Name : Manthan Dhole

Id : 201080020 Subject : Parallel Computing

Experiment 4

AIM:

Implementation using OpenMP.
i. Fork Join model, ii. Producer
Consumer problem, iii. Matrix
Multiplication, iv. find prime
number,
v. Largest Element in an array and vi.
Pi calculation

THEORY:

What is OpenMP?
OpenMP is a standard parallel programming API for shared memory environments, written in C, C++, or FORTRAN. It consists of a set of compiler directives with a "lightweight" syntax, library routines, and environment variables that influence run-time behavior. OpenMP is governed by the OpenMP Architecture Review Board (or OpenMP ARB), and is defined by several hardware and software vendors.
OpenMP behavior is directly dependent on the OpenMP implementation. Capabilities of this implementation can enable the programmer to separate the program into serial and parallel regions rather than just concurrently running threads, hides stack management, and provides synchronization of constructs. That being said, OpenMP will not guarantee speedup, parallelize dependencies, or prevent data racing. Data racing, keeping track of dependencies, and working towards a speedup are all up to the programmer.

Why do we use OpenMP?
OpenMP has received considerable attention in the past decade and is considered by many to be an ideal solution for parallel programming because it has unique advantages as a mainstream directive-based programming model.
First of all, OpenMP provides a cross-platform, cross-compiler solution. It supports lots of platforms such as Linux, macOS, and Windows. Mainstream compilers including GCC, LLVM/Clang, Intel Fortran, and C/C++ compilers provide OpenMP good support. Also, with the rapid development of OpenMP, many researchers and computer vendors are constantly exploring how to optimize the execution efficiency of OpenMP programs and continue to propose improvements for existing compilers or develop new compilers. What's more. OpenMP

Name : Manthan Dhole

Id : 201080020 Subject : Parallel Computing

is a standard specification, and all compilers that support it implement the same set of standards, and there are no portability issues.

Secondly, using OpenMP can be very convenient and flexible to modify the number of threads. To solve the scalability problem of the number of CPU cores. In the multi-core era, the number of threads needs to change according to the number of CPU cores. OpenMP has irreplaceable advantages in this regard.

Thirdly, using OpenMP to create threads is considered to be convenient and relatively easy because it does not require an entry function, the code within the same function can be decomposed into multiple threads for execution, and a for loop can be decomposed into multiple threads for execution. If OpenMP is not used, when the operating system API creates a thread, the code in a function needs to be manually disassembled into multiple thread entry functions. To sum up, OpenMP has irreplaceable advantages in parallel programming. More and more new directives are being added to achieve more functions, and they are playing an important role on many different platforms

NOTEBOOK LINK:
https://colab.research.google.com/drive/1I69Ap2SycBoVskpegLON_Mb1IVmLk6qx#scrollTo=h5ePDwBGDZsN

CODE:

i. Fork Join model,

```
code1 = """ // OpenMP program to print
Hello World
// using C language
 // OpenMP header
#include <omp.h>
#include <stdio.h> #include
<stdlib.h>  int main(int argc,
char* argv[]) {
   // Beginning of parallel region
   #pragma omp parallel    {        printf("Hello
World... from thread = %d \\n",
omp_get_thread_num());    }
   // Ending of parallel region
} """ text_file =
open("code1.c", "w")
text_file.write(code1)
text_file.close() %env
OMP_NUM_THREADS=3
```

Name : Manthan Dhole

Id : 201080020 Subject : Parallel Computing

```
!gcc -o code1 -fopenmp code1.c
!./code1
```

ii. Producer Consumer problem,

```
code2 = """

//Code to implement and solve producer consumer problem
//Using OpenMP

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

#define BUFFER_SIZE 10

int buffer[BUFFER_SIZE];
int count = 0; int in =
0; int out = 0;

omp_lock_t lock;

void producer() {     int item;     while (1) {          item
= rand() % 100;          omp_set_lock(&lock);          if
(count < BUFFER_SIZE) {              buffer[in] = item;
in = (in + 1) % BUFFER_SIZE;              count++;
printf(" Producer produced item %d\\n", item);          }
omp_unset_lock(&lock);          #pragma omp flush
sleep(1);     } }

void consumer() {
int item;
while (1) {
omp_set_lock(&lo
ck);          if
(count > 0) {
item =
buffer[out];
out = (out + 1)
% BUFFER_SIZE;
```

