Parallel and Distributed Computing
[CSE4001]

Lab Digital Assignment

Prof- Manoov R.

NAME: Yashraj Agarwal

REG NO: 18BCI0183

Q1)Execute an OpenMP program to Counts the number of primes between 1 and N.

Solution:

Code in C:

```
# include <stdlib.h>
# include <stdio.h> #
include <omp.h>
 int main ( int argc, char *argv[] ); void
prime_number_sweep ( int n_lo, int n_hi, int
n_factor ); int prime_default ( int n );
int prime_static ( int n ); int
prime_dynamic ( int n ); int main (
int argc, char *argv[] )
{   int
n_factor;   int
n_hi;   int
n_lo;
    printf("enter starting number\n");
scanf("%d",&n_lo);   printf("enter the
end number(N)\n");
scanf("%d",&n_hi);   n_factor = 2;
   prime_number_sweep ( n_lo, n_hi, n_factor
);
   printf ( "\n" );   printf ( "  Normal end
of execution.\n" );
```

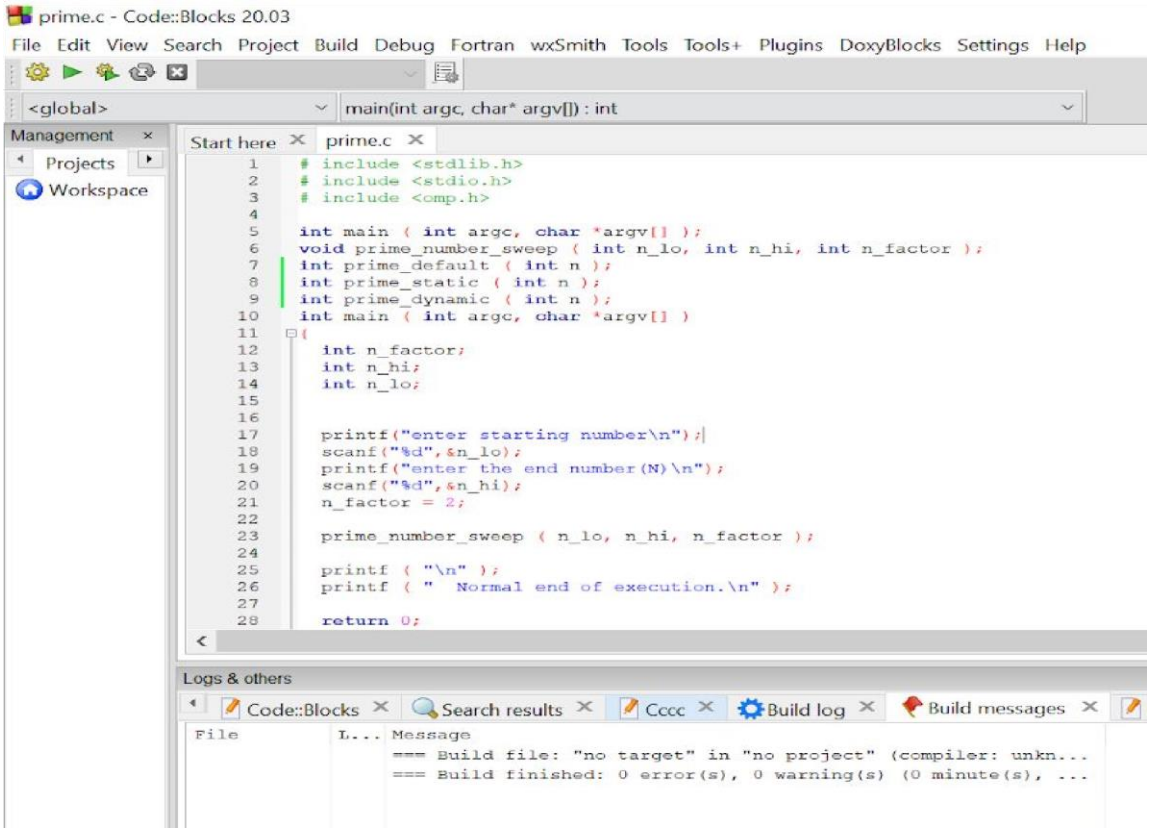
```
   return
0;
}  void prime_number_sweep ( int n_lo, int n_hi,
int
n_factor )
{   int n;   int primes;
double time1,time2,time3;
   printf ( "\n" );   printf ( "  Call PRIME_NUMBER
to count the primes from 1 to N.\n" );   printf (
"\n" );
  printf ( "         N        Pi(N)       Time1
Time2        Time3\n" );
printf ( "\n" );
   n =
n_lo;
   while ( n <= n_hi
)
  {
    time1= omp_get_wtime ( );
primes = prime_default( n );     time1
= omp_get_wtime ( ) - time1;
     time2= omp_get_wtime ( );
primes = prime_static( n );     time2
= omp_get_wtime ( ) - time2;
time3= omp_get_wtime ( );     primes =
```

```
prime_dynamic( n );     time3 =
omp_get_wtime ( ) - time3;
     printf ( "  %8d  %8d  %14f %14f %14f\n", n, primes,
time1, time2, time3 );
     n = n *
n_factor;
  }      printf ( "\n" );   printf ( "  Number of
processors available = %d\n", omp_get_num_procs ( )
);   printf ("  No of threads in
use=%d\n",omp_get_num_threads());  printf ( "
Optimal number of threads which gives minimal
execution time= %d\n", omp_get_max_threads ( )
);
return;
}
