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) \setminus 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 Time3n");
printf ( "\n" );
 n =
n lo;
 while ( n <= n hi
)
 {
   time1= omp get wtime ( );
= omp get wtime ( ) - time1;
   time2= omp_get_wtime ( );
= omp get wtime ( ) - time2;
```

```
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 \le n; i++ )
{ prime
= 1;
   for (j = 2; j < i; j++
)
{ if (i % j
== 0 )
  {
prime = 0;
break;
 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 \le 
n; i++ )
{ prime
= 1;
 for (j = 2; j < i; j++
)
 { if (i%j
== 0 )
     prime = 0;
break;
  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;
}
```

```
prime.c - Code::Blocks 20.03
File Edit View Search Project Build Debug Fortran wxSmith Tools Tools+ Plugins DoxyBlocks Settings Help
☆ ▶ % ② ⊠
                        ~ <u>B</u>
 <global>

→ main(int argc, char* argv[]): int

Management ×
                  Start here × prime.c ×
 ⁴ Projects ▶
                             # include <stdlib.h>
# include <stdio.h>
# include <omp.h>

    ₩orkspace

                             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[] )
                       10
                       11
12
                               int n_factor;
                       13
14
15
                               int n_hi;
int n_lo;
                       16
                               printf("enter starting number\n");
                              printi("enter starting number\n");|
scanf("%d", %n_lo);
printf("enter the end number(N)\n");
scanf("%d", %n_hi);
n_factor = 2;
                       18
19
                       20
                       22
                       23
                              prime_number_sweep ( n_lo, n_hi, n_factor );
                               printf ( "\n" );
printf ( " Normal end of execution.\n" );
                       25
                       28
                               return 0:
                   <
                  Logs & others
                  L... Message
                                       === Build file: "no target" in "no project" (compiler: unkn...
                                       === Build finished: 0 error(s), 0 warning(s) (0 minute(s), ...
     29
     30
     31
             void prime_number_sweep ( int n_lo, int n_hi, int n_factor )
     32
          P(
     33
               int n;
     34
               int primes;
     35
               double time1, time2, time3;
     36
     37
               printf ( "\n" );
               printf ( " Call PRIME_NUMBER to count the primes from 1 to N.\n" );
     38
               printf ( "\n" );
     39
               printf ( "
     40
                                                    Pi(N)
                                                                                                      Time3\n");
                                        N
                                                                     Time1
                                                                                      Time2
               printf ( "\n" );
     41
     42
     43
               n = n_{lo};
     44
     45
               while ( n <= n hi )
     46
     47
                  timel= omp_get_wtime ( );
     48
                  primes = prime_default( n );
     49
                  time1 = omp_get_wtime ( ) - time1;
     50
                 time2= omp_get_wtime ( );
primes = prime_static( n );
time2 = omp_get_wtime ( ) - time2;
     51
     52
     53
     54
```

time3= omp_get_wtime ();

55

```
time3= omp_get_wtime ( );
56
        primes = prime dynamic( n );
57
        time3 = omp_get_wtime ( ) - time3;
58
59
        printf ( " %8d %8d %14f %14f %14f\n", n, primes, time1, time2, time3 );
60
       n = n * n_factor;
61
62
63
       printf ( "\n" );
64
      printf ( " Number of processors available = %d\n", omp_get_num_procs ( ) );
65
     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 ());
66
67
68
      return;
69
70
71
72 | int prime_default( int n )
73
74
      int i:
75
      int j;
76
      int prime;
77
      int total = 0;
78
79
    # pragma omp parallel \
80
     shared ( n ) \
81
     private ( i, j, prime )
82
83
84 | # pragma omp for reduction ( + : total )
85
      for ( i = 2; i <= n; i++ )
86
87
        prime = 1;
88
        for (j = 2; j < i; j++)
89
90
                  if ( i % j == 0 )
  91
  92
  93
                     prime = 0;
  94
                     break:
  95
  96
  97
               total = total + prime;
  98
  99
100
            return total;
 101
 102
         int prime_static ( int n )
 103
104
105
            int i;
 106
             int j;
107
             int prime;
108
            int total = 0;
109
110
          # pragma omp parallel \
            shared (n) \
 111
            private ( i, j, prime )
112
113
 114
          # pragma omp for reduction ( + : total ) schedule(static,100) for ( i = 2; i <= n; i++ )
115
116
117
118
               prime = 1;
 119
               for (j = 2; j < i; j++)
120
121
                  if ( i % j == 0 )
 122
 123
 124
                     prime = 0;
125
                     break;
 126
 127
```

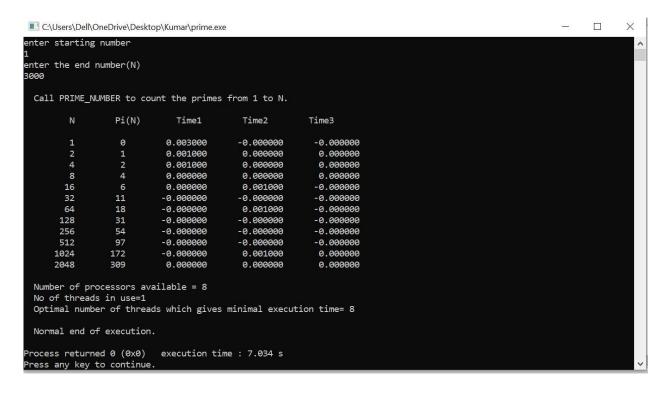
```
128
         total = total + prime;
 129
 130
 131
        return total;
 132
 133
 134
      int prime_dynamic ( int n )
     ₽(
 135
 136
         int i;
 137
         int j;
 138
         int prime;
         int total = 0;
 139
 140
 141
       # pragma omp parallel \
 142
         shared (n) \
        private ( i, j, prime )
 143
 144
 145
      # pragma omp for reduction ( + : total ) schedule(dynamic, 100)
 146
 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:

```
X
 C:\Users\Dell\OneDrive\Desktop\Kumar\prime.exe
enter starting number
enter the end number(N)
1000
 Call PRIME_NUMBER to count the primes from 1 to N.
                   Pi(N)
                                 Time1
                                                Time2
                                                               Time3
                     0
                               0.002000
                                                0.000000
                                                                 0.000000
                              -0.000000
                                               -0.000000
                                                                -0.000000
                                               -0.000000
                                                                -0.000000
                              -0.000000
                              -0.000000
                                               -0.000000
                                                                -0.000000
        16
                              -0.000000
                                               -0.000000
                                                                -0.000000
        32
                              -0.000000
                                                                -0.000000
                    11
                                               -0.000000
                              -0.000000
        64
                                               -0.000000
                                                                -0.000000
                    18
                                                0.000000
       128
                               0.001000
                                                                0.000000
       256
                    54
                               0.000000
                                                0.001000
                                                                -0.000000
       512
                              -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.