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, 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

TABLE OF CONTENTS

14	12-10-2022	Program to use FirstPrivate	
15	12-10-2022	Program for Math Application	
16	19-10-2022	Perform Static Scheduling	
17	19-10-2022	Perform Dynamic Scheduling	
18	19-10-2022	Perform Guided Scheduling	
19	19-10-2022	Try Modelling Company	
20	09-11-2022	Execute Parallel Region in Ordered Fashion	
21	09-11-2022	Program to use Locks	
22	09-11-2022	Use Locks to Print n even and n odd numbers	
23	16-11-2022	Program for TechnoVIT	
24	16-11-2022	Program using Barrier and File Operations	
25	23-11-2022	Program to find sum series using Barrier	
26	30-11-2022	OpenMP Matrix-Matrix Multiplication	
27	07-12-2022	OpenMP Matrix-Vector Multiplication	
28	14-12-2022	V-tune Profiler for Matrix-Matrix multiplication	
29	21-12-2022	V-tune Profiler for Matrix-Vector multiplication	

30	21-12-2022	V-tune Profiler for Quick Sort	
31	28-12-2022	V-tune Profiler for Minimum Spanning Tree	
32	04-01-2023	Simple CUDA Program	
33	11-01-2023	CUDA Program for Performing Vector Operations	
34	18-01-2023	CUDA Program for Matrix Fill	

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

```
~/GSelvakumar$ gcc main.c -fopenmp
~/GSelvakumar$ ./a.out

Thread Id : 10
Thread Id : 0
Thread Id : 6
Thread Id : 2
Thread Id : 11
Thread Id : 3
Thread Id : 5
Thread Id : 1
Thread Id : 4
Thread Id : 7
Thread Id : 9
~/GSelvakumar$
```

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

```
~/GSelvakumar$ gcc main.c -fopenmp
~/GSelvakumar$ ./a.out

Thread Count : 12
Thread Count : 12
Thread Count : 12
Thread Count : 12
Thread Count : 12
Thread Count : 12
Thread Count : 12
Thread Count : 12
Thread Count : 12
Thread Count : 12
Thread Count : 12
~/GSelvakuma
```

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

```
~/GSelvakumar$ ./a.out

Enter the height : 125

Enter the weight : 87

Thread Id : 7 Your BMI is : 3
Thread Id : 11 Your BMI is : 3
Thread Id : 10 Your BMI is : 3
Thread Id : 4 Your BMI is : 3
Thread Id : 0 Your BMI is : 3
Number of Processors : 12
Thread Id : 6 Your BMI is : 3
Thread Id : 8 Your BMI is : 3
Thread Id : 5 Your BMI is : 3
Thread Id : 2 Your BMI is : 3
Thread Id : 1 Your BMI is : 3
Thread Id : 9 Your BMI is : 3
~/GSelvakumar$
```

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

```
~/GSelvakumar$ gcc lab4.c -fopenmp
~/GSelvakumar$ ./a.out
Thread 0 sum is 0
Thread 0 sum is 2
Thread 0 sum is 4
Thread 0 sum is 6
Thread 0 sum is 8
Thread 0 sum is 10
Thread 0 sum is 12
Thread 0 sum is 14
Thread 0 sum is 16
~/GSelvakumar$
```

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

```
~/GSelvakumar$ ./a.out

Enter Principle Amount : 10000

Enter Interest rate : 2

Enter the time period : 3

Thread Id : 5 Simple Interest : 600
Thread Id : 3 Simple Interest : 600
Thread Id : 10 Simple Interest : 600
Thread Id : 4 Simple Interest : 600
Thread Id : 11 Simple Interest : 600
Thread Id : 2 Simple Interest : 600
Thread Id : 1 Simple Interest : 600
Thread Id : 0 Simple Interest : 600
Thread Id : 7 Simple Interest : 600
Thread Id : 8 Simple Interest : 600
Thread Id : 9 Simple Interest : 600
~/GSelvakumar$
```