  int prime_default( int n
)
{   int
i;   int
j;   int
prime;
int
total =
0;
```

```
# pragma omp parallel \
shared ( n ) \   private
( i, j, prime )


# pragma omp for reduction ( + : total )
for ( i = 2; i <= n; i++ )
  {     prime
= 1;
     for ( j = 2; j < i; j++
)
    {       if ( i % j
== 0 )
      {
prime = 0;
break;
      }     }     total =
total + prime;
  }

  return total;
}  int prime_static ( int
n )
```

```
{   int i;   int
j;   int prime;
int total = 0;

# pragma omp parallel \
shared ( n ) \   private
( i, j, prime )



# pragma omp for reduction ( + : total )
schedule(static,100)   for ( i = 2; i <=
n; i++ )
  {     prime
= 1;
     for ( j = 2; j < i; j++
)
    {       if ( i % j
== 0 )
      {
        prime = 0;
break;
      }     }     total =
total + prime;
  }    return
total;
```

```
}  int prime_dynamic ( int
n )
{   int i;   int
j;   int prime;
int total = 0;

# pragma omp parallel \
shared ( n ) \   private
( i, j, prime )


# pragma omp for reduction ( + : total )
schedule(dynamic,100)   for ( i = 2; i
<= n; i++ )
  {     prime = 1;      for (
j = 2; j < i; j++ )
    {       if ( i % j ==
0 )       {         prime
= 0;          break;
}     }     total = total
+ prime;
  }    return
total;
}
```




```
29 }
30
31 void prime_number_sweep ( int n_lo, int n_hi, int n_factor )
32 {
33   int n;
34   int primes;
35   double time1,time2,time3;
36
37   printf ( "\n" );
38   printf ( "  Call PRIME_NUMBER to count the primes from 1 to N.\n" );
39   printf ( "\n" );
40   printf ( "         N        Pi(N)        Time1        Time2        Time3\n" );
41   printf ( "\n" );
42
43   n = n_lo;
44
45   while ( n <= n_hi )
46   {
47     time1= omp_get_wtime ( );
48     primes = prime_default( n );
49     time1 = omp_get_wtime ( ) - time1;
50
51     time2= omp_get_wtime ( );
52     primes = prime_static( n );
53     time2 = omp_get_wtime ( ) - time2;
54
55     time3= omp_get_wtime ( );
```

