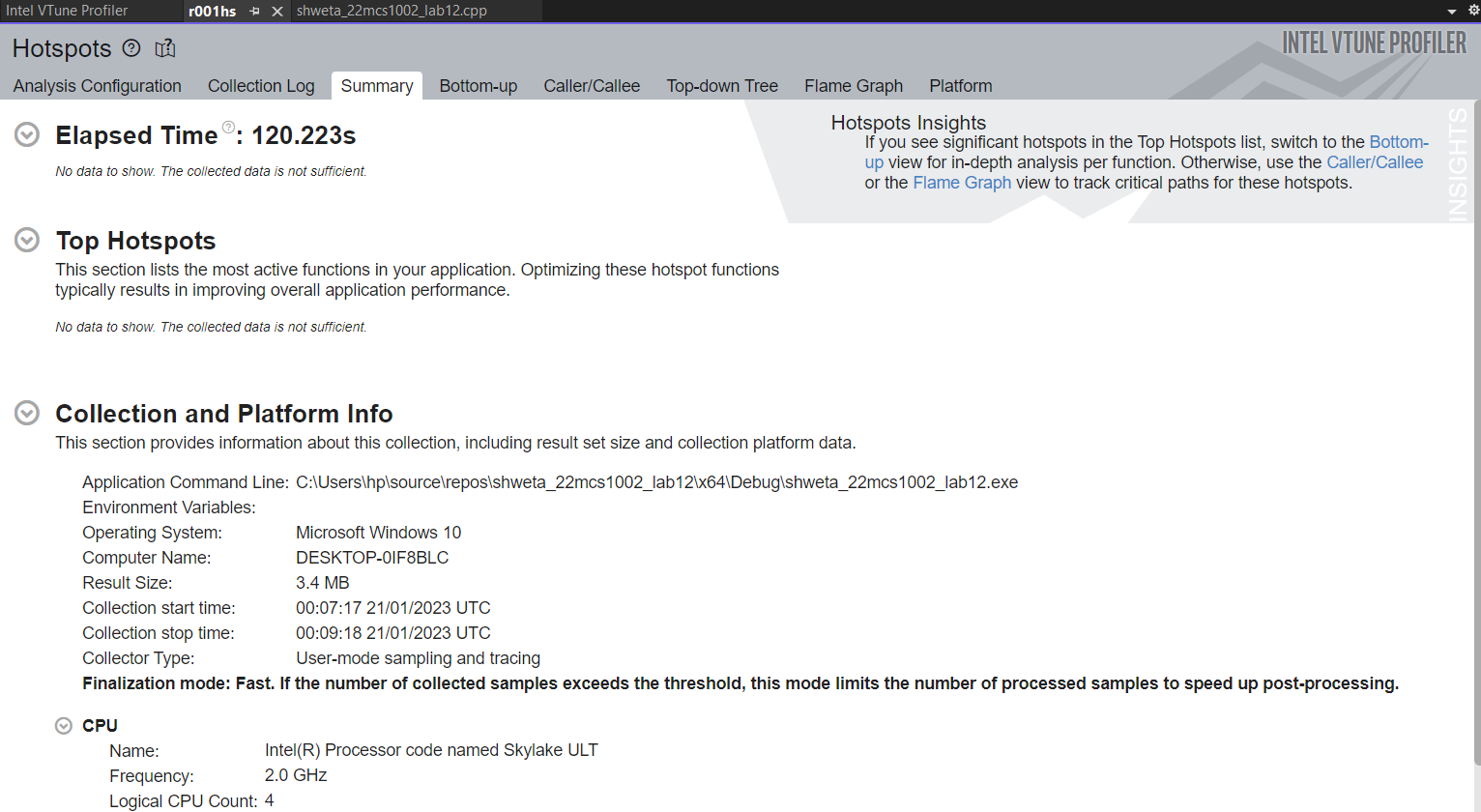
### SCREEN SHOTS OF OUTPUT



Vtune profiler procedure for configuring analysis output:







## EXPERIMENT 13

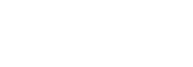
**<<** **CUDA Programming >>**

**AIM:** installing file to run in CUDA and printing a statement

**PROGRAM:**



b



# EXPERIMENT 14

**<< CUDA Programming >>**

**AIM:** performing vector arithmetic operation in CUDA

## PROGRAM:

!wget https://developer.nvidia.com/compute/cuda/9.2/Prod/local\_installers/cu da-repo-ubuntu1604-9-2-local\_9.2.88-1\_amd64 -O cuda-repo-ubuntu1604-9-2- local\_9.2.88-1\_amd64.deb

!dpkg -i cuda-repo-ubuntu1604-9-2-local\_9.2.88-1\_amd64.deb

!apt-key add /var/cuda-repo-9-2-local/7fa2af80.pub

!apt-get update

!apt-get install cuda-9.2

!apt-get update

!nvcc --version

!pip install git+https://github.com/andreinechaev/nvcc4jupyter.git

%load\_ext nvcc\_plugin

%%cu

#include <stdio.h>

global void add(int a, int b, int \*c)

{

\*c = a + b;

}

global void sub(int a, int b, int \*d)

{

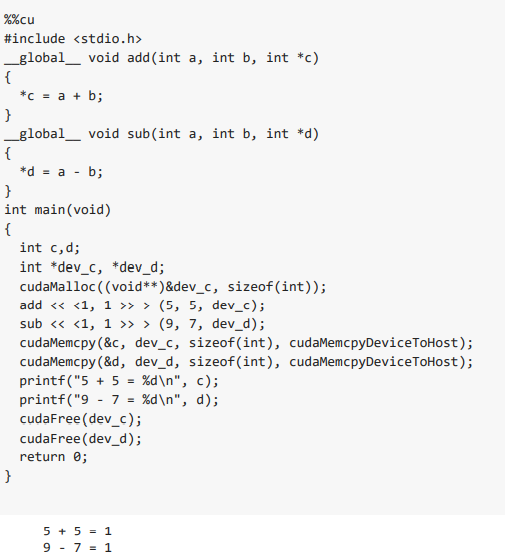
\*d = a - b;

}

int main(void)

{

## SCREEN SHOTS OF OUTPUT



%%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;

}

#a[i]=[0,1,2,3,4,5]

#b[i]=[0,2,4,6,8,10]

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");

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

{

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

}

cudaFree(d\_a); cudaFree(d\_b);

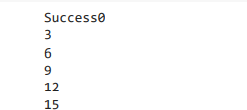
cudaFree(out\_d); free(a);

free(b); free(out);

return 0;

}

**SCREEN SHOTS OF OUTPUT**



%%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; #a[i]=[0,1,2,3,4,5]

b[i]=i\*i; #b[i]=[0,1,4,9,16,25]

}

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");

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;

}

## SCREEN SHOTS OF OUTPUT

