**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

{

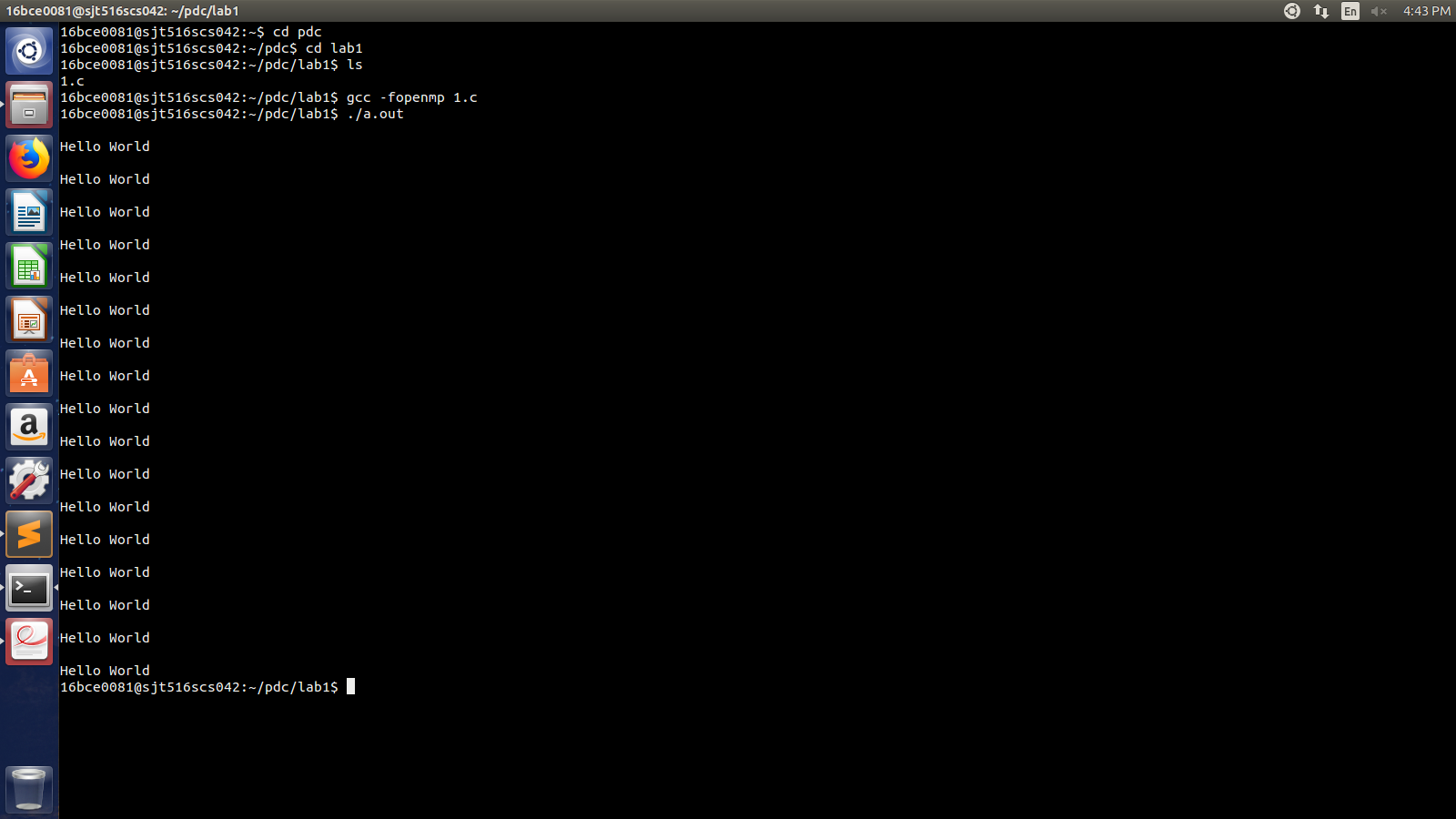
printf("\nHello World\n");

printf("\nHello World\n");

}

return 0;

}



1. 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();

