



VIT[®]

Vellore Institute of Technology

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

Parallel and Distributed Computing-CSE4001L
S. DHANYA ABHIRAMI
16BCE0965

Lab Slots: L9+L10

Date: 7th August

ASSESSMENT 1

- 1 Hello World Program**
- 2 Program to find number of threads running currently**
- 3 Program to find maximum number of threads**
- 4 Program to find the thread id**
- 5 Program to find the number of processor core in system**
- 6 Program to set number of thread to be executed**
- 7 Program to test is_ parallel function**

CODE

```
#include <omp.h>
#include<stdio.h>
int main()
{
    printf("=====EXERCISES 1 TO 7=====\\n");
    printf("    Basic OpenMP Constructs    ");
    printf("\\n\\t\\t\\t\\tS.DHANYA ABHIRAMI \\n\\t\\t\\t\\t16BCE0965 \\n\\n");
    // Checking if code is executed in parallel
    if(omp_in_parallel()==0)
        printf("In Serial\\n");
    else
        printf("In Parallel\\n");
    // Maximum threads
    int max_threads=omp_get_max_threads();
    printf("Maximum Number of Threads: %d\\n",max_threads);
```

```

// Maximum value for setting number of threads
int thread_limit=omp_get_thread_limit();
printf("Maximum Value for setting: %d\n",thread_limit);
// Initial Number of threads
int init_threads=omp_get_num_threads();
printf("Initial Number of Threads: %d\n\n",init_threads);
// Setting number of threads (default = number of cores)
int num_threads;
printf("Enter number of threads to be set: ");
scanf("%d",&num_threads);
omp_set_num_threads(num_threads);
double time_before,time_after;
#pragma omp parallel
{
    time_before=omp_get_wtime();
    if(omp_in_parallel()==0)
        printf("In Serial\n");
    else
        printf("In Parallel\n");
    // Getting number of processes
    int num_process=omp_get_num_procs();
    printf("Number of Processes: %d\n",num_process);
    int ID = 0;
    printf("hello(%d)", ID);
    printf("world(%d) \n", ID);
    int num1=omp_get_num_threads();
    printf("Number of Threads: %d\n",num1);
    int num=omp_get_thread_num();
    printf("Thread ID: %d\n\n",num);
}
time_after=omp_get_wtime();
printf("Time: %lf\n",time_after-time_before);
return 0;
}

```

OUTPUT

```

16bce0965@sjt516scs051: ~/Desktop
16bce0965@sjt516scs051:~/Desktop$ gcc -fopenmp lab1_1to7.c
16bce0965@sjt516scs051:~/Desktop$ ./a.out lab1_1to7.c
=====EXERCISES 1 TO 7=====
          Basic OpenMP Constructs
                                S.DHANYA ABHIRAMI
                                16BCE0965

In Serial
Maximum Number of Threads: 4
Maximum Value for setting: 2147483647
Initial Number of Threads: 1

Enter number of threads to be set: 3
In Parallel
In Parallel
Number of Processes: 4
hello(0)world(0)
Number of Threads: 3
Thread ID: 2

In Parallel
Number of Processes: 4
hello(0)world(0)
Number of Threads: 3
Number of Processes: 4
hello(0)world(0)
Thread ID: 1

Number of Threads: 3
Thread ID: 0

Time: 0.000039
16bce0965@sjt516scs051:~/Desktop$

```

8 Program to parallelize a simple for loop

CODE

```

#include <omp.h>
#include<stdio.h>
int main()
{
    printf("=====EXERCISES 8=====\\n");
    printf("          Parallelising For Loop          ");
    printf("\\n\\t\\t\\t\\t\\tS.DHANYA ABHIRAMI \\n\\t\\t\\t\\t16BCE0965 \\n\\n");
    int i,thread_num,iter;
    printf("Printing numbers from 1 to n\\n");
    printf("Enter n: ");
    scanf("%d",&iter);
    #pragma omp parallel
    {
        #pragma omp for
        for(i=1;i<=iter;i++){
            printf("\\n%d\\t",i);
            thread_num=omp_get_thread_num();
            printf("Thread ID: %d\\n",thread_num);
        }
    }
    return 0;
}