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

```
~/GSelvakumar$ gcc lab6.c -fopenmp
~/GSelvakumar$ ./a.out

Thread Id : 7 Sub : -10
Thread Id : 7 Sub : -9
Thread Id : 6 Add : 10
Thread Id : 6 Add : 13
Thread Id : 8 Mul : 0
~/GSelvakumar$
```

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

```
~/GSelvakumar$ ./a.out

Enter the age :12
11
18
16
15

Thread Id : 0 Age : 12 is not eligible
Thread Id : 0 Age : 11 is not eligible
Thread Id : 0 Age : 18 is not eligible
Thread Id : 0 Age : 15 is not eligible
Thread Id : 1 Age : 16 is eligible~/GSelvakumar$
```

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

```
~/GSelvakumar$ gcc lab8.c -fopenmp
~/GSelvakumar$ ./a.out

Thread Id : 6
Thread Id : 5
Thread Id : 11
Thread Id : 1
Thread Id : 0 Time : Sun Nov 25 06:51:44 4447285

Thread Id : 2
Thread Id : 10
Thread Id : 9
Thread Id : 4
Thread Id : 8
Thread Id : 7
Thread Id : 3~/GSelvakumar$
```

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

```
Thread Id : 3~/GSelvakumar$ gcc lab9.c -fopenmp
~/GSelvakumar$ ./a.out

Start time : 7467.838424
Thread Id : 5 Add : 10
Thread Id : 5 Add : 13
Thread Id : 4 Mul : 0
Thread Id : 4 Mul : 22
Thread Id : 1 Sub : -10
Thread Id : 1 Sub : -9
End time : 7467.888808
Execution time : 0.050384~/GSelvakumar$
```

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

```
Enter Registor Number of Student 5900000

Enter Pay of the Student 5900000

Thread Id : 1 Average Pay : 1450000
Total time taken by Thread : 1 is 0.000030
Thread Id : 2 No of students placed in Amazon : 30188192
Thread Id : 2 No of students placed in Google : 1
Thread Id : 2 No of students placed in Shell : 0
Thread Id : 2 No of students placed in Intel : 0
Total time taken by Thread : 2 is 0.000855
Thread Id : 6 Number of Processors : 12
Total time taken by Thread : 6 is 0.000942~/GSelvakumar$
```

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

```
22/01/2023 09:50.59 /home/mobaxterm/Desktop/Selvakumar gcc lab11.c -fopenmp

22/01/2023 09:51.04 /home/mobaxterm/Desktop/Selvakumar ./a.exe

Thread Id : 4 Value of x : 6
Thread Id : 1 Value of x : 2
Thread Id : 3 Value of x : 5
Thread Id : 2 Value of x : 4
Thread Id : 5 Value of x : 7
Thread Id : 0 Value of x : 0
Thread Id : 6 Value of x : 8
Thread Id : 7 Value of x : 9
Thread Id : 1 Value of x : 5
Thread Id : 0 Value of x : 1
Value of x outside paralle region : 9
```

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

```
22/01/2023 09:54.58 /home/mobaxterm/Desktop/Selvakumar gcc lab12.c -fopenmp

22/01/2023 09:55.35 /home/mobaxterm/Desktop/Selvakumar ./a.exe

Thread Id : 0 Value of x : 0
Thread Id : 1 Value of x : 5
Thread Id : 0 Value of x : 1
Thread Id : 1 Value of x : 11
Thread Id : 0 Value of x : 3
Thread Id : 1 Value of x : 18
Thread Id : 0 Value of x : 6
Thread Id : 1 Value of x : 26
Thread Id : 0 Value of x : 10
Thread Id : 1 Value of x : 35
Value of x outside paralle region : 9
```

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

```
22/01/2023  09:57.34  /home/mobaxterm/Desktop/Selvakumar  ./a.exe

Value of x before entering parallel region : 10
Value of x after paralle region : 5
```