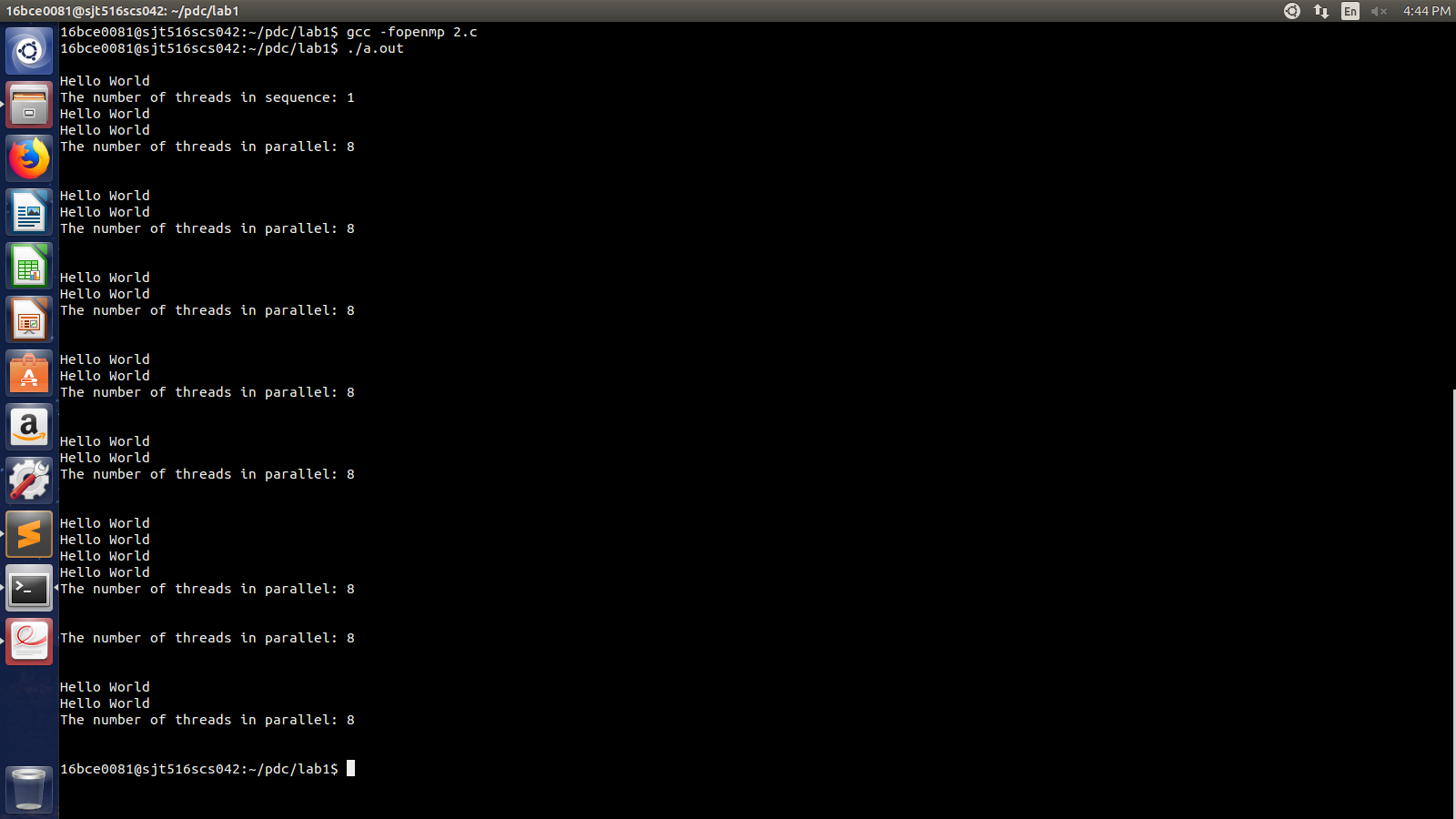
printf("The number of threads in parallel: %d\n", n);

printf("\n\n");

}

return 0;

}



1. Maximum no. of threads

#include <stdio.h>

#include <omp.h>

int main(){

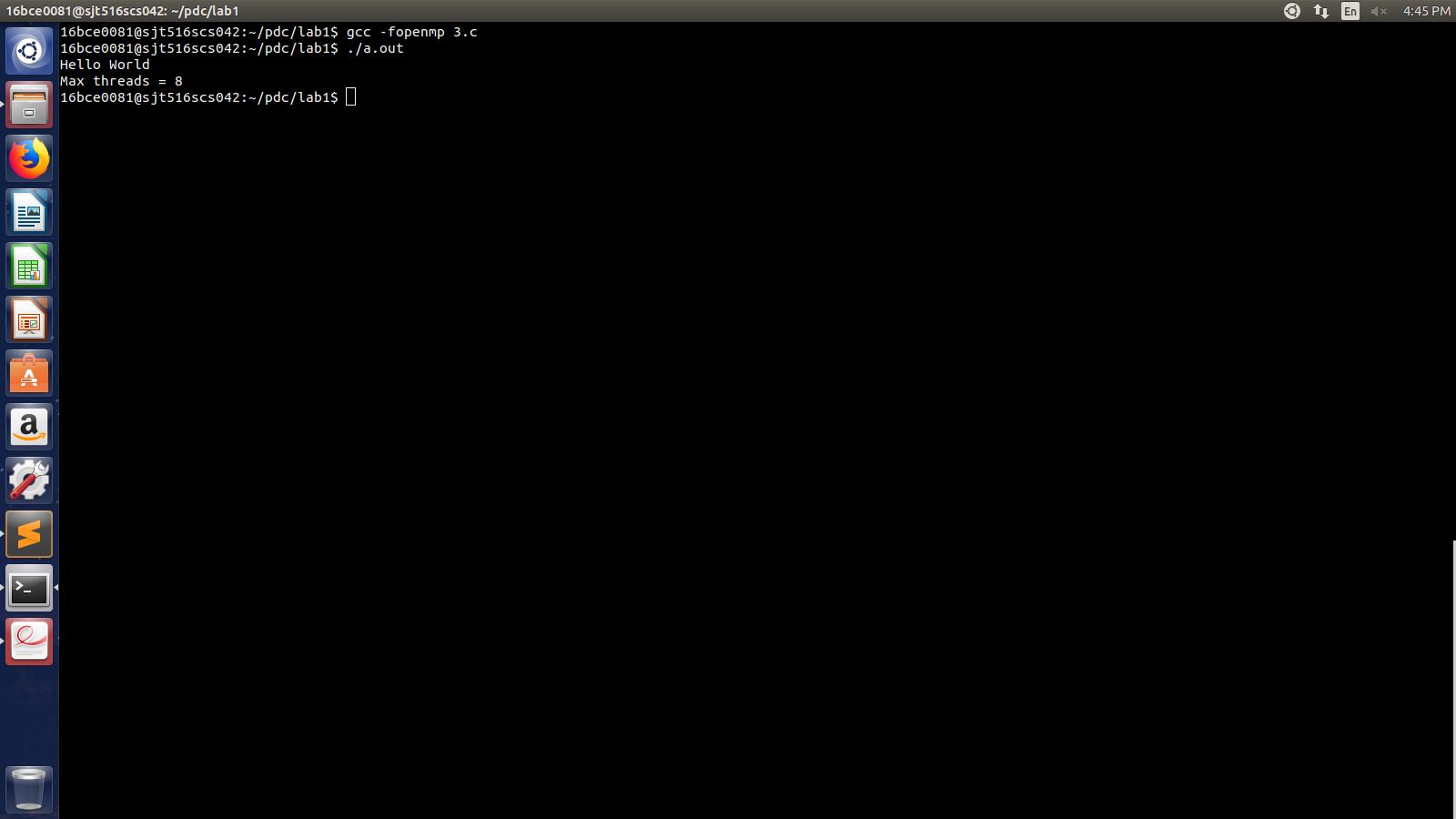
printf("Hello World\n");

int n = omp\_get\_max\_threads();

printf("Max threads = %d\n", n);

return 0;

