Abinash Satapathy

Reg. No.: 16BCE0081

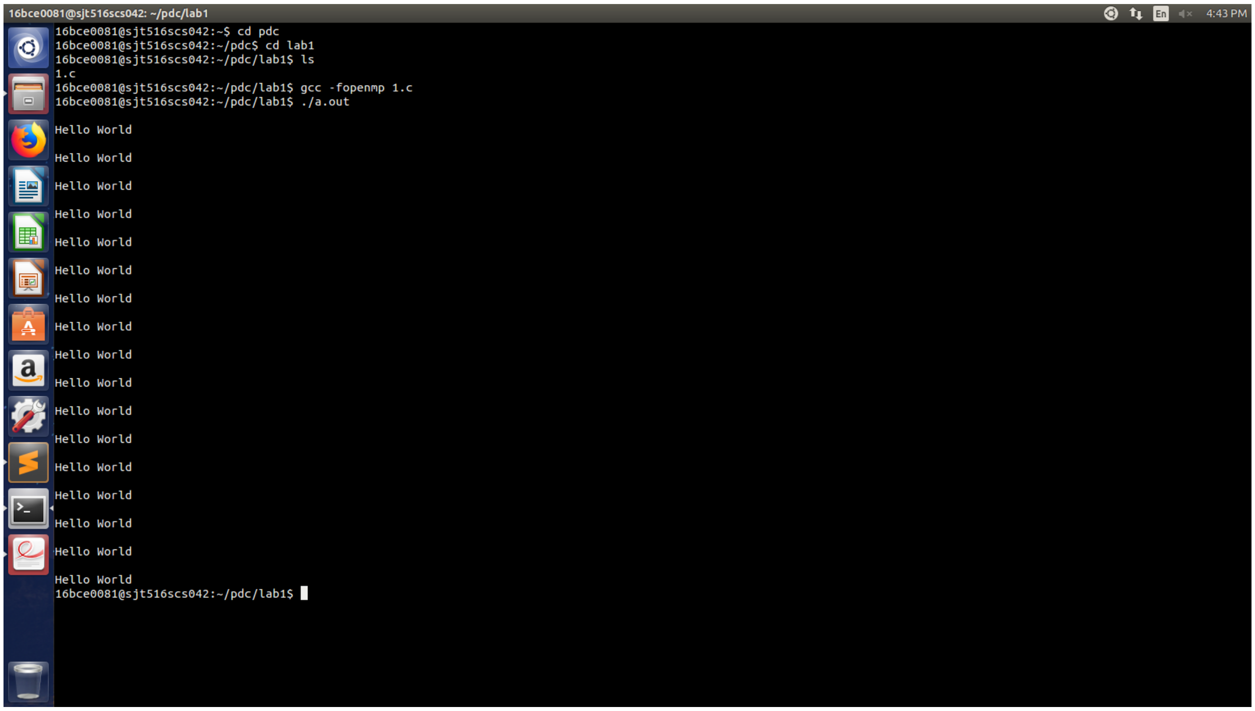
Slot: L45 + L46

Subject: Parallel and Distributed Computing (CSE4001)

Experiment: 1

1. Hello World Program

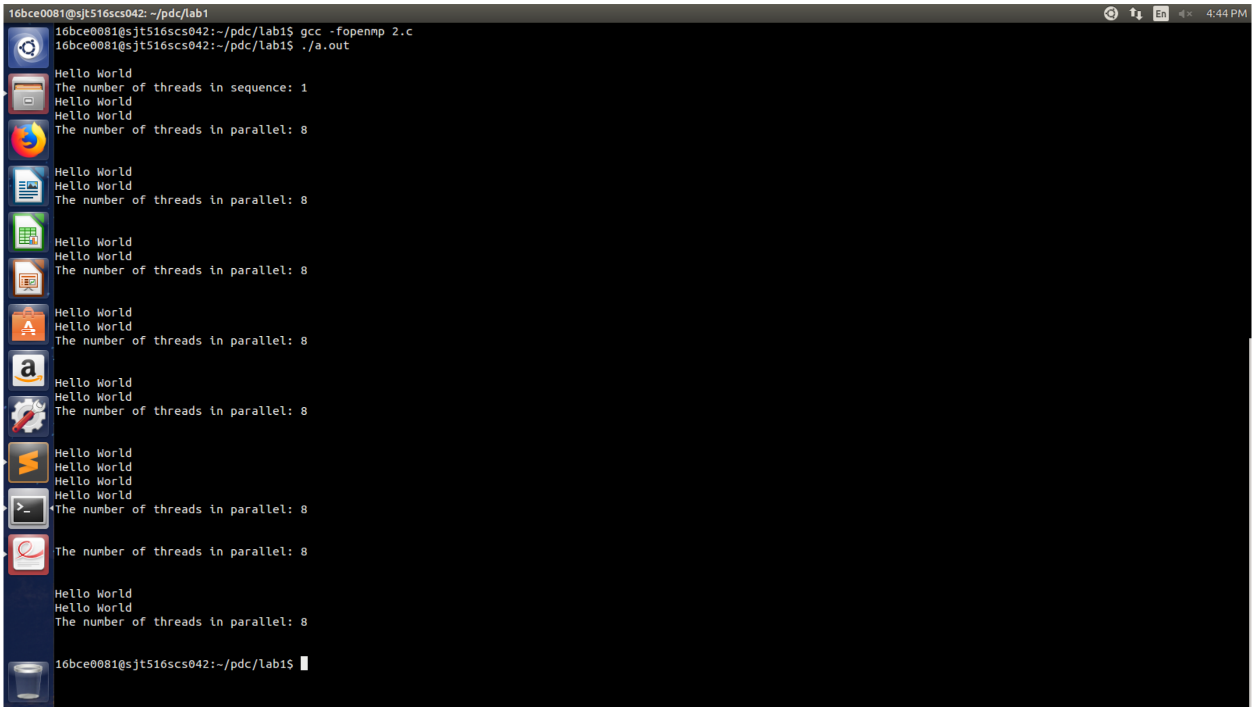
```
#include <stdio.h>
#include <omp.h>
int main(){
        printf("\nHello World\n");

        #pragma omp parallel
        {
        printf("\nHello World\n");
        printf("\nHello World\n");
}

return 0;
}
```



```
16bce0081@sjt516scs042:~$ cd pdc
16bce0081@sjt516scs042:~/pdc$ cd lab1
16bce0081@sjt516scs042:~/pdc/lab1$ ls
1.c
16bce0081@sjt516scs042:~/pdc/lab1$ gcc -fopenmp 1.c
16bce0081@sjt516scs042:~/pdc/lab1$ ./a.out

Hello World

Hello World

Hello World

Hello World

Hello World

Hello World

Hello World

Hello World

Hello World

Hello World

Hello World

Hello World

Hello World

Hello World

Hello World

Hello World

Hello World
16bce0081@sjt516scs042:~/pdc/lab1$ 
```

