CSE4001 - Parallel and Distributed Computing, Fall 2019
Vellore Institute of Technology
Instructor: Prof Deebak B D - SCOPE

Lab report

Title of Lab: Beginning with OpenMP
Assessment #: 1
Date: 26|07|2019

Author's name: Gagan Deep Singh
Registration ID: 17BCI0140
Lab section: Friday L59 + L60

AIM:

Write a simple OpenMP program to demonstrate the parallel loop construct.

a. Use OMP_SET_THREAD_NUM( ) and OMP_GET_THREAD_NUM( ) to find the number of processing unit

b. Use function invoke to print ‘Hello World’

c. To examine the above scenario, the functions such as omp_get_num_procs(), omp_set_num_threads(), omp_get_num_threads(), omp_in_parallel(), omp_get_dynamic() and omp_get_nested() are listed and the explanation is given below to explore the concept practically.

omp_set_num_threads() - takes an integer argument and requests that the Operating System provide that number of threads in subsequent parallel regions.

omp_get_num_threads() (integer function) - returns the actual number of threads in the current team of threads.

omp_get_thread_num() (integer function) - returns the ID of a thread, where the ID ranges from 0 to the number of threads minus 1. The thread with the ID of 0 is the master thread.

omp_get_num_procs() - returns the number of processors that are available when the function is called.

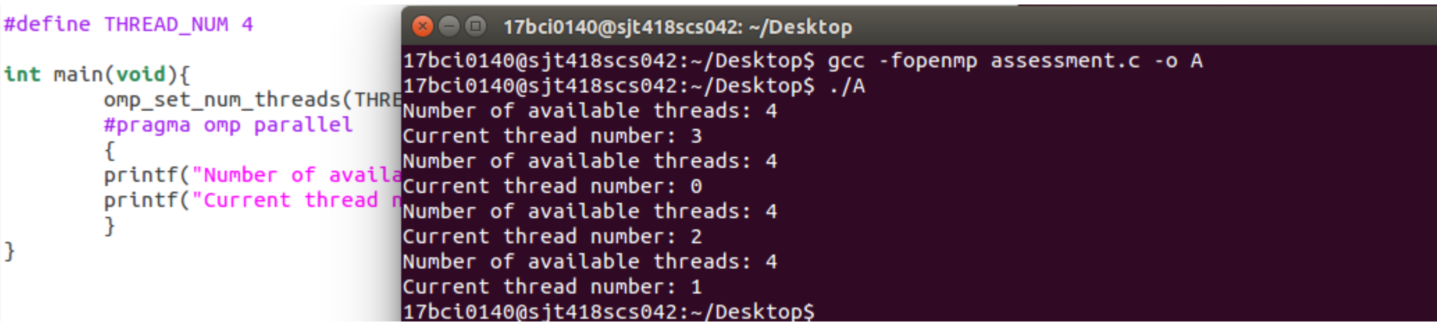
omp_get_dynamic() - returns a value that indicates if the number of threads available in subsequent parallel region can be adjusted by the run time.

omp_get_nested( ) returns a value that indicates if nested parallelism is enabled.

SOURCE CODE: part-a

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

#define THREAD_NUM 4

int main(void){
        omp_set_num_threads(THREAD_NUM);
        #pragma omp parallel
        {
        printf("Number of available threads: %d\n", omp_get_num_threads());
        printf("Current thread number: %d\n", omp_get_thread_num());
        }
}
```

EXECUTION:



```
#define THREAD_NUM 4

int main(void){
        omp_set_num_threads(THRE
        #pragma omp parallel
        {
        printf("Number of availa
        printf("Current thread n
        }
}
```