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

```
22/01/2023 09:58.50 /home/mobaxterm/Desktop/Selvakumar gcc lab14.c -fopenmp

22/01/2023 09:59.07 /home/mobaxterm/Desktop/Selvakumar ./a.exe

Value of x before entering parallel region : 10
Thread Id : 0 Value of x inside parallel region : 10
Thread Id : 1 Value of x inside parallel region : 10
Thread Id : 4 Value of x inside parallel region : 10
Thread Id : 3 Value of x inside parallel region : 10
Thread Id : 5 Value of x inside parallel region : 10
Thread Id : 2 Value of x inside parallel region : 10
Value of x after paralle region : 10
```

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

```
~/GSelvakumar$ ./a.out

Enter the Number : 6

Thread Id : 0 Num : 6 is Rational
Thread Id : 0 Num : 6 is not Prime
Thread Id : 2 Num : 6 is Perfect~/GSelvakumar$
```

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

```
~/GSelvakumar$ gcc lab16.c -fopenmp
~/GSelvakumar$ ./a.out

Thread Id : 1 Iteration : 2
Thread Id : 1 Iteration : 3
Thread Id : 4 Iteration : 8
Thread Id : 4 Iteration : 9
Thread Id : 3 Iteration : 6
Thread Id : 3 Iteration : 7
Thread Id : 2 Iteration : 4
Thread Id : 2 Iteration : 5
Thread Id : 0 Iteration : 0
Thread Id : 0 Iteration : 1~/GSelvakumar$
```

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

```
~/GSelvakumar$ gcc lab17.c -fopenmp
~/GSelvakumar$ ./a.out

Thread Id : 3 Iteration : 0
Thread Id : 7 Iteration : 2
Thread Id : 7 Iteration : 3
Thread Id : 11 Iteration : 6
Thread Id : 11 Iteration : 7
Thread Id : 5 Iteration : 4
Thread Id : 5 Iteration : 5
Thread Id : 0 Iteration : 8
Thread Id : 0 Iteration : 9
Thread Id : 3 Iteration : 1~/GSelvakumar$
```

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

```
~/GSelvakumar$ gcc lab18.c -fopenmp
~/GSelvakumar$ ./a.out

Thread Id : 8 Iteration : 0
Thread Id : 7 Iteration : 6
Thread Id : 7 Iteration : 7
Thread Id : 8 Iteration : 1
Thread Id : 9 Iteration : 4
Thread Id : 9 Iteration : 5
Thread Id : 0 Iteration : 8
Thread Id : 6 Iteration : 2
Thread Id : 6 Iteration : 3
Thread Id : 0 Iteration : 9~/GSelvakumar$
```

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

```
~/GSelvakumar$ gcc lab19.c -fopenmp
~/GSelvakumar$ ./a.out

Thread : 9 Testing Toy Id : 91
Thread : 9 Testing Toy Id : 92
Thread : 9 Testing Toy Id : 93
Thread : 9 Testing Toy Id : 94
Thread : 9 Testing Toy Id : 95
Thread : 9 Testing Toy Id : 96
Thread : 9 Testing Toy Id : 97
Thread : 9 Testing Toy Id : 98
Thread : 9 Testing Toy Id : 99
Thread : 9 Testing Toy Id : 100
Thread : 9 Testing Toy Id : 211
Thread : 9 Testing Toy Id : 212
Thread : 9 Testing Toy Id : 213
Thread : 9 Testing Toy Id : 214
Thread : 9 Testing Toy Id : 215
Thread : 9 Testing Toy Id : 216
Thread : 9 Testing Toy Id : 217
Thread : 9 Testing Toy Id : 218
Thread : 9 Testing Toy Id : 219
Thread : 9 Testing Toy Id : 220
Thread : 2 Testing Toy Id : 21
Thread : 2 Testing Toy Id : 22
```

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

```
~/GSelvakumar$ gcc lab20.c -fopenmp
~/GSelvakumar$ ./a.out

Thread Id : 4 Un_Ordered Iteration : 4
Thread Id : 2 Un_Ordered Iteration : 2
Thread Id : 1 Un_Ordered Iteration : 1
Thread Id : 3 Un_Ordered Iteration : 3
Thread Id : 0 Un_Ordered Iteration : 0
Thread Id : 0 Ordered Iteration : 0
Thread Id : 1 Ordered Iteration : 1
Thread Id : 2 Ordered Iteration : 2
Thread Id : 3 Ordered Iteration : 3
~/GSelvakumar$
```