2. No. of threads running

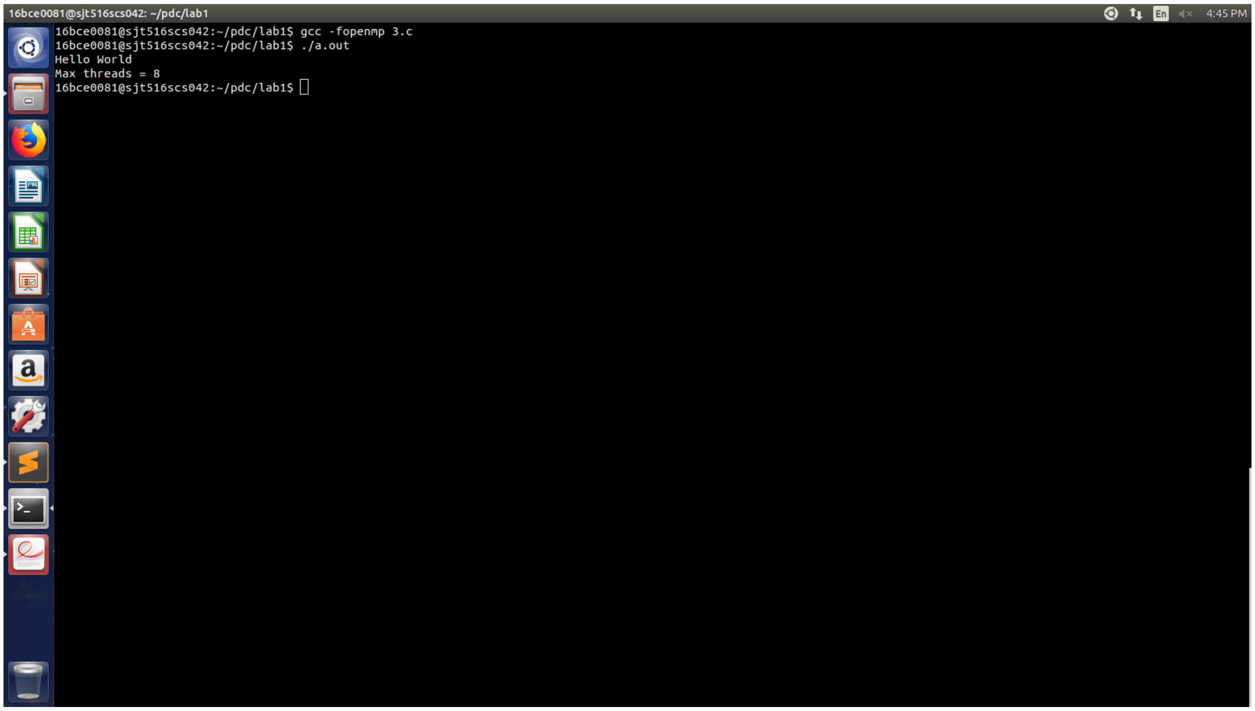
```
#include <stdio.h>
#include <omp.h>

int main(){
        printf("\nHello World\n");
        int a = omp_get_num_threads();
        printf("The number of threads in sequence: %d\n", a);
        #pragma omp parallel
        {
        printf("Hello World\n");
        printf("Hello World\n");

        int n = omp_get_num_threads();
        printf("The number of threads in parallel: %d\n", n);

        printf("\n\n");
}
return 0;
}
```



```
16bce0081@sjt516scs042:~/pdc/lab1$ gcc -fopenmp 2.c
16bce0081@sjt516scs042:~/pdc/lab1$ ./a.out

Hello World
The number of threads in sequence: 1
Hello World
Hello World
The number of threads in parallel: 8


Hello World
Hello World
The number of threads in parallel: 8


Hello World
Hello World
The number of threads in parallel: 8


Hello World
Hello World
The number of threads in parallel: 8


Hello World
Hello World
The number of threads in parallel: 8


Hello World
Hello World
Hello World
Hello World
The number of threads in parallel: 8


The number of threads in parallel: 8


Hello World
Hello World
The number of threads in parallel: 8


16bce0081@sjt516scs042:~/pdc/lab1$ 
```

3. Maximum no. of threads

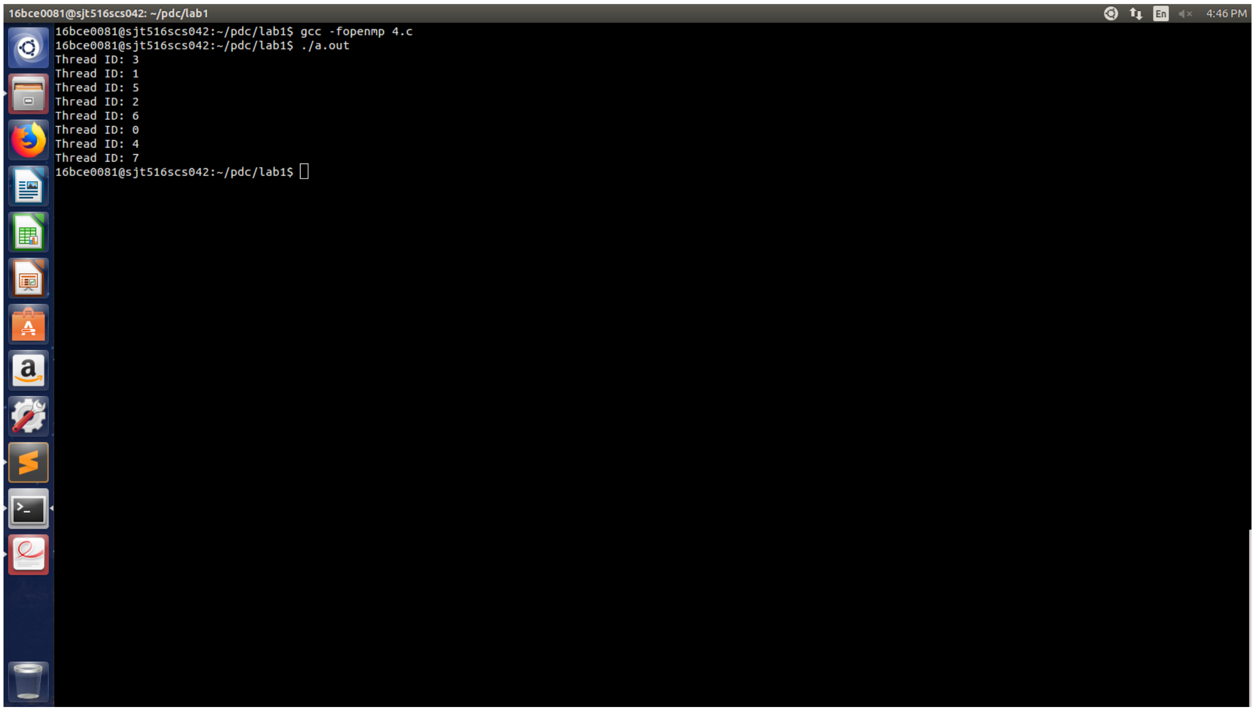
```
#include <stdio.h>
#include <omp.h>

int main(){
        printf("Hello World\n");
        int n = omp_get_max_threads();
        printf("Max threads = %d\n", n);

return 0;
}
```



```
16bce0081@sjt516scs042:~/pdc/lab1$ gcc -fopenmp 3.c
16bce0081@sjt516scs042:~/pdc/lab1$ ./a.out
Hello World
Max threads = 8
16bce0081@sjt516scs042:~/pdc/lab1$ 
```