```
17bci0140@sjt418scs042:~/Desktop$ gcc -fopenmp assessment.c -o A
17bci0140@sjt418scs042:~/Desktop$ ./A
Number of available threads: 4
Current thread number: 3
Number of available threads: 4
Current thread number: 0
Number of available threads: 4
Current thread number: 2
Number of available threads: 4
Current thread number: 1
17bci0140@sjt418scs042:~/Desktop$
```

RESULTS:

omp_set_num_threads() - requests that the Operating System provide a specific number of threads in subsequent parallel regions.

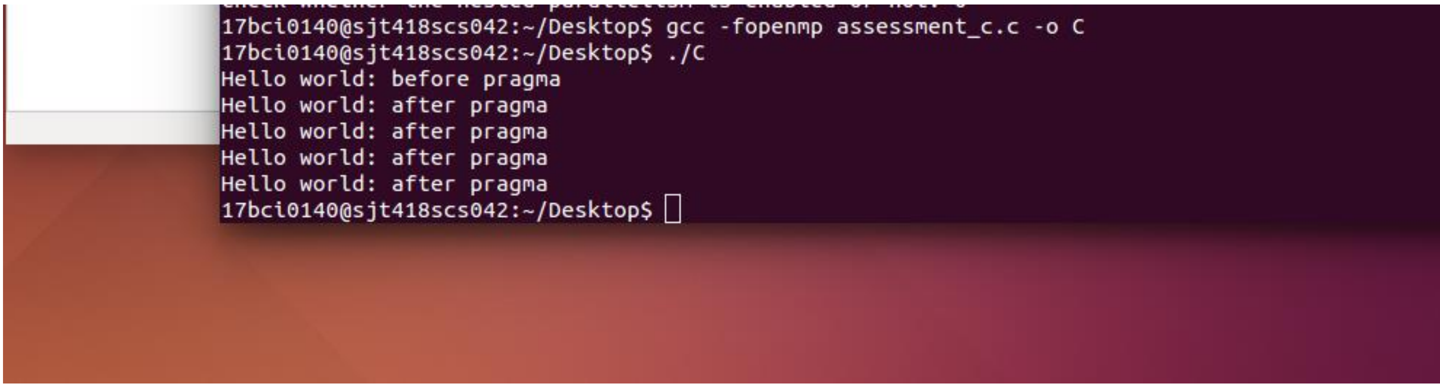
omp_get_num_threads() - returns the number of threads in the current team of threads.

omp_get_thread_num() - returns the ID of a thread.

SOURCE CODE: part-b

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


int main(void){

      printf("Hello world: before pragma\n");
        #pragma omp parallel

      printf("Hello world: after pragma\n");

      return 0;
}
```

EXECUTION:



```
check whether the nested parallelism is enabled or not: 0
17bci0140@sjt418scs042:~/Desktop$ gcc -fopenmp assessment_c.c -o C
17bci0140@sjt418scs042:~/Desktop$ ./C
Hello world: before pragma
Hello world: after pragma
Hello world: after pragma
Hello world: after pragma
Hello world: after pragma
17bci0140@sjt418scs042:~/Desktop$ 
```

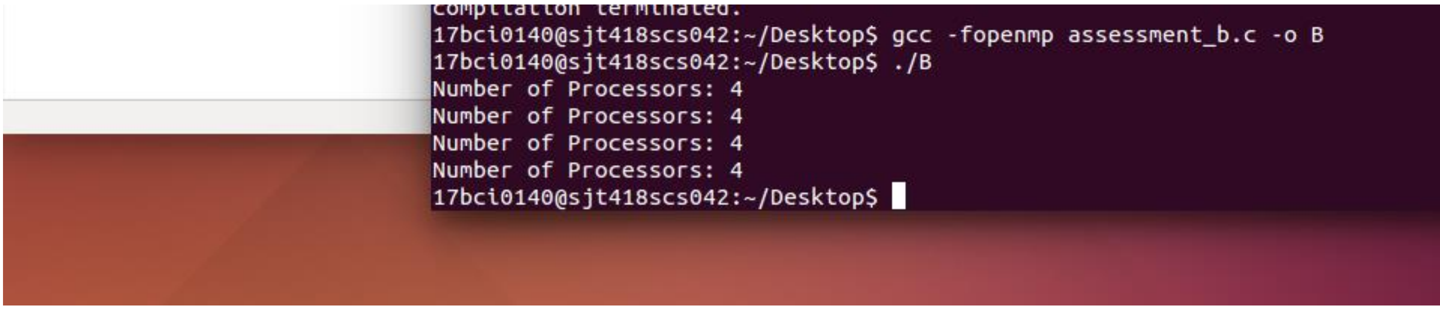
RESULTS:

The “#pragma omp parallel” is used to fork additional threads to carry out the work in parallel.