Name : Manthan Dhole

Id : 201080020 Subject : Parallel Computing

```
count--;
printf("Consumer
consumed item
%d\\n", item);
}
omp_unset_lock(&
lock);
#pragma omp
flush
sleep(1);    } }

int main() {    omp_init_lock(&lock);
#pragma omp parallel num_threads(2)
{        int tid =
omp_get_thread_num();        if (tid
== 0) {            producer();
} else {             consumer();
}    }    omp_destroy_lock(&lock);
return 0; }

""" text_file = open("code2.c",
"w") text_file.write(code2)
text_file.close() %env
OMP_NUM_THREADS=3
!gcc -o code2 -fopenmp code2.c
!./code2
```

iii. Matrix Multiplication,

```
code3="""
#include <pthread.h>

#include <stdio.h>
#include <stdlib.h>
#include <omp.h>
#include <sys/time.h>


#define N 5
```

Name : Manthan Dhole

Id : 201080020 Subject : Parallel Computing

```
int A[N][N];
int B[N][N];
int C[N][N];

int main() {     int i,j,k;     struct timeval
tv1, tv2;     struct timezone tz;   double
elapsed;
omp_set_num_threads(omp_get_num_procs());
for (i= 0; i< N; i++)         for (j= 0; j<
N; j++)   {
           A[i][j] = 2;
           B[i][j] = 2;   }
printf("Matrix A:\\n");  for (i=
0; i< N; i++)    {         for (j=
0; j< N; j++)         {
printf("%d\\t",A[i][j]);        }
printf("\\n");    }
printf("\\nMatrix B:\\n");  for
(i= 0; i< N; i++)    {         for
(j= 0; j< N; j++)         {
printf("%d\\t",B[i][j]);        }
printf("\\n");    }
gettimeofday(&tv1, &tz);
#pragma omp parallel for
private(i,j,k) shared(A,B,C)
for (i = 0; i < N; ++i) {
for (j = 0; j < N; ++j) {
for (k = 0; k < N; ++k) {
C[i][j] += A[i][k] * B[k][j];
           }
       }
}


   gettimeofday(&tv2, &tz);    elapsed = (double)
(tv2.tv_sec-tv1.tv_sec) + (double) (tv2.tv_usectv1.tv_usec) * 1.e-6;
printf("\\nelapsed time = %f seconds.\\n", elapsed);
```

```
   printf("\\nThe product:\\n");
for (i= 0; i< N; i++)     {
for (j= 0; j< N; j++)          {
printf("%d\\t",C[i][j]);          }
printf("\\n");     }
} """ text_file =
open("code3.c", "w")
text_file.write(code3)
text_file.close() %env
OMP_NUM_THREADS=3
!gcc -o code3 -fopenmp code3.c
!./code3
```

iv. find prime number, `code4="""`

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

int is_prime(int n) {     if (n <= 1)
{         return 0;     }     for (int
i = 2; i * i <= n; i++) {         if
(n % i == 0) {             return 0;
}     }     return 1; }

int main() {     #pragma omp parallel
for     for (int i = 2; i <= 100; i++) {
if (is_prime(i)) {
printf("%d is prime\\n", i);          }
}     return 0; } """ text_file =
open("code4.c", "w")
text_file.write(code4)
text_file.close() %env
OMP_NUM_THREADS=3
!gcc -o code4 -fopenmp code4.c
!./code4
```

Name : Manthan Dhole

Id : 201080020 Subject : Parallel Computing

v. Largest Element in an array and

```
code5="""           #include
<stdio.h>
#include <stdlib.h>
#include <omp.h>

#define ARRAY_SIZE 10

int main() {     int i;
int largest = 0;     int
array[ARRAY_SIZE];