```

OUTPUT

```
dhanya@dhanya-Lenovo-G50-80: ~/PDC Lab
dhanya@dhanya-Lenovo-G50-80:~/PDC Lab$ gcc -fopenmp lab1_8.c
dhanya@dhanya-Lenovo-G50-80:~/PDC Lab$ ./a.out lab1_8.c
=====EXERCISES 8=====
        Parallelising For Loop
                                S.DHANYA ABHIRAMI
                                16BCE0965

Printing numbers from 1 to n
Enter n: 10

4      Thread ID: 1
5      Thread ID: 1
6      Thread ID: 1
1      Thread ID: 0
2      Thread ID: 0
3      Thread ID: 0
9      Thread ID: 3
10     Thread ID: 3
7      Thread ID: 2
8      Thread ID: 2
dhanya@dhanya-Lenovo-G50-80:~/PDC Lab$
```

9 Write an OpenMP program to find the number of prime numbers in a list of numbers generated randomly. Output the prime number and the thread id that is calculating it. Print the number of prime numbers in the main thread

CODE

```
# include <stdio.h>
# include <omp.h>
# include <stdlib.h>
int isPrime(int n){
    if(n<=1)
        return 0;
    if(n!=2 && n%2==0)
        return 0;
    int i;
    for (i = 3; i * i<= n; i=i+2)
    {
        if(n%i==0)
            return 0;
    }
    return 1;
}
```

```

int main(){
    int i,j,n,upper,lower,count=0;
    printf("=====EXERCISE 9=====\\n");
    printf("    PRIME NUMBER FINDER\\n\\t\\tS. DHANYA
ABHIRAMI\\n\\t\\t16BCE0965\\n");
    printf("Enter size of list: ");
    scanf("%d",&n);
    printf("Enter lower limit: ");
    scanf("%d",&lower);
    printf("Enter upper limit: ");
    scanf("%d",&upper);
    int list[n];
    printf("The generated list: ");
    for (i = 0; i < n; i++) {
        list[i]= (rand() % (upper - lower + 1)) + lower;
        printf("%d ",list[i]);
    }
    printf("\\n");

    // Parallel Execution
    #pragma omp parallel for
    for(i=0;i<n;i++){
        printf("Thread ID: %d\\n",omp_get_thread_num() );
        if(isPrime(list[i])==1)
        {
            printf("%d: Prime\\n",list[i] );
            count=count+1 ;
        }
        //else
        //printf("%d: Not Prime\\n",list[i] );
    }
    printf("\\nNumber of prime numbers: %d\\n",count );
    return(0);
}

```

OUTPUT

```

16bce0965@sjt516scs051: ~/Desktop
16bce0965@sjt516scs051:~/Desktop$ gcc -fopenmp lab1_9.c
16bce0965@sjt516scs051:~/Desktop$ ./a.out lab1_9.c
=====EXERCISE 9=====
    PRIME NUMBER FINDER
        S. DHANYA ABHIRAMI
        16BCE0965
Enter size of list: 5
Enter lower limit: 1
Enter upper limit: 5
The generated list: 4 2 3 1 4
Thread ID: 2
Thread ID: 3
Thread ID: 1
3: Prime
Thread ID: 0
Thread ID: 0
2: Prime

Number of prime numbers: 2
16bce0965@sjt516scs051:~/Desktop$

```

10 Write an OpenMP program to compute the sum of all the elements in a

one dimensional array A using reduction. Create another program that does the same, without using the REDUCE clause. Compare the two versions.[use dynamic memory allocation]

CODE

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

int main()
{
    int n,i,sum;
    double start_time,end_time;
    printf("=====EXERCISE 10=====\\n");
    printf("    SUM OF ARRAY ELEMENTS\\n\\t\\tS. DHANYA
ABHIRAMI\\n\\t\\t16BCE0965\\n");
    printf("Enter size of array: ");
    scanf("%d",&n);
    int *a;
    a=(int *) malloc(sizeof(int)*n);
    printf("Enter elements of the array: ");
    for (i=0;i<n;i++)
    {
        scanf("%d",&a[i]);
    }
    // Parallel Execution without reduce clause
    printf("Parallel Execution without reduce clause\\n");
    sum=0;
    start_time = omp_get_wtime();
        #pragma omp parallel for
        for (i=0;i<n;i++)
        {
            sum+=a[i];
        }
    end_time = omp_get_wtime();
    printf("Execution Time without reduce clause: %lf\\n", end_time-
start_time);
    printf("Sum = %d\\n",sum);
    // Parallel Execution with reduce clause
    printf("Parallel Execution\\n");
    sum=0;
    start_time = omp_get_wtime();
    #pragma omp parallel for reduction (+:sum)
        for (i=0;i<n;i++)
        {
            sum+=a[i];
        }
    end_time = omp_get_wtime();
    printf("Execution Time with reduce clause: %lf\\n", end_time-
start_time);
    printf("Sum = %d\\n",sum);
    return 0;
}
```

OUTPUT