}



1. Find thread ID

#include <stdio.h>

#include <omp.h>

int main(){

#pragma omp parallel

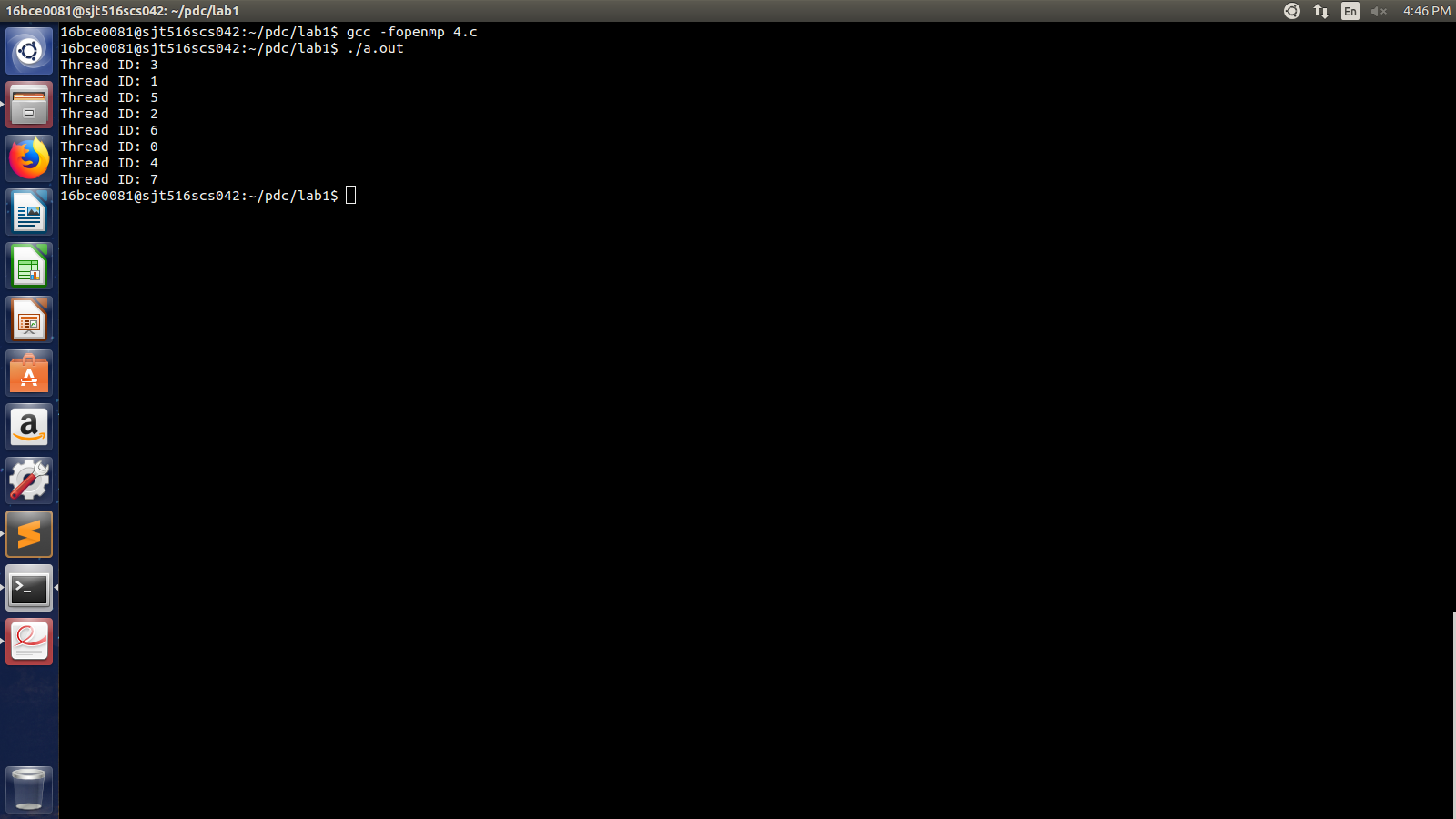
{

printf("Thread ID: %d\n", omp\_get\_thread\_num());

}

return 0;

