Lab Submission – 02

Arnab Mondal

20BCE1294

Program: B.Tech

Semester: Fall 2022-23

Course: CSE4001 – Parallel and Distributed Computing

Faculty: Dr. Sudha A

Date: 07-08-2022

Exercise: 02

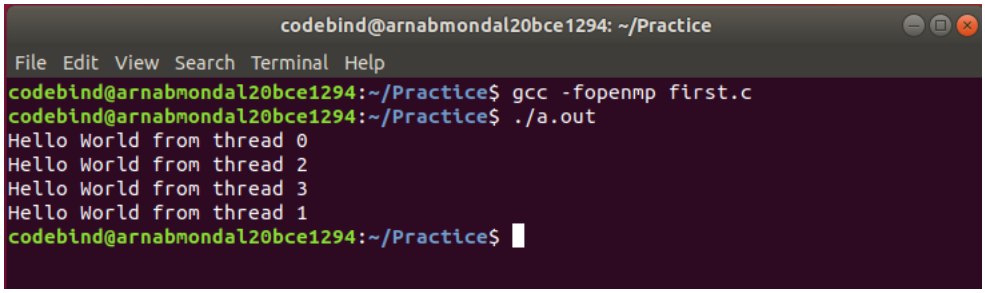
1. **Write hello world program that executes the hello world along with the thread id.**

   Aim: To make a hello world program that executes the hello world along with the thread id.

   Code:

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

int main()
{
	#pragma omp parallel
	{
		printf("Hello World from thread %d\n", omp_get_thread_num());
	}
	return 0;
}
```

   Output:



Explanation:
In the above program the "#pragma omp parallel" creates number of threads as specified. Each thread executes the print statement once. "omp_get_thread_num()" returns the thread id of the current thread in execution which is then printed in the printf() statement.

2. **Perform an OMP program to count the total number of boys and girls in each section (CSE, SW and IT) in two different campuses. Let the boys count be 3,3,3 in campus 1 and 6,7,8 in campus 2 of CSE, SW and IT. The girls count is 4, 4 ,4 in campus 1 and 5,4,3 in campus 2 of CSE, SW and IT.**

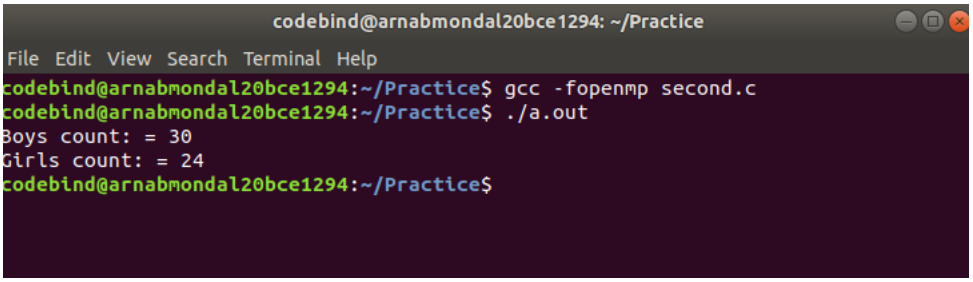
Aim:
To execute an OMP program to count the total number of boys and girls in each section (CSE, SW and IT) in two different campuses. Let the boys count be 3,3,3 in campus 1 and 6,7,8 in campus 2 of CSE, SW and IT. The girls count is 4, 4 ,4 in campus 1 and 5,4,3 in campus 2 of CSE, SW and IT.

Code:

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

int main()
{
        int n1[]={3,3,3};
        int n2[]={6,7,8};
        int n3[] = {4,4,4};
        int n4[] = {5,4,3};
        int boys = 0;
        int girls = 0;
        #pragma omp parallel num_threads(3)
        {
                int id = omp_get_thread_num();
                boys += n1[id] + n2[id];
                girls += n3[id] + n4[id];
        }
        printf("Boys count: = %d\n", boys);
```

```
        printf("Girls count: = %d\n",girls);
    return 0;
}
```

Output:



Explanation:
In the above code "#pragma omp parallel num_threads(3)" creates three threads to be executed in parallel. In the body we store the value of thread if in a variable returned by the function "omp_get_thread_num()". We then add the value of two arrays at index number equal to the thread id returned. Since there are three threads created, thread ids would 0,1,2 which corresponds to the index number of all the elements in an array and hence with each thread execution each element of an array is visited and added to find the total sum.

3. **Let there be two vectors [3, 2, -1] and [5, -4, 3]. Find the dot product of the vectors ($\vec{a}$. $\vec{b}$ = a1a2 + b1b2 + c1c2).**

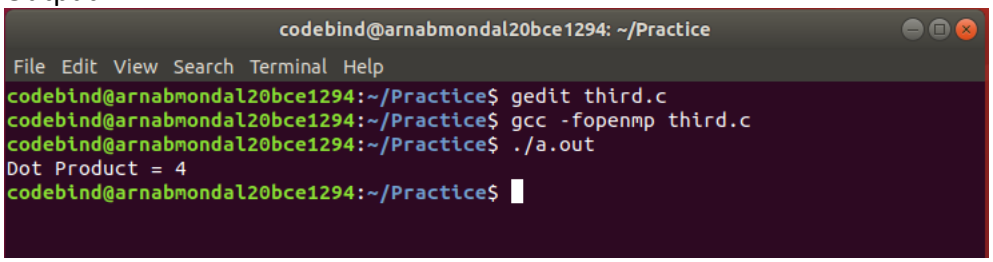
Aim:
To find dot product of two given vectors.

Code:

```
#include <omp.h>

#include <stdio.h>



int main()

{

        int n1[]={3,2,-1};

        int n2[]={5,-4,3};

        int dp=0;
```

```
        #pragma omp parallel num_threads(3)
        {
                int id = omp_get_thread_num();
                dp += n1[id] * n2[id];
        }
        printf("Dot Product = %d\n", dp);
}
```

Output:



Explanation:

In the above code “#pragma omp parallel num_threads(3)” creates three threads to be executed in parallel. In the body we store the value of thread if in a variable returned by the function “omp_get_thread_num()”. We then multiply the value of two arrays at index number equal to the thread id returned. Since there are three threads created, thread ids would 0,1,2 which corresponds to the index number of all the elements in an array and hence with each thread execution each element of an array is visited, multiplied and added to find the dot product.