```

16bce0965@sjt516scs051: ~/Desktop
16bce0965@sjt516scs051:~/Desktop$ gcc -fopenmp lab1_10.c
16bce0965@sjt516scs051:~/Desktop$ ./a.out lab1_10.c
=====EXERCISE 10=====
      SUM OF ARRAY ELEMENTS
              S. DHANYA ABHIRAMI
              16BCE0965
Enter size of array: 5
Enter elements of the array: 1 2 3 4 5
Parallel Execution without reduce clause
Execution Time without reduce clause: 0.001613
Sum = 15
Parallel Execution
Execution Time with reduce clause: 0.000006
Sum = 15
16bce0965@sjt516scs051:~/Desktop$

```

RESULTS

The code with reduction clause executes much faster compared to the program without reduction clause.

11 Write a program to find sum of squares of first hundred natural numbers see that half computation is done by one core and another half is computed by another core. Finally results of computations are added and the final result is to be printed in master thread.

CODE

```

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

int main(){
    int i,first_fifty_sum = 0,next_fifty_sum = 0;
    printf("=====SUM OF SQUARES FIRST 100 NATURAL
NUMBERS=====\\n\\t\\tS. DHANYA ABHIRAMI\\n\\t\\t16BCE0965\\n");
    double start_time=omp_get_wtime();
    #pragma omp parallel sections
    {
        #pragma omp section
        {
            int thread_id = omp_get_thread_num();
            printf("Thread ID: %d\\n",thread_id );
            for(i=1;i<=50;i++)
            {
                first_fifty_sum+=i*i;
            }
            printf("Partial sum = %d\\n", first_fifty_sum);
        }
        #pragma omp section
        {
            int thread_id = omp_get_thread_num();
            printf("Thread ID: %d\\n",thread_id );
            for(i=51;i<=100;i++)
            {
                next_fifty_sum+=i*i;
            }
        }
    }
    printf("Final sum = %d\\n", first_fifty_sum + next_fifty_sum);
    double end_time=omp_get_wtime();
    printf("Execution Time: %f\\n", end_time - start_time);
}

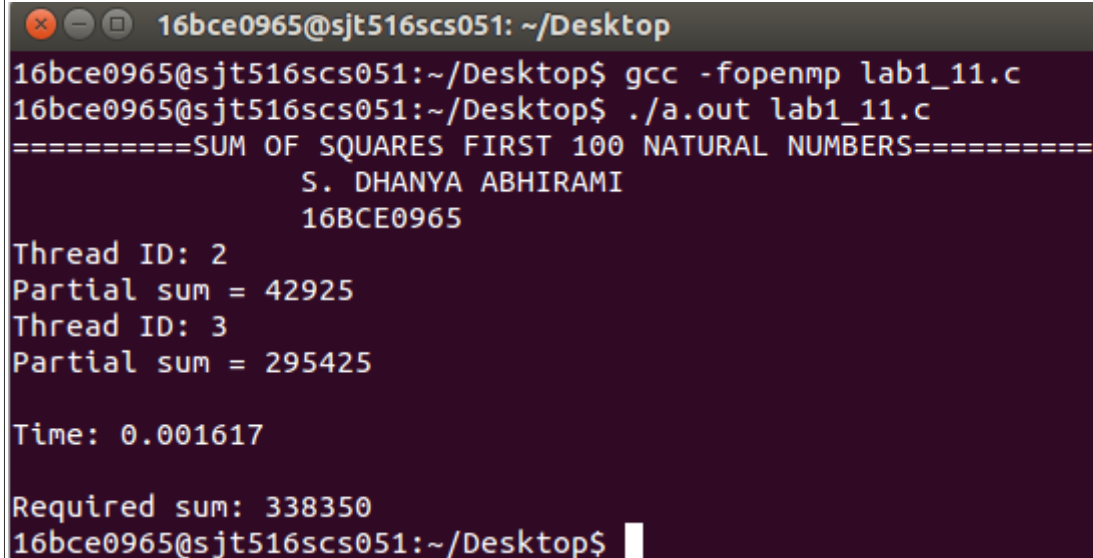
```

```

    }
    printf("Partial sum = %d\n", next_fifty_sum);
}
}
printf("\nTime: %lf\n", omp_get_wtime()-start_time);
printf("\nRequired sum: %d\n", first_fifty_sum+next_fifty_sum);
return(0);
}

```

OUTPUT