}



1. Find no. of processor cores in system

#include <stdio.h>

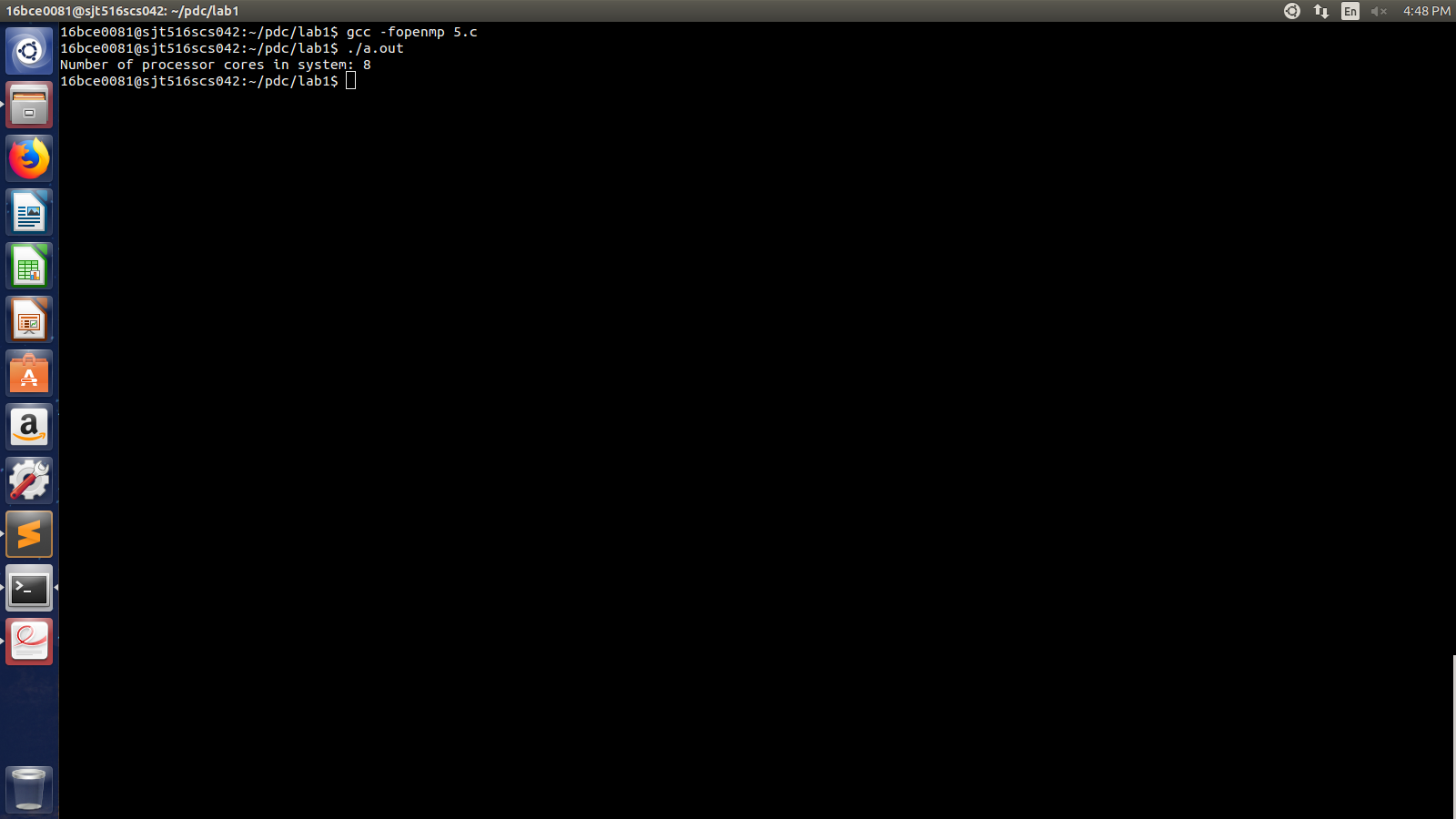
#include <omp.h>

int main(){

printf("Number of processor cores in system: %d\n", omp\_get\_num\_procs());

return 0;

}



1. 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());

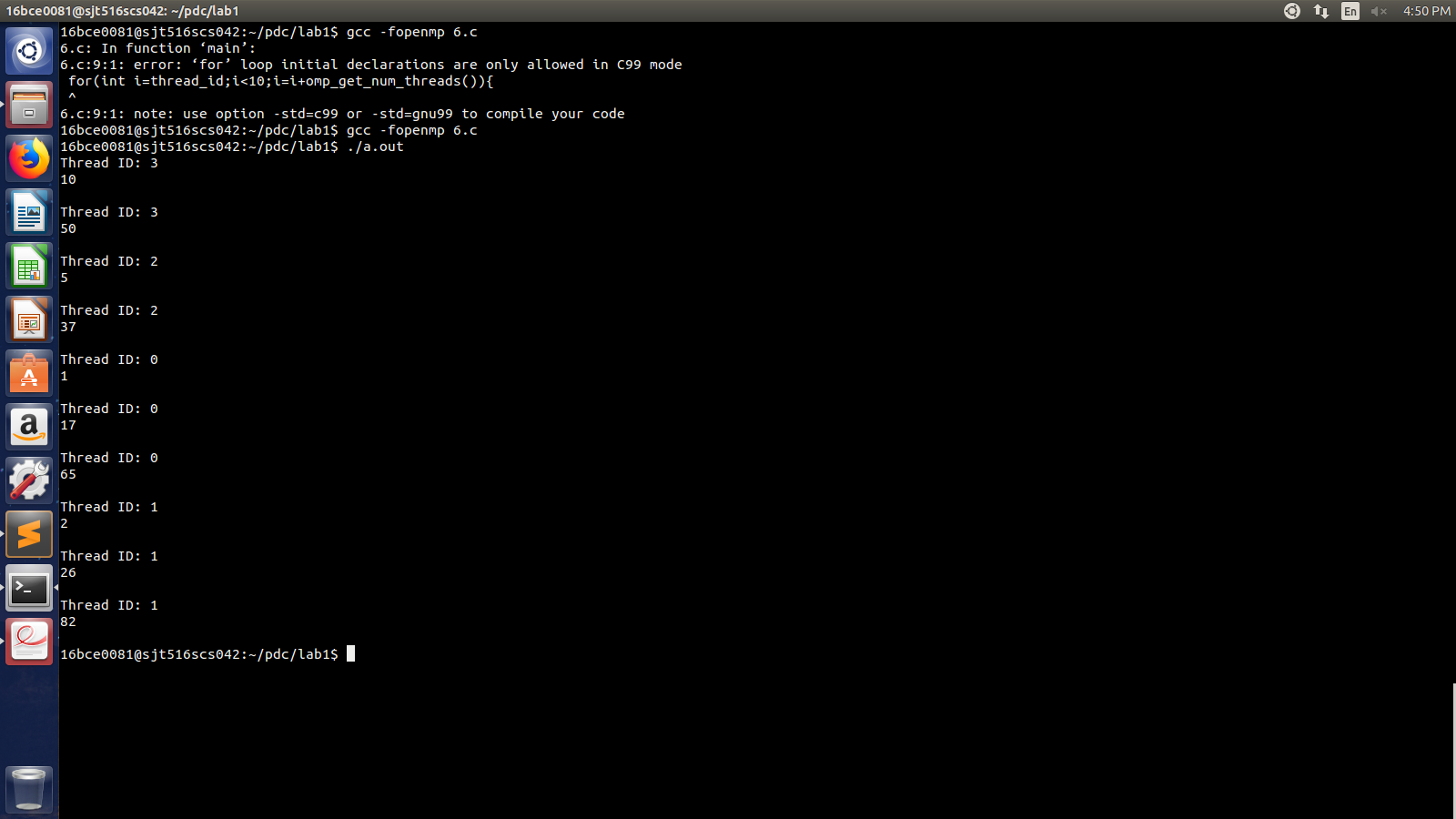
printf("%d\n\n", a[i]);