4. Find thread ID

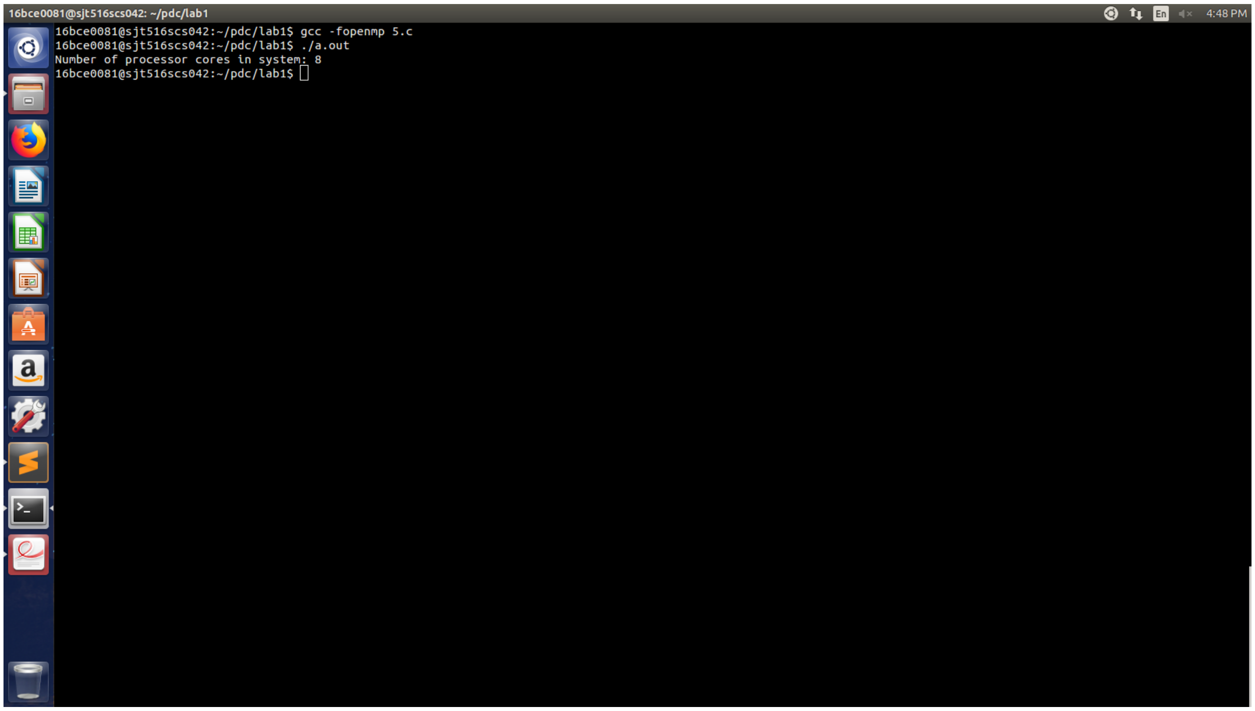
```
#include <stdio.h>
#include <omp.h>

int main(){
        #pragma omp parallel
        {
                printf("Thread ID: %d\n", omp_get_thread_num());
        }

        return 0;
}
```



```
16bce0081@sjt516scs042:~/pdc/lab1$ gcc -fopenmp 4.c
16bce0081@sjt516scs042:~/pdc/lab1$ ./a.out
Thread ID: 3
Thread ID: 1
Thread ID: 5
Thread ID: 2
Thread ID: 6
Thread ID: 0
Thread ID: 4
Thread ID: 7
16bce0081@sjt516scs042:~/pdc/lab1$ 
```

5. Find no. of processor cores in system

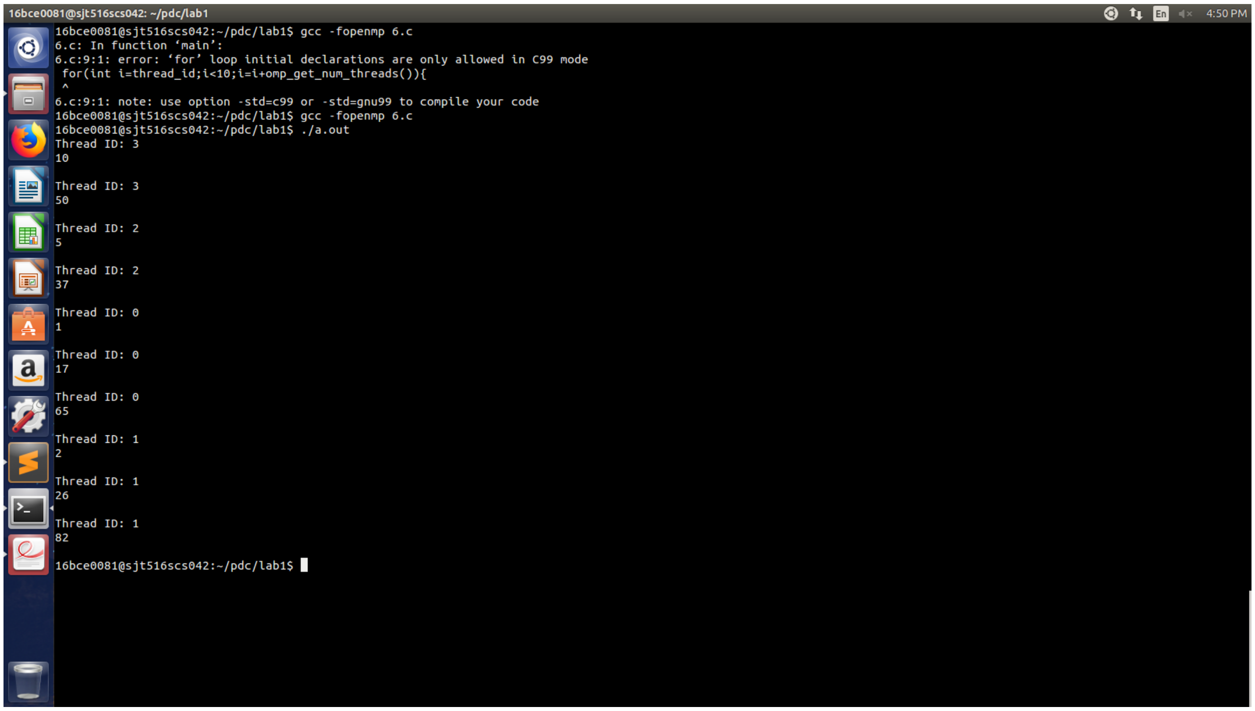
```
#include <stdio.h>
#include <omp.h>

int main(){
        printf("Number of processor cores in system: %d\n", omp_get_num_procs());

        return 0;
}
```



```
16bce0081@sjt516scs042:~/pdc/lab1$ gcc -fopenmp 5.c
16bce0081@sjt516scs042:~/pdc/lab1$ ./a.out
Number of processor cores in system: 8
16bce0081@sjt516scs042:~/pdc/lab1$ 
```