```

16bce0965@sjt516scs051: ~/Desktop
16bce0965@sjt516scs051:~/Desktop$ gcc -fopenmp lab1_11.c
16bce0965@sjt516scs051:~/Desktop$ ./a.out lab1_11.c
=====SUM OF SQUARES FIRST 100 NATURAL NUMBERS=====
                S. DHANYA ABHIRAMI
                16BCE0965

Thread ID: 2
Partial sum = 42925
Thread ID: 3
Partial sum = 295425

Time: 0.001617

Required sum: 338350
16bce0965@sjt516scs051:~/Desktop$

```

12 Write a C program and parallelize it using OpenMP Sections construct for the following scenario

- a. Biggest of n numbers
- b. Smallest of n numbers
- c. Factorial of n
- d. Fibonacci sequence.

and compute its execution time. Compare the execution time for sequential and parallel for different number of elements and tabulate the results for five entries.

CODE

```

#include<stdio.h>
#include<stdlib.h>
#include<omp.h>
int main(){
    int i,n,min,max, fact;
    double seq_start_time, seq_end_time, start_time,
end_time, parallel_time, sequential_time;
    printf("=====EXERCISE 12=====\\n\\t\\t\\tS. DHANYA
ABHIRAMI\\n\\t\\t\\t16BCE0965\\n");
    printf("Enter n: ");
    scanf("%d",&n);
    int *a;
    a=(int*)malloc(sizeof(int)*n);
    printf("Enter array elements: ");
    for(i=0;i<n;i++)
    {scanf("%d",&a[i]);}
    int *fib_array_seq = (int*) malloc(sizeof(int)*n);
    int *fib_array = (int*) malloc(sizeof(int)*n);
    //Executing sequentially
    printf("Sequential Computation\\n");
    seq_start_time = omp_get_wtime();

```



```

max = -1;
for(i=0;i<n;i++){
if(a[i]>max)
max = a[i];}

min = 10000;
for(i=0;i<n;i++){
if(a[i]<min)
min = a[i];}

fact = 1;
for(i=1;i<=n;i++){
fact = fact*i;
}

int first,second;
fib_array_seq[0]=0; first=fib_array_seq[0];
fib_array_seq[1]=1; second = fib_array_seq[1];
for(i=2;i<n;i++){
fib_array_seq[i]=first+second;
first=second;
second=fib_array_seq[i];
}

seq_end_time = omp_get_wtime();
sequential_time = seq_end_time - seq_start_time;
printf("Max = %d\n",max);
printf("Min = %d\n",min);
printf("Factorial = %d\n",fact);
printf("Fibonacci Sequence: ");
for(i=0;i<n;i++)
{printf("%d ",fib_array_seq[i]);}
printf("\nExecution Time: %g\n\n",sequential_time);

//Executing in parallel
printf("Parallel Computation\n");
start_time = omp_get_wtime();
#pragma omp parallel
{
    #pragma omp sections
    {
        #pragma omp section
        {
            int thread_id = omp_get_thread_num();
            //printf("Thread: %d\n",thread_id);
            max = -1;
            for(i=0;i<n;i++){
            if(a[i]>max)
            max = a[i];
            }
        }
        #pragma omp section
        {
            int thread_id = omp_get_thread_num();
            printf("Thread: %d\n",thread_id);
            min = 10000;
            for(i=0;i<n;i++){
            if(a[i]<min)
            min = a[i];
            }
        }
    }
}

```

```

    }
    #pragma omp section
    {
        int thread_id = omp_get_thread_num();
        printf("Thread: %d\n", thread_id);
        fact = 1;
        for(i=1; i<=n; i++){
            fact = fact*i;
        }
    }
    #pragma omp section
    {
        int thread_id = omp_get_thread_num();
        printf("Thread: %d\n", thread_id);
        int first, second;
        fib_array[0]=0; first=fib_array[0];
        fib_array[1]=1; second = fib_array[1];
        for(i=2; i<n; i++){
            fib_array[i]=first+second;
            first=second;
            second=fib_array[i];
        }
    }

}

end_time = omp_get_wtime();
parallel_time = end_time - start_time;
printf("Max = %d\n", max);
printf("Min = %d\n", min);
printf("Factorial = %d\n", fact);
printf("Fibonacci Sequence: ");
for(i=0; i<n; i++)
{printf("%d ", fib_array[i]);}
printf("\nExecution Time: %g\n", parallel_time);
return(0);
}

```

OUTPUT

```
16bce0965@sjt516scs051: ~/Desktop
16bce0965@sjt516scs051:~/Desktop$ gcc -fopenmp lab1_12.c
16bce0965@sjt516scs051:~/Desktop$ ./a.out lab1_12.c
=====EXERCISE 12=====
                                S. DHANYA ABHIRAMI
                                16BCE0965