}

}

return 0;

}



1. 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");

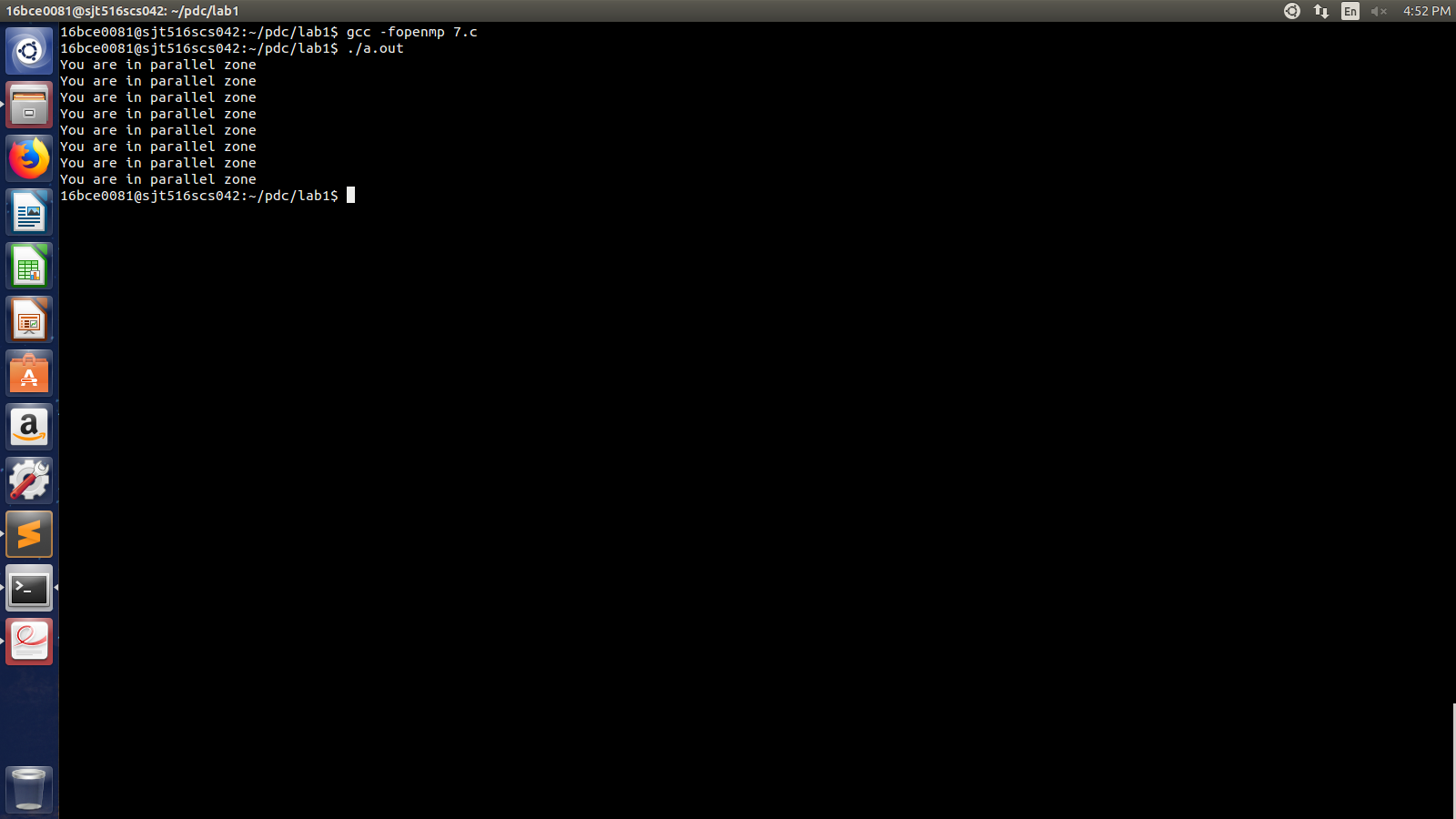
else

printf("You are in sequence zone\n");

}

return 0;

}



1. 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);