   // initialize the array with random values     for (i
= 0; i < ARRAY_SIZE; i++) {         array[i] = rand() %
10000;         printf("%d\\t",array[i]);     }
printf("\\n");     // find the largest element in the
array using OpenMP     #pragma omp parallel for
reduction(max:largest)     for (i = 0; i < ARRAY_SIZE;
i++) {         if (array[i] > largest) {
largest = array[i];         }     }

   printf("The largest element in the array is %d\\n", largest);

   return 0; } """ text_file =
open("code5.c", "w")
text_file.write(code5)
text_file.close() %env
OMP_NUM_THREADS=3
!gcc -o code5 -fopenmp code5.c
!./code5
```

vi. Pi calculation

```
code6=""" #include
<stdio.h>
#include <stdlib.h>
#include <time.h>
#include <omp.h>
```

Name : Manthan Dhole

Id : 201080020 Subject : Parallel Computing

```
int main() {     int n = 10000;   // number
of iterations     double x, y, pi;     int
count = 0;

   // set random seed
srand(time(NULL));

   #pragma omp parallel for private(x, y) reduction(+:count)
for (int i = 0; i < n; i++) {         // generate random
point (x, y) in [0, 1] x [0, 1]         x = (double)rand() /
RAND_MAX;         y = (double)rand() / RAND_MAX;

       // test if point is inside unit circle
if (x * x + y * y <= 1) {
count++;         }
   }

   // calculate pi    pi
= 4.0 * count / n;

   printf("pi = %f\\n", pi);

   return 0;
}

""" text_file = open("code6.c",
"w") text_file.write(code6)
text_file.close() %env
OMP_NUM_THREADS=3
!gcc -o code6 -fopenmp code6.c
!./code6
```

OUTPUT:

i. Fork Join model,

```
env: OMP_NUM_THREADS=3
Hello World... from thread = 0
Hello World... from thread = 2
Hello World... from thread = 1
```

Name : Manthan Dhole

Id : 201080020 Subject : Parallel Computing

ii. Producer Consumer problem,

```
code2.c: In function ‘producer’:
code2.c:32:9: warning: implicit declaration of function ‘sleep’ [-Wimplicit-function-declaration]
   32 |         sleep(1);
      |         ^~~~~
 Producer produced item 83
Consumer consumed item 83
 Producer produced item 86
Consumer consumed item 86
 Producer produced item 77
Consumer consumed item 77
 Producer produced item 15
 Producer produced item 93
Consumer consumed item 15
 Producer produced item 35
Consumer consumed item 93
Consumer consumed item 35
 Producer produced item 86
Consumer consumed item 86
 Producer produced item 92
Consumer consumed item 92
 Producer produced item 49
Consumer consumed item 49
 Producer produced item 21
Consumer consumed item 21
 Producer produced item 62
Consumer consumed item 62
 Producer produced item 27
Consumer consumed item 27
 Producer produced item 90
^C
```

iii. Matrix Multiplication,

```
Matrix A:
2       2       2       2       2
2       2       2       2       2
2       2       2       2       2
2       2       2       2       2
2       2       2       2       2

Matrix B:
2       2       2       2       2
2       2       2       2       2
2       2       2       2       2
2       2       2       2       2
2       2       2       2       2

elapsed time = 0.000049 seconds.

The product:
20      20      20      20      20
20      20      20      20      20
20      20      20      20      20
20      20      20      20      20
20      20      20      20      20
```

iv. find prime number,

Name : Manthan Dhole

Id : 201080020 Subject : Parallel Computing

```
2 is prime
3 is prime
5 is prime
7 is prime
11 is prime
13 is prime
17 is prime
19 is prime
23 is prime
29 is prime
31 is prime
71 is prime
73 is prime
79 is prime
83 is prime
89 is prime
97 is prime
37 is prime
41 is prime
43 is prime
47 is prime
53 is prime
59 is prime
61 is prime
67 is prime
```

v. Largest Element in an array and

```
9383    886    2777    6915    7793    8335    5386    492    6649    1421
The largest element in the array is 9383
```

vi. Pi calculation

```
pi = 3.134400
```

CONCLUSION:

In this lab we learned about openMP, and used many openMP inbuilt functions in order to include parallel processing into the program we are writing. We learned the format in which the code is written so that we can fork it and run it in parallel. We also learned how to declare the number of threads and the fork join model. We also revised some other concepts like consumer producer problem.