Enter n: 5
Enter array elements: 1 2 3 4 5
Sequential Computation
Max = 5
Min = 1
Factorial = 120
Fibonacci Sequence: 0 1 1 2 3
Execution Time: 3.47e-07

Parallel Computation
Thread: 2
Thread: 2
Thread: 3
Max = 5
Min = 1
Factorial = 120
Fibonacci Sequence: 0 1 1 2 3
Execution Time: 0.0016143
16bce0965@sjt516scs051:~/Desktop$
```

```
16bce0965@sjt516scs051: ~/Desktop
16bce0965@sjt516scs051:~/Desktop$ ./a.out lab1_12.c
=====EXERCISE 12=====
                                S. DHANYA ABHIRAMI
                                16BCE0965

Enter n: 10
Enter array elements: 1 2 3 4 5 6 7 8 9 10
Sequential Computation
Max = 10
Min = 1
Factorial = 3628800
Fibonacci Sequence: 0 1 1 2 3 5 8 13 21 34
Execution Time: 3.78001e-07

Parallel Computation
Thread: 2
Thread: 3
Thread: 0
Max = 10
Min = 1
Factorial = 3628800
Fibonacci Sequence: 0 1 1 2 3 5 8 13 21 34
Execution Time: 0.00312063
16bce0965@sjt516scs051:~/Desktop$
```

```

16bce0965@sjt516scs051: ~/Desktop
16bce0965@sjt516scs051:~/Desktop$ ./a.out lab1_12.c
=====EXERCISE 12=====
                        S. DHANYA ABHIRAMI
                        16BCE0965

Enter n: 15
Enter array elements: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
Sequential Computation
Max = 15
Min = 1
Factorial = 2004310016
Fibonacci Sequence: 0 1 1 2 3 5 8 13 21 34 55 89 144 233 377
Execution Time: 5.11e-07

Parallel Computation
Thread: 2
Thread: 2
Thread: 3
Max = 15
Min = 1
Factorial = 2004310016
Fibonacci Sequence: 0 1 1 2 3 5 8 13 21 34 55 89 144 233 377
Execution Time: 0.00161483
16bce0965@sjt516scs051:~/Desktop$

```

```

dhanya@dhanya-Lenovo-G50-80: ~/PDC Lab
dhanya@dhanya-Lenovo-G50-80:~/PDC Lab$ gcc -fopenmp lab1_12.c
dhanya@dhanya-Lenovo-G50-80:~/PDC Lab$ ./a.out lab1_12.c
=====EXERCISE 12=====
                        S. DHANYA ABHIRAMI
                        16BCE0965

Enter n: 6
Enter array elements: 198 124 35 56 9879 467
Sequential Computation
Max = 9879
Min = 35
Factorial = 720
Fibonacci Sequence: 0 1 1 2 3 5
Execution Time: 1.894e-06

Parallel Computation
Thread: 0
Thread: 1
Thread: 3
Max = 9879
Min = 35
Factorial = 720
Fibonacci Sequence: 0 1 1 2 3 5
Execution Time: 0.00106289
dhanya@dhanya-Lenovo-G50-80:~/PDC Lab$

```

```
dhanya@dhanya-Lenovo-G50-80: ~/PDC Lab
dhanya@dhanya-Lenovo-G50-80:~/PDC Lab$ ./a.out lab1_12.c
=====EXERCISE 12=====
                        S. DHANYA ABHIRAMI
                        16BCE0965

Enter n: 7
Enter array elements: 5676 345 235 45 789 123 56
Sequential Computation
Max = 5676
Min = 45
Factorial = 5040
Fibonacci Sequence: 0 1 1 2 3 5 8
Execution Time: 1.669e-06

Parallel Computation
Thread: 3
Thread: 0
Thread: 1
Max = 5676
Min = 45
Factorial = 5040
Fibonacci Sequence: 0 1 1 2 3 5 8
Execution Time: 0.00162393
dhanya@dhanya-Lenovo-G50-80:~/PDC Lab$
```

RESULTS

n	Sequential Time	Parallel Time(With sections)
5	3.47e-07	0.0016143
10	3.78001e-07	0.00312063
15	5.11e-07	0.00161483
6	1.894e-06	0.00106289
7	1.669e-06	0.00162393

The execution time of parallel code is much higher than the execution times of sequential program code. This is due to overheads in forking and joining threads. The parallelisation provides better performance for very large input sizes.