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

```
~/GSelvakumar$ gcc lab21.c -fopenmp
~/GSelvakumar$ ./a.out
Thread 0 is executing iteration 0
Thread 0 is executing iteration 1
Thread 0 is executing iteration 2
Thread 1 is executing iteration 0
Thread 1 is executing iteration 1
Thread 1 is executing iteration 2
Thread 5 is executing iteration 0
Thread 5 is executing iteration 1
Thread 5 is executing iteration 2
Thread 7 is executing iteration 0
Thread 7 is executing iteration 1
Thread 7 is executing iteration 2
Thread 8 is executing iteration 0
Thread 8 is executing iteration 1
Thread 8 is executing iteration 2
Thread 4 is executing iteration 0
Thread 4 is executing iteration 1
Thread 4 is executing iteration 2
Thread 6 is executing iteration 0
Thread 6 is executing iteration 1
Thread 6 is executing iteration 2
Thread 10 is executing iteration 0
Thread 10 is executing iteration 1
Thread 10 is executing iteration 2
Thread 3 is executing iteration 0
Thread 3 is executing iteration 1
Thread 3 is executing iteration 2
Thread 2 is executing iteration 0
Thread 2 is executing iteration 1
Thread 2 is executing iteration 2
Thread 9 is executing iteration 0
```

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

```
~/GSelvakumar$ ./a.out

Enter the value of n : 3

Thread Id : 0 Odd number : 1
Thread Id : 1 Odd number : 3
Thread Id : 2 Odd number : 5
Thread Id : 0 Even number : 2
Thread Id : 1 Even number : 4
Thread Id : 2 Even number : 6~/GSelvakumar$
```

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

```
~/GSelvakumar$ gcc lab23.c -fopenmp
~/GSelvakumar$ ./a.out

Enter the Id to register : 2

Registeration completed for : 2
Enter the Id to register : 3

Registeration completed for : 3
Enter the Id to register : 1

Registeration completed for : 1
Enter the Id to register : 6

Registeration completed for : 6
Enter the Id to register : 4

Registeration completed for : 4
Enter the Id to Un-register : 4

Un-Registered : 4
Enter the Id to Un-register : 5

Un-Registered : 5
Enter the Id to Un-register : 6

Un-Registered : 6
Enter the Id to Un-register : 4

Un-Registered : 4
Enter the Id to Un-register : 3

Un-Registered : 3
Thread Id : 0 Registered Student : 1
~/GSelvakumar$
```

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

```
~/GSelvakumar$ gcc lab24.c -fopenmp
~/GSelvakumar$ ./a.out

Thread Id : 11
Thread Id : 2
Thread Id : 4
Thread Id : 10
Thread Id : 7
Thread Id : 0
Thread Id : 1
Thread Id : 6
Thread Id : 5
Thread Id : 3
Thread Id : 8
Thread Id : 9~/GSelvakumar$
```

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

```
~/GSelvakumar$ ./a.out

Enter the value of N : 3
~/GSelvakumar$
```

test.txt

```
Thread Id : 8 Sum is : 0.875000 Last Value : 0.125000
Thread Id : 3 Sum is : 0.875000 Last Value : 0.125000
Thread Id : 11 Sum is : 0.875000 Last Value : 0.125000
Thread Id : 2 Sum is : 0.875000 Last Value : 0.125000
Thread Id : 7 Sum is : 0.875000 Last Value : 0.125000
Thread Id : 1 Sum is : 0.875000 Last Value : 0.125000
Thread Id : 4 Sum is : 0.875000 Last Value : 0.125000
Thread Id : 9 Sum is : 0.875000 Last Value : 0.125000
Thread Id : 0 Sum is : 0.875000 Last Value : 0.125000
Thread Id : 5 Sum is : 0.875000 Last Value : 0.125000
Thread Id : 10 Sum is : 0.875000 Last Value : 0.125000
Thread Id : 6 Sum is : 0.875000 Last Value : 0.125000

