Lab Assignment-1

Basic functions of OpenMP and creating a c program for dot product using OpenMP

Done By: Arshdeep Singh Bhatia

Registration number: 19BCB0086

Submitted to: Prof. Balamurugan R

VIT®

Vellore Institute of Technology

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

Task 1

Implement the OpenMP standard includes the following functions and data types using C and include the appropriate declarations of the routines in your source code and explain it.

1. omp_init_lock

Initializes a simple lock. Needs variable of type `omp_lock_t`.

```
void omp_init_lock (
   omp_lock_t *lock
);
```

2. omp_get_thread_num

Returns the thread number of the thread executing within its thread team.

```
int omp_get_thread_num ();
```

3. omp_set_lock

Blocks thread execution until a lock is available.

```
void omp_set_lock(
   omp_lock_t *lock
);
```

4. omp_unset_lock

Releases a lock

```
void omp_unset_lock(
   omp_lock_t *lock
);
```

5. omp_destroy_lock

Uninitializes a lock. Works on variable of type omp_lock_t that was initialized with omp_init_lock.

```
void omp_destroy_lock (
   omp_lock_t *lock
);
```

CODE involving functions 1-5

```
#include <stdio.h>
#include <omp.h>
omp_lock_t my_lock;
int main()
{
   omp_init_lock(&my_lock);
   #pragma omp parallel num_threads(4)
   {
      int tid = omp_get_thread_num( );
      int i, j;
```

```
      for (i = 0; i < 2; ++i) {
         omp_set_lock(&my_lock);
         printf("Thread %d - starting locked region\n", tid);
         printf("Thread %d - ending locked region\n", tid);
         omp_unset_lock(&my_lock);
      }
   }
   omp_destroy_lock(&my_lock);
}
```

Output

```
arshdeep@arshdeep-HP-Laptop-14s-cr1xxx:~/Desktop/pdc-lab/da1$ ./init_lock
Thread 1 - starting locked region
Thread 1 - ending locked region
Thread 1 - starting locked region
Thread 1 - ending locked region
Thread 3 - starting locked region
Thread 3 - ending locked region
Thread 3 - starting locked region
Thread 3 - ending locked region
Thread 2 - starting locked region
Thread 2 - ending locked region
Thread 2 - starting locked region
Thread 2 - ending locked region
Thread 0 - starting locked region
Thread 0 - ending locked region
Thread 0 - starting locked region
Thread 0 - ending locked region
```

6. omp_set_dynamic

Indicates that the number of threads available in upcoming parallel regions can be adjusted by the run time. A value that indicates if the number of threads available in upcoming parallel regions can be adjusted by the runtime. If nonzero, the runtime can adjust the number of threads, if zero, the runtime won't dynamically adjust the number of threads.

```
void omp_set_dynamic(
   int val
);
```

7. omp_set_num_threads

Sets the number of threads in upcoming parallel regions, unless overridden by a num_threads clause.

```
void omp_set_num_threads(
   int num_threads
);
```

8. omp_get_dynamic

Returns a value that indicates if the number of threads available in upcoming parallel regions can be adjusted by the run time.

```
int omp_get_dynamic();
```

Code involving functions 6-8

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

int main()
{
    omp_set_dynamic(9);
    omp_set_num_threads(4);
    printf("%d\n", omp_get_dynamic( ));
    #pragma omp parallel
        #pragma omp master
        {
            printf("%d\n", omp_get_dynamic( ));
        }
}
```

Output

```
arshdeep@arshdeep-HP-Laptop-14s-cr1xxx:~/Desktop/pdc-lab/da1$ ./get_dynamic
1
1
arshdeep@arshdeep-HP-Laptop-14s-cr1xxx:~/Desktop/pdc-lab/da1$
```

9. omp_get_max_threads

Returns an integer that is equal to or greater than the number of threads that would be available if a parallel region without num_threads were defined at that point in the code.

```
int omp_get_max_threads()
```

```
#include <stdio.h>
#include <omp.h>
int main( )
{
    omp_set_num_threads(8);
    printf("%d\n", omp_get_max_threads( ));
    #pragma omp parallel
        #pragma omp master
        {
            printf("%d\n", omp_get_max_threads( ));
        }

    printf("%d\n", omp_get_max_threads( ));

    #pragma omp parallel num_threads(3)
        #pragma omp master
        {
```

```
            printf("%d\n", omp_get_max_threads( ));
        }

    printf("%d\n", omp_get_max_threads( ));
}
```

Output

```
arshdeep@arshdeep-HP-Laptop-14s-cr1xxx:~/Desktop/pdc-lab/da1$ ./max_thread
8
8
8
8
8
```

10. omp_set_nested

Enables nested parallelism. A nonzero value enables nested parallelism, while zero disables nested parallelism.

```
void omp_set_nested(
   int val
);
```

11. omp_get_nested

Returns a value that indicates if nested parallelism is enabled. A nonzero value means nested parallelism is enabled.