6. Set number of threads to be executed

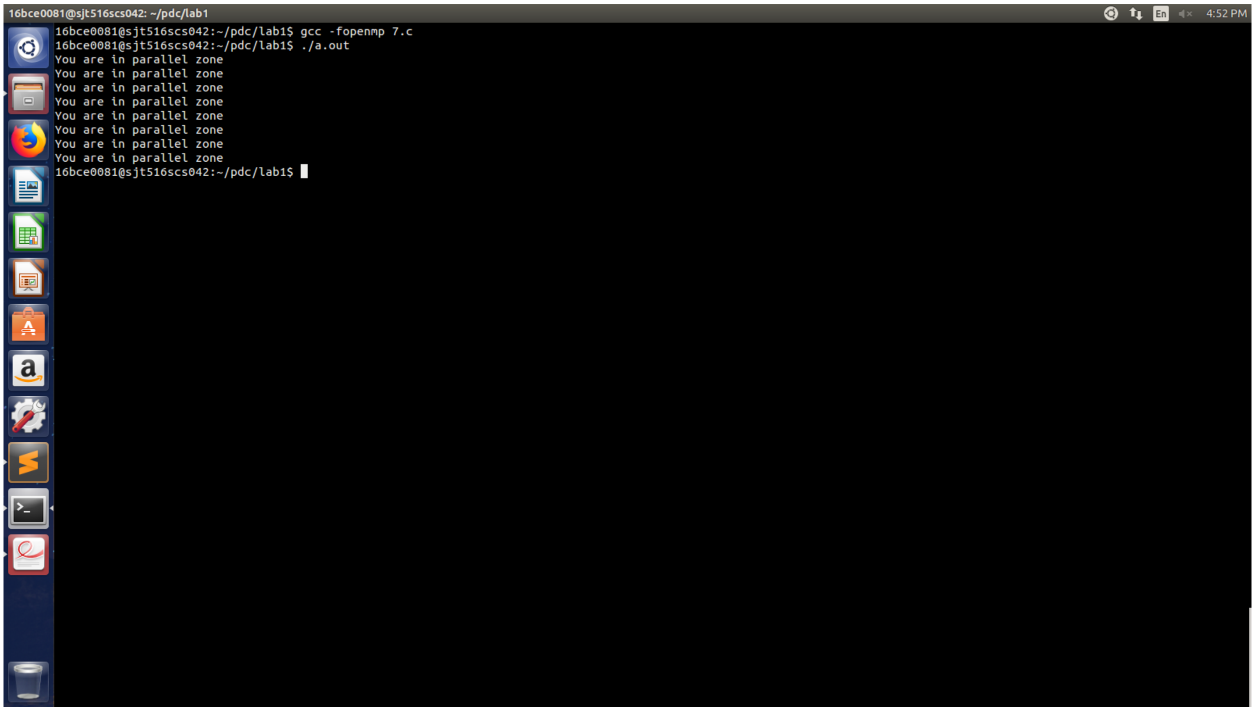
```
#include <stdio.h>
#include <omp.h>

int main(){
        int a[10];

        omp_set_num_threads(4);
        #pragma omp parallel
        {
                int thread_id = omp_get_thread_num();
                for(int i=thread_id;i<10;i=i+omp_get_num_threads()){
                        a[i] = i*i + 1;
                        printf("Thread ID: %d\n", omp_get_thread_num());
                        printf("%d\n\n", a[i]);
                }
        }

        return 0;
}
```



```
16bce0081@sjt516scs042:~/pdc/lab1$ gcc -fopenmp 6.c
6.c: In function ‘main’:
6.c:9:1: error: ‘for’ loop initial declarations are only allowed in C99 mode
 for(int i=thread_id;i<10;i=i+omp_get_num_threads()){
 ^
6.c:9:1: note: use option -std=c99 or -std=gnu99 to compile your code
16bce0081@sjt516scs042:~/pdc/lab1$ gcc -fopenmp 6.c
16bce0081@sjt516scs042:~/pdc/lab1$ ./a.out
Thread ID: 3
10

Thread ID: 3
50

Thread ID: 2
5

Thread ID: 2
37

Thread ID: 0
1

Thread ID: 0
17

Thread ID: 0
65

Thread ID: 1
2

Thread ID: 1
26

Thread ID: 1
82

16bce0081@sjt516scs042:~/pdc/lab1$ 
```