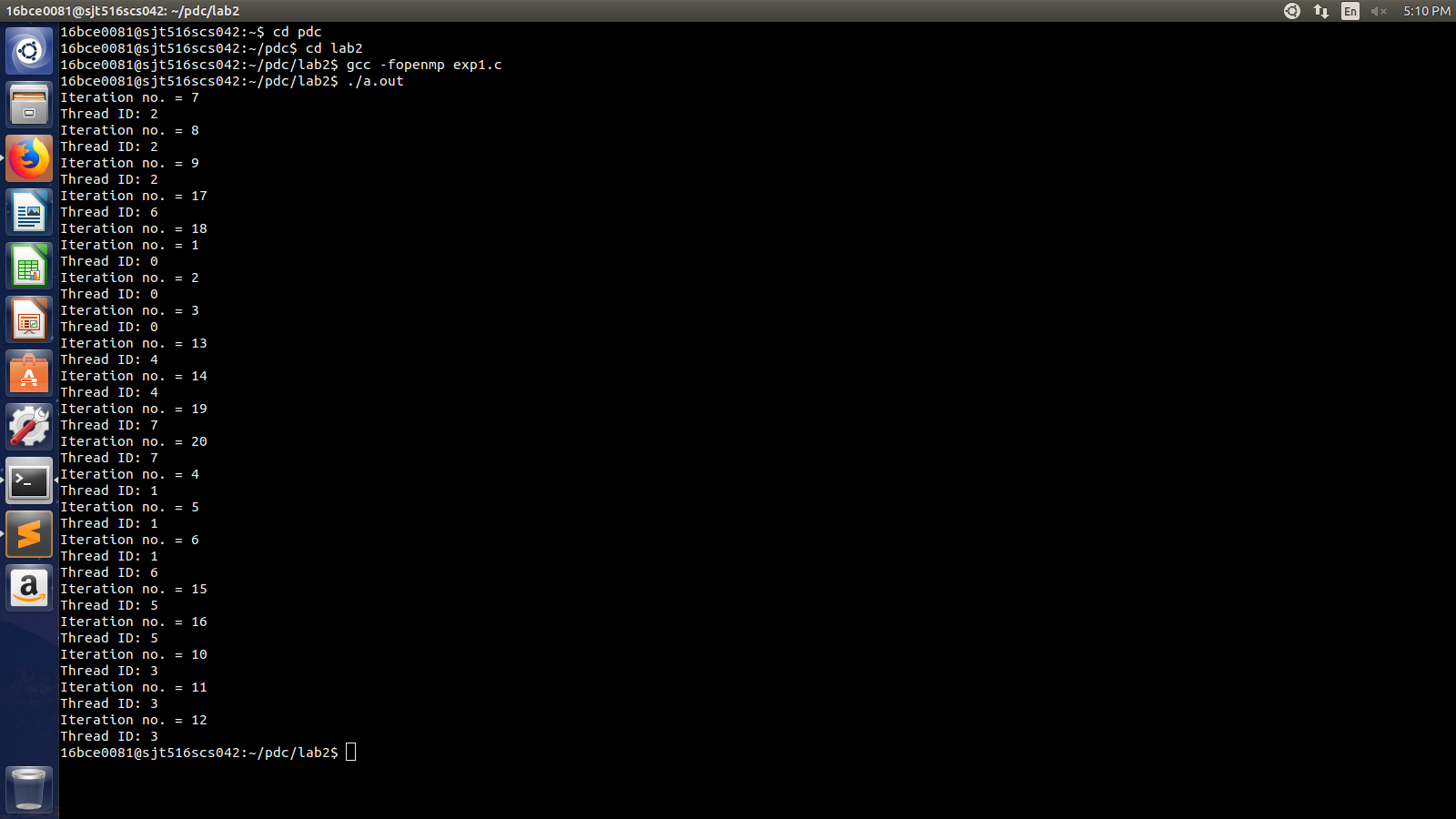
printf("Thread ID: %d\n", omp\_get\_thread\_num());

}

}

return 0;

}



1. 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);