SOURCE CODE: part-c

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

#define THREAD_NUM 4

int main(void){

        #pragma omp parallel
        {
            printf("Number of Processors: %d\n", omp_get_num_procs());
        }
      return 0;
}
```

EXECUTION:



```
compilation terminated.
17bci0140@sjt418scs042:~/Desktop$ gcc -fopenmp assessment_b.c -o B
17bci0140@sjt418scs042:~/Desktop$ ./B
Number of Processors: 4
Number of Processors: 4
Number of Processors: 4
Number of Processors: 4
17bci0140@sjt418scs042:~/Desktop$ 
```

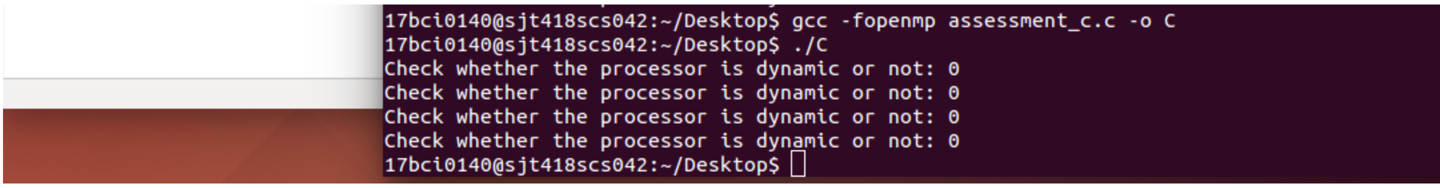
RESULTS:

omp_get_num_procs() - returns the number of processors that are available.

SOURCE CODE: part-c

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

#define THREAD_NUM 4

int main(void){

        #pragma omp parallel
        {
            printf("Check whether the processor is dynamic or not: %d\n",
                  omp_get_dynamic());
        }
      return 0;
}
```

EXECUTION:



```
17bci0140@sjt418scs042:~/Desktop$ gcc -fopenmp assessment_c.c -o C
17bci0140@sjt418scs042:~/Desktop$ ./C
Check whether the processor is dynamic or not: 0
Check whether the processor is dynamic or not: 0
Check whether the processor is dynamic or not: 0
Check whether the processor is dynamic or not: 0
17bci0140@sjt418scs042:~/Desktop$
```

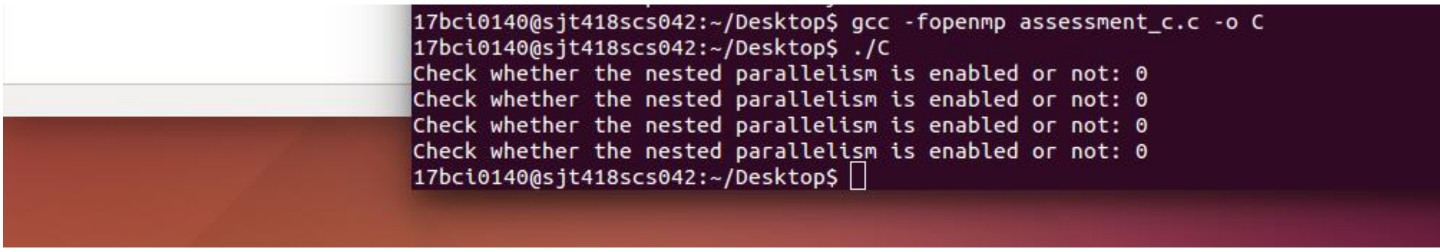
RESULTS:

The omp_get_dynamic function returns ‘1’, if dynamic thread adjustment is enabled. Otherwise, returns ‘0’.

SOURCE CODE:

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

#define THREAD_NUM 4

int main(void){

        #pragma omp parallel
        {
            printf("Check whether the nested parallelism is enabled or
not: %d\n",                             omp_get_nested());
        }
      return 0;
}
```

EXECUTION:



```
17bci0140@sjt418scs042:~/Desktop$ gcc -fopenmp assessment_c.c -o C
17bci0140@sjt418scs042:~/Desktop$ ./C
Check whether the nested parallelism is enabled or not: 0
Check whether the nested parallelism is enabled or not: 0
Check whether the nested parallelism is enabled or not: 0
Check whether the nested parallelism is enabled or not: 0
17bci0140@sjt418scs042:~/Desktop$
```

RESULTS:

The omp_get_nested function returns ‘1’, if nested parallelism is enabled and ‘0’ if disabled.