7. Test in_parallel() function

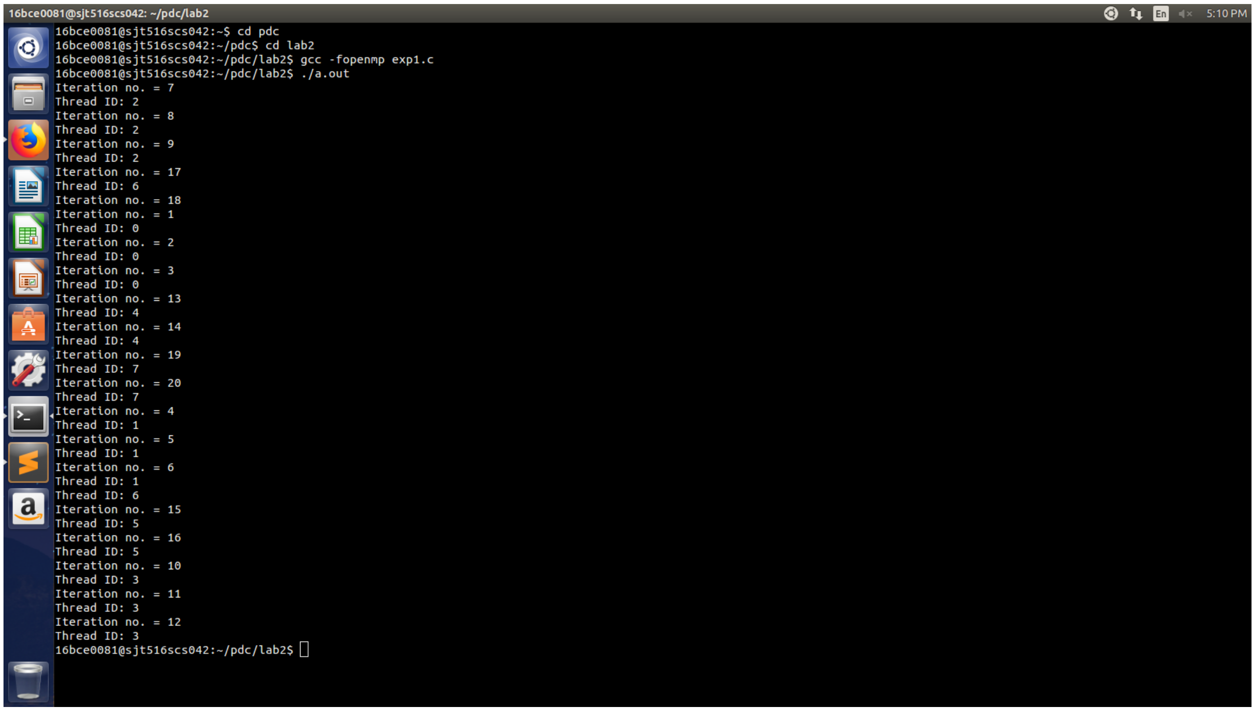
```
#include <stdio.h>
#include <omp.h>

int main(){
        #pragma omp parallel
        {
                int n = omp_in_parallel();
                if(n==1)
                        printf("You are in parallel zone\n");
                else
                        printf("You are in sequence zone\n");
        }

return 0;

}
```



```
16bce0081@sjt516scs042:~/pdc/lab1$ gcc -fopenmp 7.c
16bce0081@sjt516scs042:~/pdc/lab1$ ./a.out
You are in parallel zone
You are in parallel zone
You are in parallel zone
You are in parallel zone
You are in parallel zone
You are in parallel zone
You are in parallel zone
You are in parallel zone
16bce0081@sjt516scs042:~/pdc/lab1$ 
```

8. Program to parallelize a simple **for** loop

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

int main(){
        int i = 0;
        // clock_t start_clock = clock();
        #pragma omp parallel
        {
                #pragma omp for
                for(i=0;i<20;i++){
                        printf("Iteration no. = %d\n", i+1);
                        printf("Thread ID: %d\n", omp_get_thread_num());
                }
        }

        return 0;
}
```

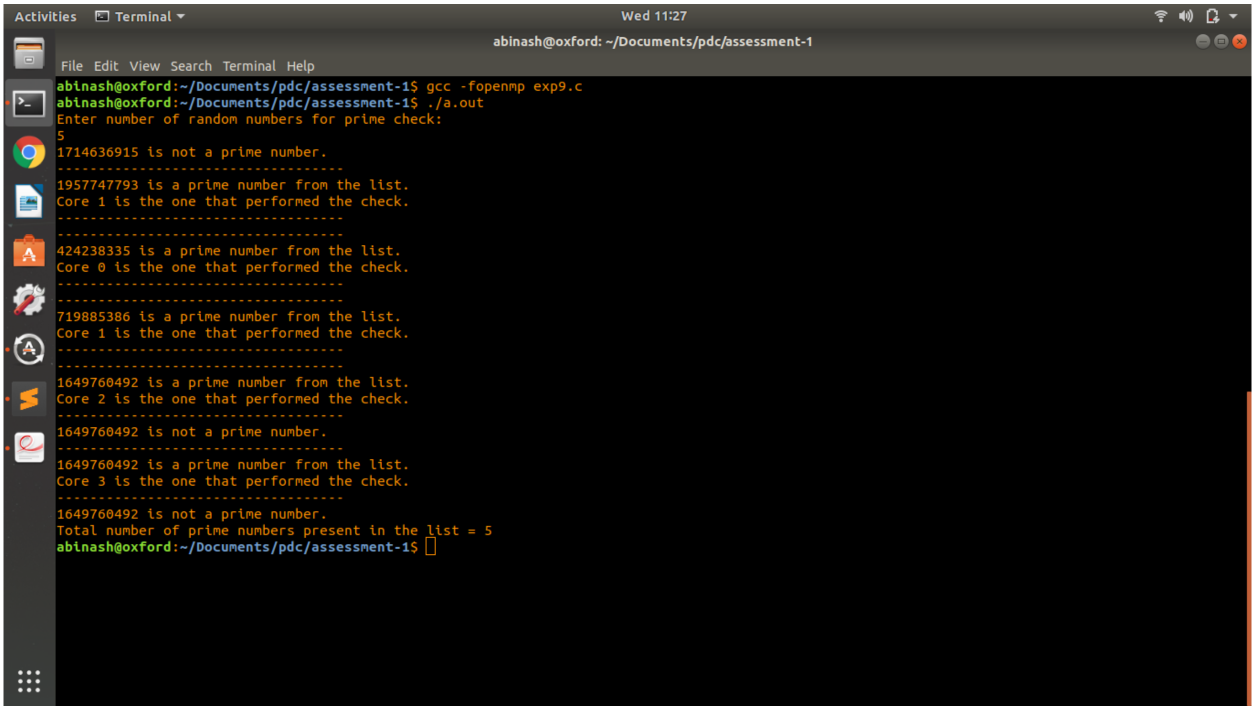


```
16bce0081@sjt516scs042:~$ cd pdc
16bce0081@sjt516scs042:~/pdc$ cd lab2
16bce0081@sjt516scs042:~/pdc/lab2$ gcc -fopenmp exp1.c
16bce0081@sjt516scs042:~/pdc/lab2$ ./a.out
Iteration no. = 7
Thread ID: 2
Iteration no. = 8
Thread ID: 2
Iteration no. = 9
Thread ID: 2
Iteration no. = 17
Thread ID: 6
Iteration no. = 18
Iteration no. = 1
Thread ID: 0
Iteration no. = 2
Thread ID: 0
Iteration no. = 3
Thread ID: 0
Iteration no. = 13
Thread ID: 4
Iteration no. = 14
Thread ID: 4
Iteration no. = 19
Thread ID: 7
Iteration no. = 20
Thread ID: 7
Iteration no. = 4
Thread ID: 1
Iteration no. = 5
Thread ID: 1
Iteration no. = 6
Thread ID: 1
Thread ID: 6
Iteration no. = 15
Thread ID: 5
Iteration no. = 16
Thread ID: 5
Iteration no. = 10
Thread ID: 3
Iteration no. = 11
Thread ID: 3
Iteration no. = 12
Thread ID: 3
16bce0081@sjt516scs042:~/pdc/lab2$ 
```

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.

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

bool prime(int n){
        int i;
        int c = 0;
        for(i=2;i<=n/2;i++){
                if(n%2==0)
                        c = c + 1;
        }

        if(c==0)
                return true;
        else
                return false;
}

int main(){
        bool result = false;
        int num, i;
        int count = 0; // Number of prime numbers
        int list_size;
        printf("Enter number of random numbers for prime check: \n");
        scanf("%d", &list_size);

        #pragma omp parallel
        {
                for(i=1;i<=list_size;i++){
                        num = rand();
                        result = prime(num);
                        if(result==true){
                                count = count + 1;
                                printf("-----------------------------------\n");
                                printf("%d is a prime number from the list.\n", num);
                                printf("Core %d is the one that performed the check.\n",
omp_get_thread_num());
                                printf("-----------------------------------\n");
                        }

                        else
                                printf("%d is not a prime number.\n", num);
```

```
            }
        }

        printf("Total number of prime numbers present in the list = %d\n", count);

        return 0;
}
```