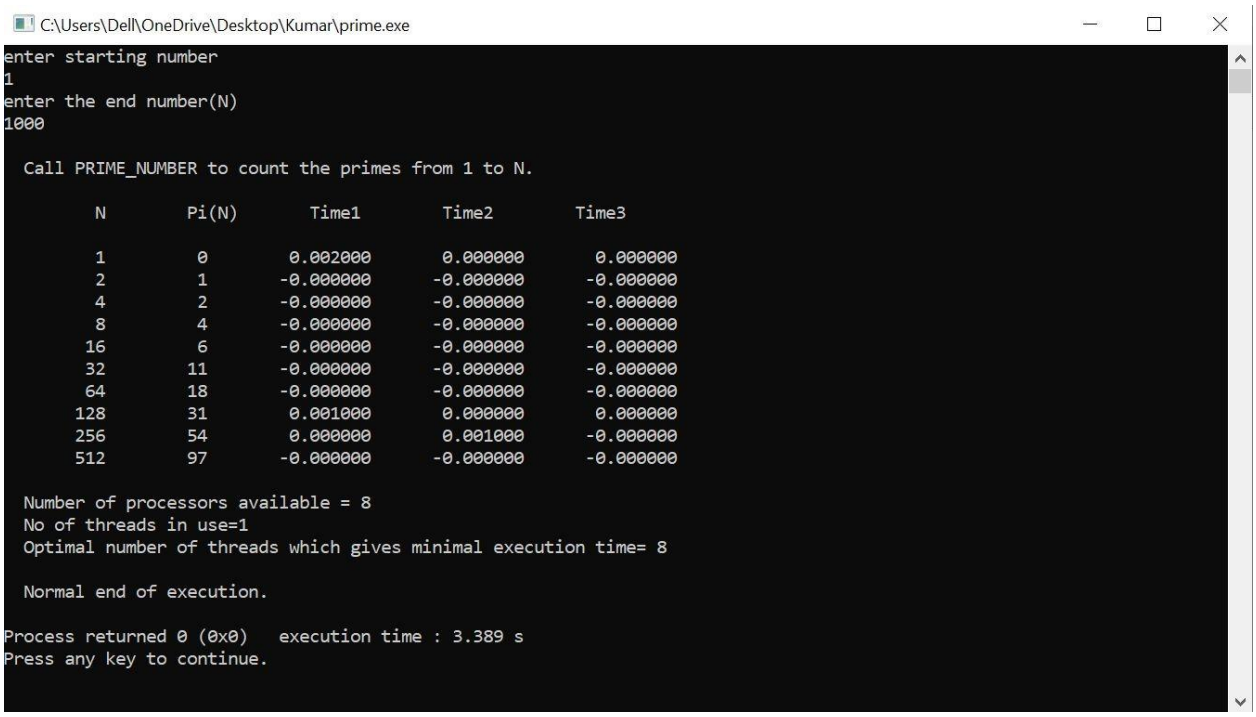
```
    time3= omp_get_wtime ( );
    primes = prime_dynamic( n );
    time3 = omp_get_wtime ( ) - time3;

    printf ( "  %8d  %8d  %14f %14f %14f\n", n, primes, time1, time2, time3 );

    n = n * n_factor;
  }

    printf ( "\n" );
  printf ( "  Number of processors available = %d\n", omp_get_num_procs ( ) );
  printf ("  No of threads in use=%d\n",omp_get_num_threads());
 printf ( "  Optimal number of threads which gives minimal execution time= %d\n", omp_get_max_threads ( ) );
  return;
}


int prime_default( int n )
{
  int i;
  int j;
  int prime;
  int total = 0;

# pragma omp parallel \
  shared ( n ) \
  private ( i, j, prime )


# pragma omp for reduction ( + : total )
  for ( i = 2; i <= n; i++ )
  {
    prime = 1;

    for ( j = 2; j < i; j++ )
    {
```

```
      if ( i % j == 0 )
      {
        prime = 0;
        break;
      }
    }
    total = total + prime;
  }

  return total;
}

int prime_static ( int n )
{
  int i;
  int j;
  int prime;
  int total = 0;

# pragma omp parallel \
  shared ( n ) \
  private ( i, j, prime )


# pragma omp for reduction ( + : total ) schedule(static,100)
  for ( i = 2; i <= n; i++ )
  {
    prime = 1;

    for ( j = 2; j < i; j++ )
    {
      if ( i % j == 0 )
      {
        prime = 0;
        break;
      }
    }
```