```

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

```
~/GSelvakumar$ gcc lab26.c -fopenmp
~/GSelvakumar$ ./a.out

Elapsed time : 0.013252
40  40  40  40  40  40  40  40  40  40
40  40  40  40  40  40  40  40  40  40
40  40  40  40  40  40  40  40  40  40
40  40  40  40  40  40  40  40  40  40
40  40  40  40  40  40  40  40  40  40
40  40  40  40  40  40  40  40  40  40
40  40  40  40  40  40  40  40  40  40
40  40  40  40  40  40  40  40  40  40
40  40  40  40  40  40  40  40  40  40
40  40  40  40  40  40  40  40  40  40
~/GSelvakumar$
```

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

```
~/GSelvakumar$ gcc lab27.c -fopenmp
~/GSelvakumar$ ./a.out

Elapsed time : 0.024862
110 110 110 110 110 110 110 110 110 110 ~/GSelvakumar$
```

EXPERIMENT 28
VTune Profiler for Matrix-Matrix multiplication

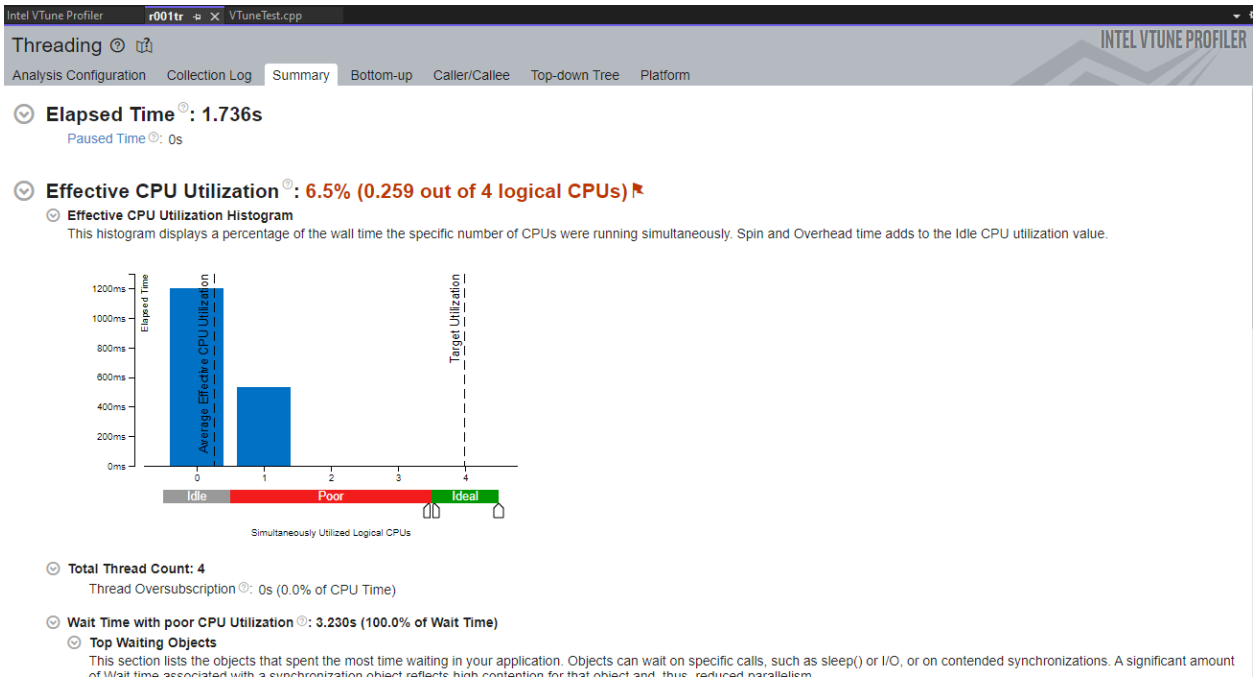
AIM:
To use VTune profiler for analysing Matrix-Matrix multiplication