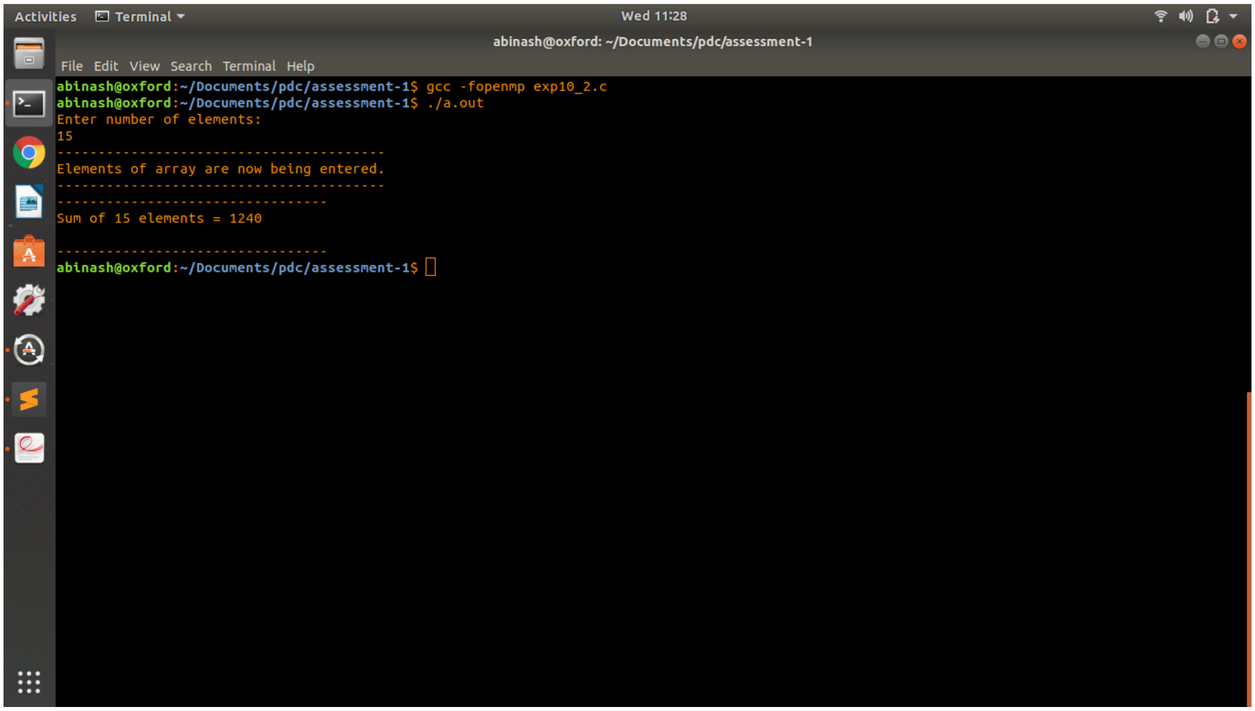
Activities
Terminal
Wed 11:27
abinash@oxford: ~/Documents/pdc/assessment-1
File Edit View Search Terminal Help

```
abinash@oxford:~/Documents/pdc/assessment-1$ gcc -fopenmp exp9.c
abinash@oxford:~/Documents/pdc/assessment-1$ ./a.out
Enter number of random numbers for prime check:
5
1714636915 is not a prime number.
-----------------------------------
1957747793 is a prime number from the list.
Core 1 is the one that performed the check.
-----------------------------------
-----------------------------------
424238335 is a prime number from the list.
Core 0 is the one that performed the check.
-----------------------------------
-----------------------------------
719885386 is a prime number from the list.
Core 1 is the one that performed the check.
-----------------------------------
-----------------------------------
1649760492 is a prime number from the list.
Core 2 is the one that performed the check.
-----------------------------------
1649760492 is not a prime number.
-----------------------------------
1649760492 is a prime number from the list.
Core 3 is the one that performed the check.
-----------------------------------
1649760492 is not a prime number.
Total number of prime numbers present in the list = 5
abinash@oxford:~/Documents/pdc/assessment-1$ 
```

10.

(a) Write an OpenMP program to compute the sum of all the elements in a one dimensional array A using reduction.

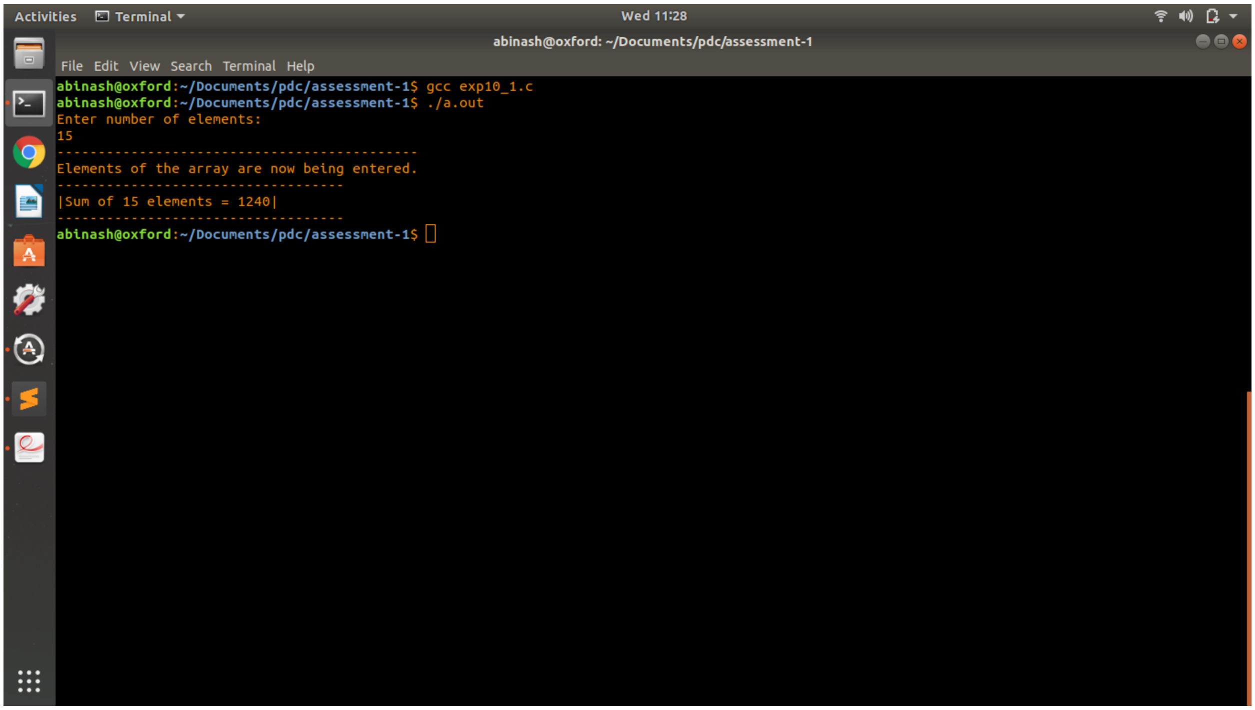
```
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>

int main(){
   int n, i, *ptr;
   int sum = 0;

   printf("Enter number of elements: \n");
   scanf("%d", &n);

   ptr = (int*) malloc(n*sizeof(int));
   if(ptr==NULL){
      printf("Memory is not allocated.\n");
      exit(0);
   }

   printf("----------------------------------------\n");
   printf("Elements of array are now being entered.\n");
   printf("----------------------------------------\n");
   #pragma omp parallel
   {
   for(i=1;i<=n;i++)
      *(ptr+i-1) = i*i;

   #pragma omp parallel for reduction(+:sum)
   for(i=0;i<n;i++)
      sum = sum + *(ptr+i);

   }

   printf("--------------------------------\n");
   printf("Sum of %d elements = %d\n\n", n, sum);
   printf("--------------------------------\n");

   free(ptr);
   return 0;
}
```