```
int omp_get_nested( );
```

CODE using functions 10 and 11

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

int main( )
{

    omp_set_nested(1);
    omp_set_num_threads(4);
    printf("%d\n", omp_get_nested( ));

    #pragma omp parallel
        #pragma omp master
        {
            printf("%d\n", omp_get_nested( ));
        }

}
```

Output

```
arshdeep@arshdeep-HP-Laptop-14s-cr1xxx:~/Desktop/pdc-lab/da1$ ./get_nested
1
1
arshdeep@arshdeep-HP-Laptop-14s-cr1xxx:~/Desktop/pdc-lab/da1$
```

12. omp_get_num_procs

Returns the number of processors that are available when the function is called. Basically 8 for my computer.

```
int omp_get_num_procs();
```

```
#include <stdio.h>
#include <omp.h>
int main( )
{
    printf("%d\n", omp_get_num_procs( ));
    #pragma omp parallel
        #pragma omp master
        {
            printf("%d\n", omp_get_num_procs( ));
        }
}
```

Output

```
arshdeep@arshdeep-HP-Laptop-14s-cr1xxx:~/Desktop/pdc-lab/da1$ ./get_num_process
8
8
arshdeep@arshdeep-HP-Laptop-14s-cr1xxx:~/Desktop/pdc-lab/da1$
```

13. omp_get_num_threads

Returns the number of threads in the parallel region.

```
int omp_get_num_threads( );
```

```
int main()
{
    omp_set_num_threads(4);
    printf("%d\n", omp_get_num_threads( ));
    #pragma omp parallel
        #pragma omp master
        {
            printf("%d\n", omp_get_num_threads( ));
        }
```

```
    printf("%d\n", omp_get_num_threads( ));

    #pragma omp parallel num_threads(3)
        #pragma omp master
        {
            printf("%d\n", omp_get_num_threads( ));
        }

    printf("%d\n", omp_get_num_threads( ));
}
```

Output

```
arshdeep@arshdeep-HP-Laptop-14s-cr1xxx:~/Desktop/pdc-lab/da1$ ./get_num_threads
1
4
1
3
1
arshdeep@arshdeep-HP-Laptop-14s-cr1xxx:~/Desktop/pdc-lab/da1$
```

14. omp_get_default_device ()

gets the value of the default device

CODE

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

int main( )
{
    omp_set_default_device(1);
    printf("%d\n", omp_get_default_device( ));
}
```

OUTPUT

```
arshdeep@arshdeep-HP-Laptop-14s-cr1xxx:~/Desktop/pdc-lab/da1$ ./get_default_device
1
arshdeep@arshdeep-HP-Laptop-14s-cr1xxx:~/Desktop/pdc-lab/da1$
```

15. omp_get_wtick

Returns the number of seconds between processor clock ticks.

```
double omp_get_wtick( );
```

16. omp_get_wtime

Returns a value in seconds of the time elapsed from some point. Returns a value in seconds of the time elapsed from some arbitrary, but consistent point.

```
double omp_get_wtime( );
```

CODE with function 15 and 16

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

int main() {
    double start = omp_get_wtime( );
    double end = omp_get_wtime( );
    double wtick = omp_get_wtick( );
    printf("start = %.16g\nend = %.16g\ndiff = %.16g\n",
             start, end, end - start);
    printf("wtick = %.16g\n1/wtick = %.16g\n",
             wtick, 1.0 / wtick);
}
```

Output

```
arshdeep@arshdeep-HP-Laptop-14s-cr1xxx:~/Desktop/pdc-lab/da1$ ./get_wtick
start = 1495.797890144
end = 1495.797890625
diff = 4.810001428268151e-07
wtick = 1e-09
1/wtick = 999999999.9999999
arshdeep@arshdeep-HP-Laptop-14s-cr1xxx:~/Desktop/pdc-lab/da1$
```

17. omp_in_parallel

Returns nonzero if called from within a parallel region.

```
int omp_in_parallel( );
```

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

int main( )
{
    omp_set_num_threads(4);
    printf("%d\n", omp_in_parallel( ));
```

```
    #pragma omp parallel
        #pragma omp master
        {
            printf("%d\n", omp_in_parallel( ));
        }
}
```

Output

```
arshdeep@arshdeep-HP-Laptop-14s-cr1xxx:~/Desktop/pdc-lab/da1$ ./in_parallel
0
1
arshdeep@arshdeep-HP-Laptop-14s-cr1xxx:~/Desktop/pdc-lab/da1$
```

18. omp_init_nest_lock

Initializes a lock.

Parameter used is variable of type omp_nest_lock_t.

```
void omp_init_nest_lock(
   omp_nest_lock_t *lock
);
```

19. omp_set_nest_lock

Blocks thread execution until a lock is available. Works on variable of type omp_nest_lock_t that was initialized with omp_init_nest_lock

```
void omp_set_nest_lock(
   omp_nest_lock_t *lock
);
```

20. omp_unset_nest_lock

Releases a nestable lock. Works on variable of type omp_nest_lock_t that was initialized with omp_init_nest_lock, owned by the thread and executing in the function.