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


Intel VTune Profiler
r001tr
VTuneTest.cpp
Threading
INTEL VTUNE PROFILER
Analysis Configuration
Collection Log
Summary
Bottom-up
Caller/Callee
Top-down Tree
Platform
Elapsed Time: 1.736s
Paused Time: 0s
Effective CPU Utilization: 6.5% (0.259 out of 4 logical CPUs)
Effective CPU Utilization Histogram
This histogram displays a percentage of the wall time the specific number of CPUs were running simultaneously. Spin and Overhead time adds to the Idle CPU utilization value.
Elapsed Time
1200ms
1000ms
800ms
600ms
400ms
200ms
0ms
Average Effective CPU Utilization
Target Utilization
0
1
2
3
4
Idle
Poor
Ideal
Simultaneously Utilized Logical CPUs
Total Thread Count: 4
Thread Oversubscription: 0s (0.0% of CPU Time)
Wait Time with poor CPU Utilization: 3.230s (100.0% of Wait Time)
Top Waiting Objects
This section lists the objects that spent the most time waiting in your application. Objects can wait on specific calls, such as sleep() or I/O, or on contended synchronizations. A significant amount

EXPERIMENT 29

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

System Overview System Overview

Analysis Configuration | Collection Log | Summary | Platform

Elapsed Time: 0.091s

- CPU Time: 0.307s
- GPU Time: 0.006s
- Context Switch Count: 2,533
- Total Thread Count: 363
- Paused Time: 0s

Top Hotspots

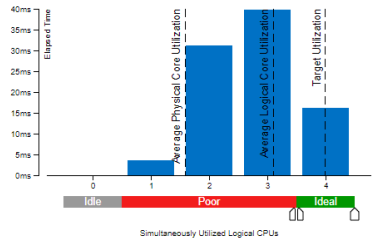
This section lists the most active functions in your application. Optimizing these hotspot functions typically results in improving overall application performance.

Function	Module	CPU Time	% of CPU Time
[Outside any known module]	[Unknown]	0.050s	16.3%
GetQueuedCompletionStatus	kernelbase.dll	0.012s	4.1%
WaitForSingleObjectEx	kernelbase.dll	0.011s	3.4%
func@0x14020de70	ntoskrnl.exe	0.006s	1.9%
func@0x18002bea5	kernelbase.dll	0.006s	1.9%
[Others]	N/A*	0.222s	72.5%

*N/A is applied to non-summable metrics.

Effective CPU Utilization Histogram

This histogram displays a percentage of the wall time the specific number of CPUs were running simultaneously. Spin and Overhead time adds to the Idle CPU utilization value.



EXPERIMENT 30
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
```

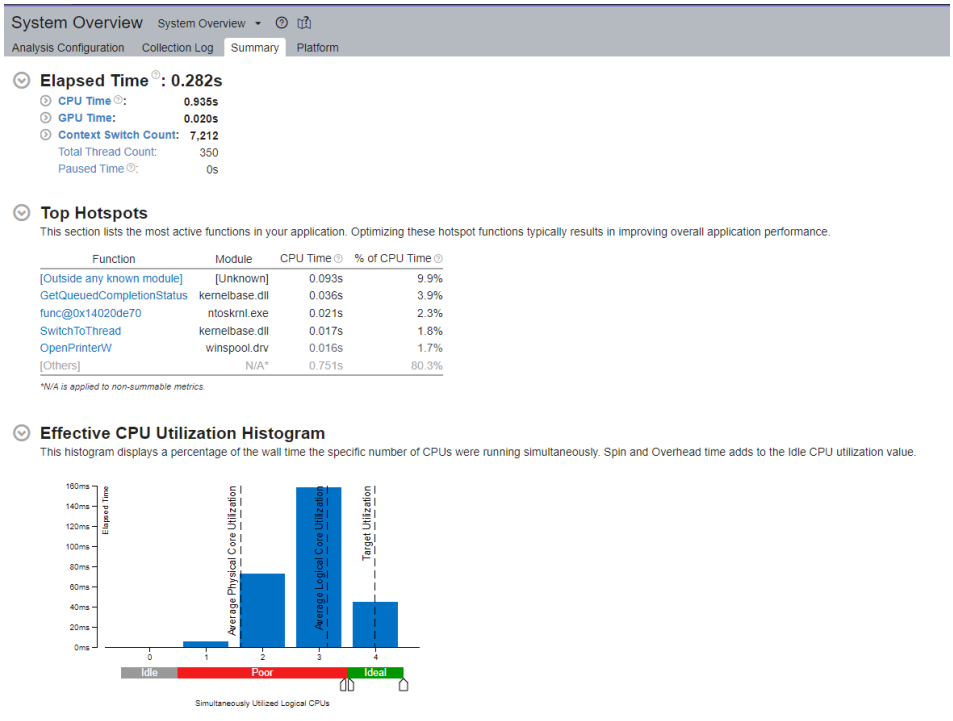
```
                {
                        quicksort(a, low, j - 1);
                }
#pragma omp section
                {
                        quicksort(a, j + 1, high);
                }
        }
    }
}