Activities
Terminal
Wed 11:28
abinash@oxford: ~/Documents/pdc/assessment-1
File Edit View Search Terminal Help

```
abinash@oxford:~/Documents/pdc/assessment-1$ gcc -fopenmp exp10_2.c
abinash@oxford:~/Documents/pdc/assessment-1$ ./a.out
Enter number of elements:
15
----------------------------------------
Elements of array are now being entered.
----------------------------------------
---------------------------------
Sum of 15 elements = 1240

---------------------------------
abinash@oxford:~/Documents/pdc/assessment-1$ 
```

(b) Create another program that does the same, without using the REDUCE clause. Compare the two versions. [use dynamic memory allocation]

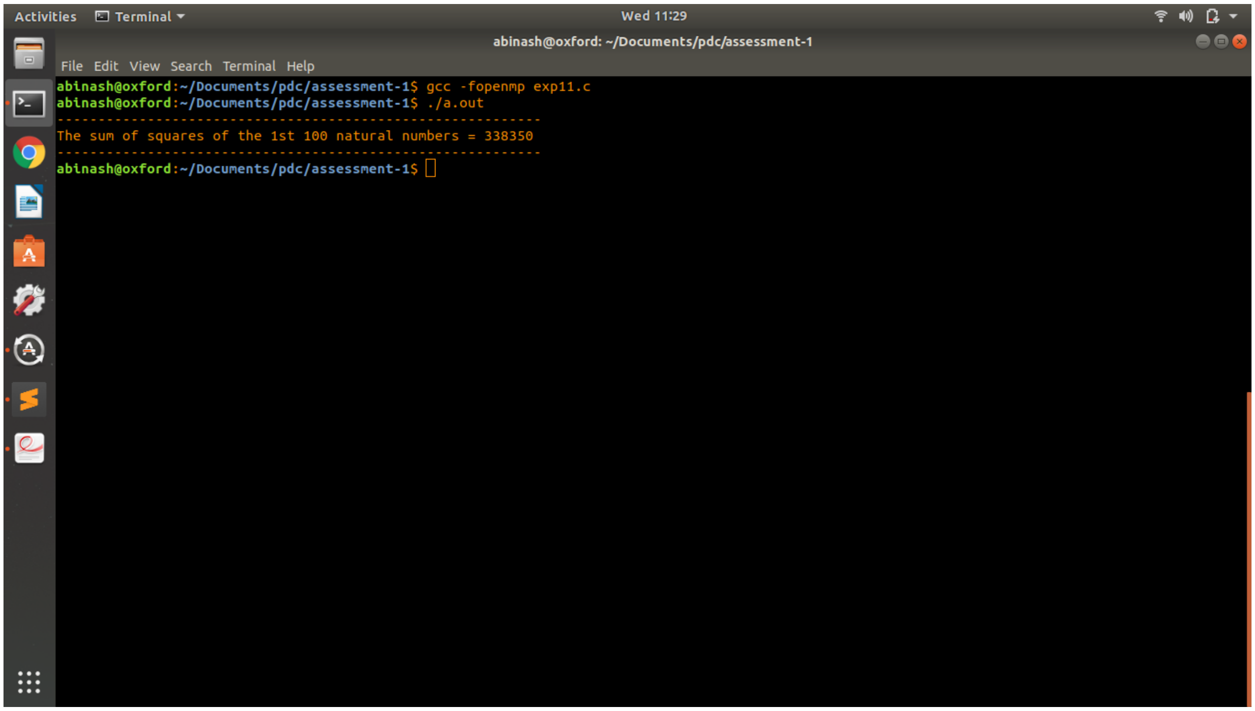
```
#include <stdio.h>
#include <stdlib.h>

int main(){
    int n, i, *ptr;
    int sum = 0;

    printf("Enter number of elements: \n");
    scanf("%d", &n);

    ptr = (int*) malloc(n*sizeof(int));
    if(ptr==NULL){
        printf("Memory is not allocated.\n");
        exit(0);
    }

    printf("----------------------------------------\n");
    printf("Elements of the array are now being entered.\n");
    for(i=1;i<=n;i++)
        *(ptr+i-1) = i*i;

    for(i=0;i<n;i++)
        sum = sum + *(ptr+i);
    printf("-----------------------------------\n");
    printf("|Sum of %d elements = %d|\n", n, sum);
    printf("-----------------------------------\n");

    free(ptr);
    return 0;
}
```


Activities
Terminal
Wed 11:28
abinash@oxford: ~/Documents/pdc/assessment-1
File Edit View Search Terminal Help

```
abinash@oxford:~/Documents/pdc/assessment-1$ gcc exp10_1.c
abinash@oxford:~/Documents/pdc/assessment-1$ ./a.out
Enter number of elements:
15
--------------------------------------------
Elements of the array are now being entered.
-----------------------------------
|Sum of 15 elements = 1240|
-----------------------------------
abinash@oxford:~/Documents/pdc/assessment-1$ 
```

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.