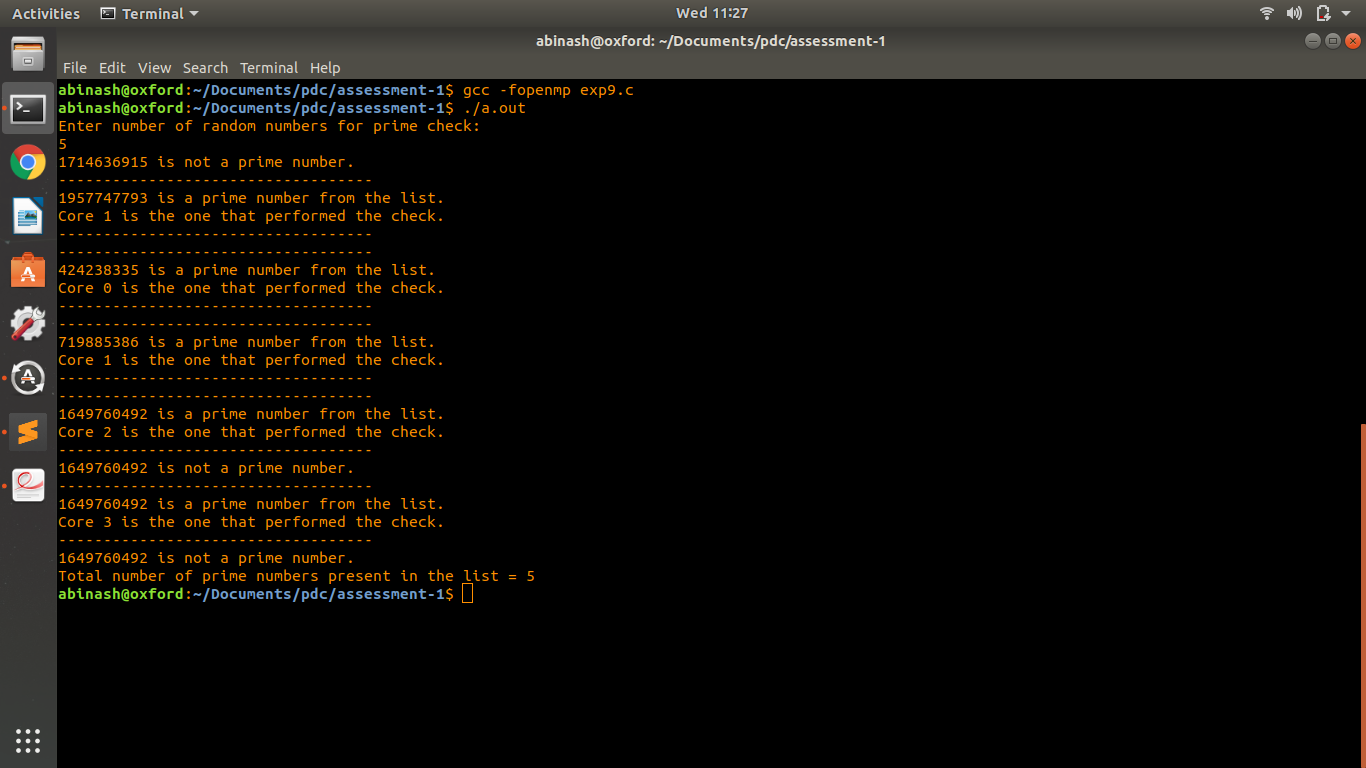
}

}

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

return 0;

}



(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");

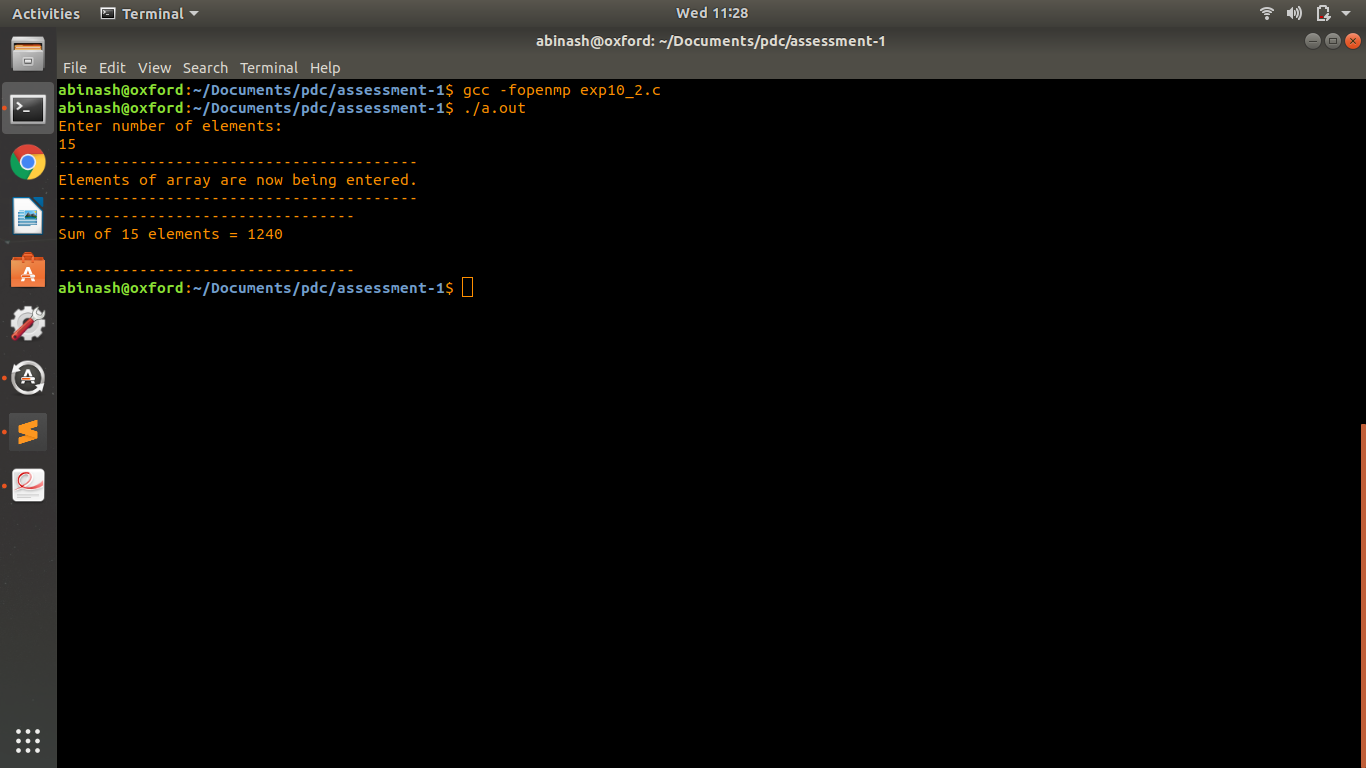
printf("Sum of %d elements = %d\n\n", n, sum);

printf("---------------------------------\n");

free(ptr);

return 0;

}



(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");