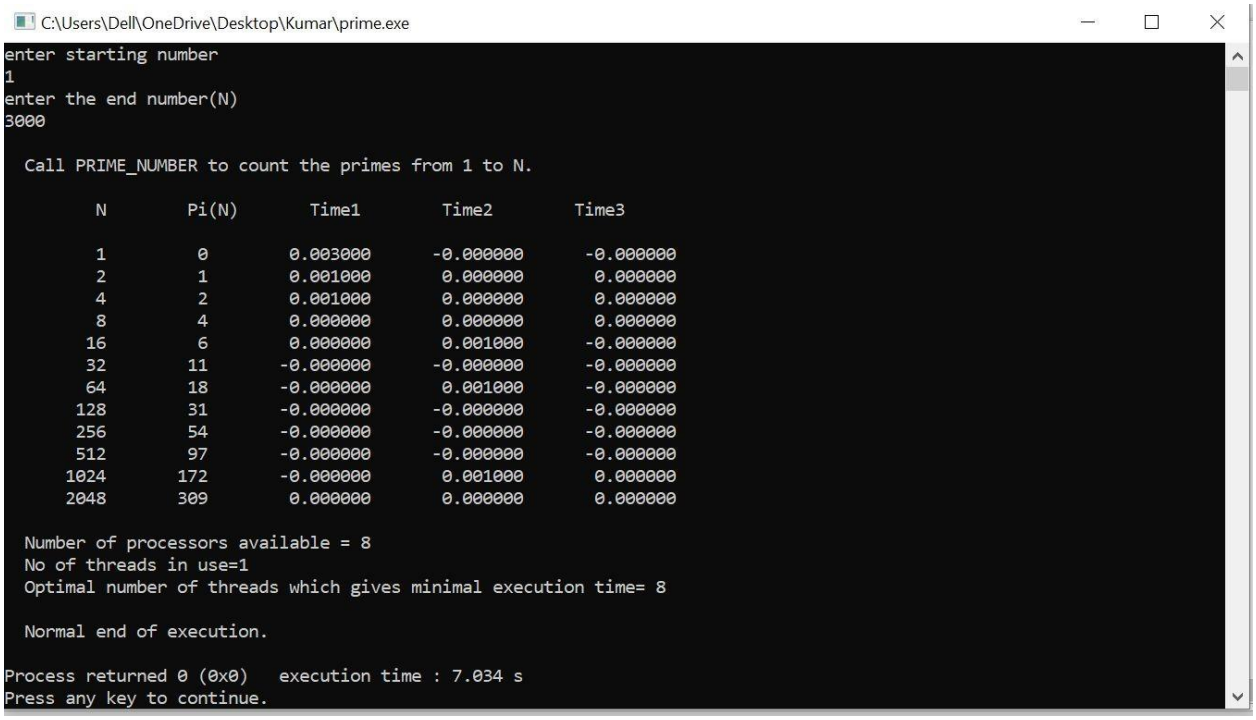
```
128       total = total + prime;
129     }
130
131     return total;
132   }
133
134   int prime_dynamic ( int n )
135   {
136     int i;
137     int j;
138     int prime;
139     int total = 0;
140
141   # pragma omp parallel \
142     shared ( n ) \
143     private ( i, j, prime )
144
145
146   # pragma omp for reduction ( + : total ) schedule(dynamic,100)
147     for ( i = 2; i <= n; i++ )
148     {
149       prime = 1;
150
151       for ( j = 2; j < i; j++ )
152       {
153         if ( i % j == 0 )
154         {
155           prime = 0;
156           break;
157         }
158       }
159       total = total + prime;
160     }
161
162     return total;
163   }
```

OUTPUTS:

When the N is set to 1000:


C:\Users\Dell\OneDrive\Desktop\Kumar\prime.exe
enter starting number
1
enter the end number(N)
1000
Call PRIME_NUMBER to count the primes from 1 to N.
N Pi(N) Time1 Time2 Time3
1 0 0.002000 0.000000 0.000000
2 1 -0.000000 -0.000000 -0.000000
4 2 -0.000000 -0.000000 -0.000000
8 4 -0.000000 -0.000000 -0.000000
16 6 -0.000000 -0.000000 -0.000000
32 11 -0.000000 -0.000000 -0.000000
64 18 -0.000000 -0.000000 -0.000000
128 31 0.001000 0.000000 0.000000
256 54 0.000000 0.001000 -0.000000
512 97 -0.000000 -0.000000 -0.000000
Number of processors available = 8
No of threads in use=1
Optimal number of threads which gives minimal execution time= 8
Normal end of execution.
Process returned 0 (0x0) execution time : 3.389 s
Press any key to continue.

When the N is set to 3000:



Conclusion: -

We can clearly see that as and when the value of N increases, the time taken to run the program increases. Thus the program runs faster when run parallely compared to the serial code. And the **open mp** code is **faster** than when written without it.