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

int main(){
        int i;
        int sum = 0;
        // #pragma omp parallel
        #pragma omp parallel for reduction(+:sum)
        for(i=1;i<=100;i++)
                sum = sum + i*i;

        printf("------------------------------------------------------------\n");
        printf("The sum of squares of the 1st 100 natural numbers = %d\n", sum);
        printf("------------------------------------------------------------\n");
        return 0;
}
```

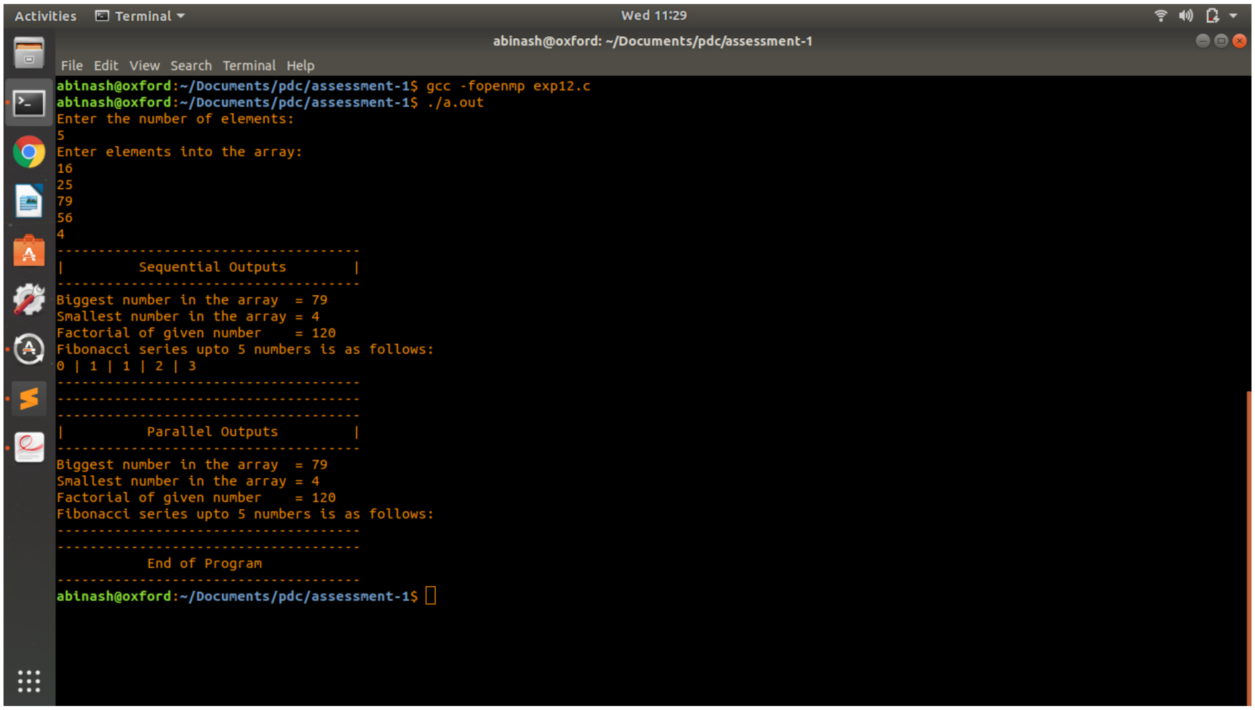


Activities
Terminal
Wed 11:29
abinash@oxford: ~/Documents/pdc/assessment-1
File Edit View Search Terminal Help

```
abinash@oxford:~/Documents/pdc/assessment-1$ gcc -fopenmp exp11.c
abinash@oxford:~/Documents/pdc/assessment-1$ ./a.out
------------------------------------------------------------
The sum of squares of the 1st 100 natural numbers = 338350
------------------------------------------------------------
abinash@oxford:~/Documents/pdc/assessment-1$ 
```

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. Fibonocci 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.

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

int biggest(int a[], int no_of_elements){
        int max = a[0];
        int i;
        for(i=1;i<no_of_elements;i++){
                if(a[i]>max)
                        max = a[i];
        }

        return max;
}

int smallest(int a[], int no_of_elements){
        int min = a[0];
        int i;
        for(i=1;i<no_of_elements;i++){
                if(a[i]<min)
                        min = a[i];
        }

        return min;
}

int factorial(int no_of_elements){
        int i, f = 1;
        for(i=no_of_elements;i>1;i--)
                f = f * i;

        return f;
}

void fibonacci(int no_of_elements){
        int num1 = 0, num2 = 1;
        printf("%d | %d ", num1, num2);
        int i, s = 0;
        for(i=3;i<=no_of_elements;i++){
                s = num1+num2;
                printf("| %d ", s);
                num1 = num2;
                num2 = s;
        }
        printf("\n");
}

int main(){
        int n;
        printf("Enter the number of elements: \n");
        scanf("%d", &n);
        int i;
        printf("Enter elements into the array:\n");
        int arr[n];
        for(i=0;i<n;i++)
                scanf("%d", &arr[i]);

        printf("-------------------------------------\n");
        printf("|        Sequential Outputs        |\n");
        printf("-------------------------------------\n");
        printf("Biggest number in the array  = %d\n", biggest(arr, n));
        printf("Smallest number in the array = %d\n", smallest(arr, n));
        printf("Factorial of given number    = %d\n", factorial(n));
        printf("Fibonacci series upto %d numbers is as follows: \n", n);
        fibonacci(n);
        printf("-------------------------------------\n");
        printf("-------------------------------------\n");

        printf("-------------------------------------\n");
        printf("|         Parallel Outputs         |\n");
        printf("-------------------------------------\n");
        #pragma omp parallel
        {
                #pragma omp sections
                {
                        #pragma omp section
                        printf("Biggest number in the array  = %d\n", biggest(arr, n));

                        #pragma omp section
                        printf("Smallest number in the array = %d\n", smallest(arr, n));

                        #pragma omp section
                        printf("Factorial of given number    = %d\n", factorial(n));

                        #pragma omp section
                        printf("Fibonacci series upto %d numbers is as follows: \n", n);
                }
        }

        printf("-------------------------------------\n");
        printf("-------------------------------------\n");
        printf("          End of Program           \n");
        printf("-------------------------------------\n");

        return 0;
}
```



```
abinash@oxford:~/Documents/pdc/assessment-1$ gcc -fopenmp exp12.c
abinash@oxford:~/Documents/pdc/assessment-1$ ./a.out
Enter the number of elements:
5
Enter elements into the array:
16
25
79
56
4
-------------------------------------
|         Sequential Outputs        |
-------------------------------------
Biggest number in the array  = 79
Smallest number in the array = 4
Factorial of given number    = 120
Fibonacci series upto 5 numbers is as follows:
0 | 1 | 1 | 2 | 3
-------------------------------------
-------------------------------------
-------------------------------------
|          Parallel Outputs         |
-------------------------------------
Biggest number in the array  = 79
Smallest number in the array = 4
Factorial of given number    = 120
Fibonacci series upto 5 numbers is as follows:
-------------------------------------
-------------------------------------
           End of Program
-------------------------------------
abinash@oxford:~/Documents/pdc/assessment-1$
```