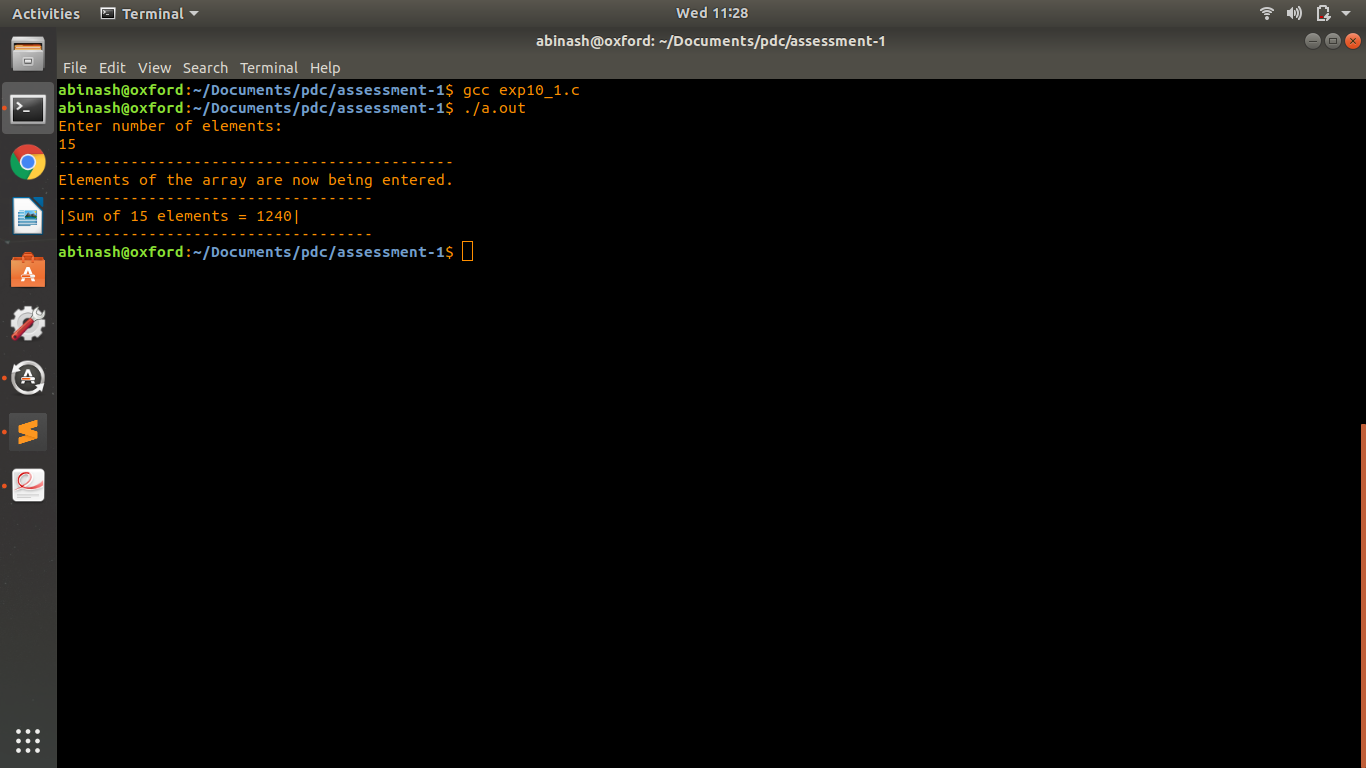
printf("|Sum of %d elements = %d|\n", n, sum);

printf("-----------------------------------\n");

free(ptr);

return 0;

}



1. 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;

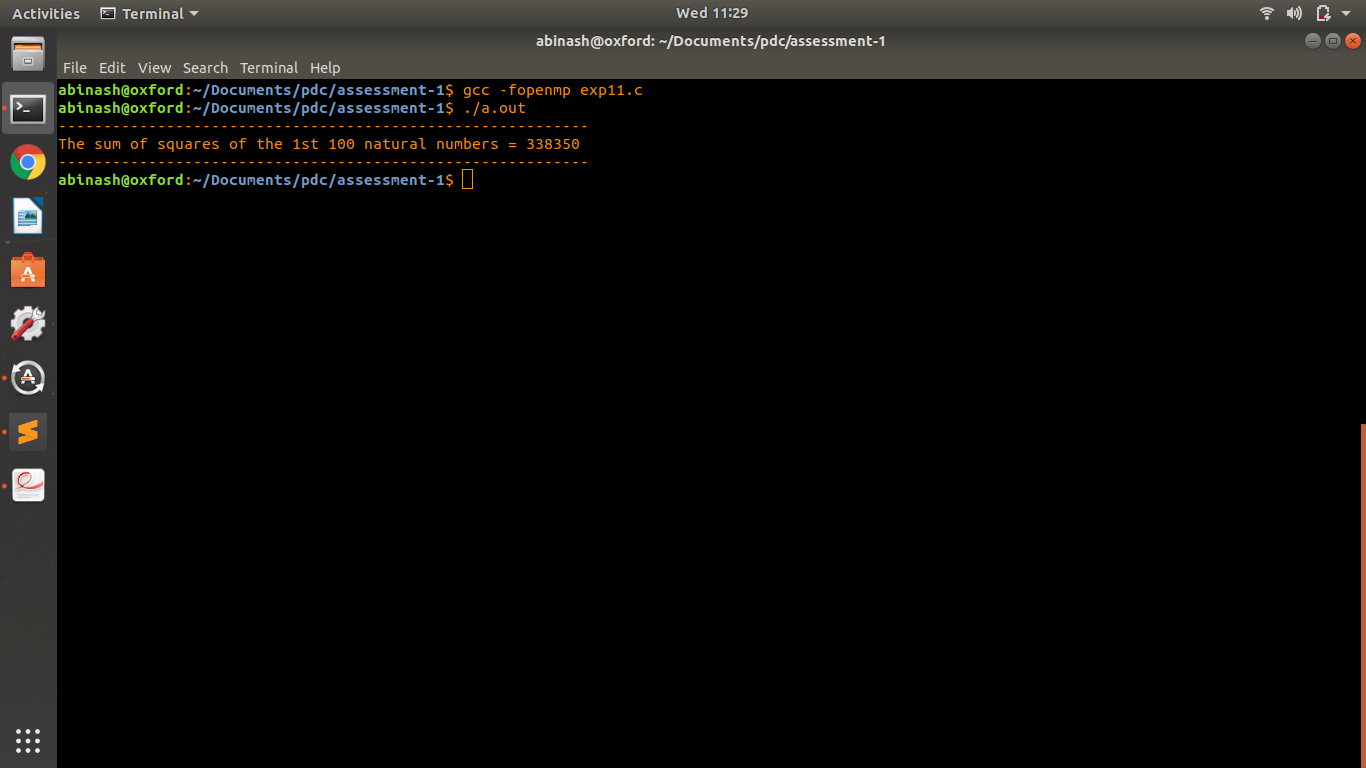
printf("-----------------------------------------------------------\n");

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

printf("-----------------------------------------------------------\n");

return 0;

}



1. 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;

}