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



EXPERIMENT 31
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

Threading

INTEL VTUNE PROFILE

Analysis Configuration | Collection Log | Summary | Bottom-up | Caller/Callee | Top-down Tree | Platform

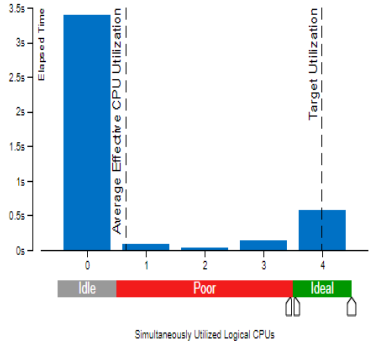
Elapsed Time: 4.206s

Paused Time: 0s

Effective CPU Utilization: 16.6% (0.663 out of 4 logical CPUs)

Effective CPU Utilization Histogram

This histogram displays a percentage of the wall time the specific number of CPUs were running simultaneously. Spin and Overhead time adds to the Idle CPU utilization value.



Total Thread Count: 4

Thread Oversubscription: 0s (0.0% of CPU Time)

Wait Time with poor CPU Utilization: 2.853s (99.8% of Wait Time)

Top Waiting Objects

This section lists the objects that spent the most time waiting in your application. Objects can wait on specific calls, such as sleep() or I/O, or on contended synchronizations. A significant amount of Wait time associated with a synchronization object reflects high contention for that object and, thus, reduced parallelism.

Sync Object	Wait Time with poor CPU Utilization	(% from Object Wait Time)	Wait Count
stdin	2.823s	100.0%	6
Manual Reset Event 0x4c155e90	0.022s	100.0%	1
Manual Reset Event 0xbbba64a8	0.004s	100.0%	1
Stream 0x4b8b3a9e	0.004s	42.1%	9

*N/A is applied to non-summable metrics.

EXPERIMENT 32

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

```
    return 0;
}
```

```
Hello Selvakumar
```

```
return 0;
}
```

```
Selvakumar: 2 + 7 = 9
```

EXPERIMENT 33

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

```
Success Selvakumar!
0
3
6
9
12
15
```

```
Success Selvakumar!
3672
3744
3816
3888
3960
4032
4104
4176
4248
9504
9738
9972
10206
10440
```

EXPERIMENT 34
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

```
Welcome Selvakumar!
allocating array of length 16...
x[0][0][0][0][0][0] = 0
x[0][0][0][0][1][0] = 1
x[0][0][0][1][0][0] = 2
x[0][0][0][1][1][0] = 3
x[0][1][0][0][0][0] = 4
x[0][1][0][0][1][0] = 5
x[0][1][0][1][0][0] = 6
x[0][1][0][1][1][0] = 7
x[1][0][0][0][0][0] = 8
x[1][0][0][0][1][0] = 9
x[1][0][0][1][0][0] = 10
x[1][0][0][1][1][0] = 11
x[1][1][0][0][0][0] = 12
x[1][1][0][0][1][0] = 13
x[1][1][0][1][0][0] = 14
x[1][1][0][1][1][0] = 15
```