```
void omp_unset_nest_lock(
   omp_nest_lock_t *lock
);
```

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

omp_nest_lock_t my_lock;

void Test() {
   int tid = omp_get_thread_num( );
   omp_set_nest_lock(&my_lock);
```

```
   printf("Thread %d - starting nested locked region\n", tid);
   printf("Thread %d - ending nested locked region\n", tid);
   omp_unset_nest_lock(&my_lock);
}
int main() {
   omp_init_nest_lock(&my_lock);

   #pragma omp parallel num_threads(4)
   {
      int i, j;
      for (i = 0; i < 5; ++i) {
         omp_set_nest_lock(&my_lock);
            if (i % 3)
               Test();
            omp_unset_nest_lock(&my_lock);
        }
    }
    omp_destroy_nest_lock(&my_lock);
}
```

Output

```
arshdeep@arshdeep-HP-Laptop-14s-cr1xxx:~/Desktop/pdc-lab/da1$ ./omp_init_nest_lock
Thread 1 - starting nested locked region
Thread 1 - ending nested locked region
Thread 1 - starting nested locked region
Thread 1 - ending nested locked region
Thread 1 - starting nested locked region
Thread 1 - ending nested locked region
Thread 2 - starting nested locked region
Thread 2 - ending nested locked region
Thread 2 - starting nested locked region
Thread 2 - ending nested locked region
Thread 3 - starting nested locked region
Thread 3 - ending nested locked region
Thread 3 - starting nested locked region
Thread 3 - ending nested locked region
Thread 3 - starting nested locked region
Thread 3 - ending nested locked region
Thread 0 - starting nested locked region
Thread 0 - ending nested locked region
Thread 0 - starting nested locked region
Thread 0 - ending nested locked region
Thread 0 - starting nested locked region
Thread 0 - ending nested locked region
Thread 2 - starting nested locked region
Thread 2 - ending nested locked region
```

21. omp_test_lock

Attempts to set a lock but doesn't block thread execution. Zero means failed and nonzero means lock acquired.

```
int omp_test_lock(
   omp_lock_t *lock
);
```

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

omp_lock_t simple_lock;

int main() {
    omp_init_lock(&simple_lock);

    #pragma omp parallel num_threads(4)
    {
        int tid = omp_get_thread_num();

        while (!omp_test_lock(&simple_lock))
            printf("Thread %d - failed to acquire simple_lock\n",
                   tid);

        printf("Thread %d - acquired simple_lock\n", tid);

        printf("Thread %d - released simple_lock\n", tid);
        omp_unset_lock(&simple_lock);
    }

    omp_destroy_lock(&simple_lock);
}
```

Output

```
arshdeep@arshdeep-HP-Laptop-14s-cr1xxx:~/Desktop/pdc-lab/da1$ gcc -o omp_test_lock  -fopenmp omp_test_lock.c
arshdeep@arshdeep-HP-Laptop-14s-cr1xxx:~/Desktop/pdc-lab/da1$ ./omp_test_lock
Thread 0 - acquired simple_lock
Thread 0 - released simple_lock
Thread 1 - failed to acquire simple_lock
Thread 1 - acquired simple_lock
Thread 1 - released simple_lock
arshdeep@arshdeep-HP-Laptop-14s-cr1xxx:~/Desktop/pdc-lab/da1$
arshdeep@arshdeep-HP-Laptop-14s-cr1xxx:~/Desktop/pdc-lab/da1$
arshdeep@arshdeep-HP-Laptop-14s-cr1xxx:~/Desktop/pdc-lab/da1$
arshdeep@arshdeep-HP-Laptop-14s-cr1xxx:~/Desktop/pdc-lab/da1$
arshdeep@arshdeep-HP-Laptop-14s-cr1xxx:~/Desktop/pdc-lab/da1$
```

Task 2

Using OpenMP, Design, develop and run a multi-threaded program to perform dot product and explain it

CODE

```
#include <omp.h>
#include <stdio.h>
int main()
{
    int i, n, chunk;
    float a[3], b[3], result;
    result = 0.0;
    a[0] = 1;
    a[1] = 2;
    a[2] = 2;
    // vector 1i+2j+2k
    printf("Vector 1 is ");
    for (int i=0;i<3;i++)
    {
        printf("%f ", a[i]);
    }

    b[0] = 3;
    b[1] = 4;
    b[2] = -2;
    // vector 3i+4j-2k
    printf("\nVector 2 is ");
    for (int i=0;i<3;i++)
    {
        printf("%f ", b[i]);
    }
#pragma omp parallel for default(shared) private(i) schedule(static, chunk)
reduction(+ : result)

    for (i = 0; i < 3; i++)
        result += (a[i] * b[i]);

    printf("\nDot product result= %f\n", result);
}
```

Output

```
arshdeep@arshdeep-HP-Laptop-14s-cr1xxx:~/Deskto
Vector 1 is 1.000000 2.000000 2.000000
Vector 2 is 3.000000 4.000000 -2.000000
Dot product result= 7.000000
```