Name: Gavali Deshabhakt Nagnath **Introduction To Scalabel Systems**
Roll No. 202CD005

OpenMP Assignment

Index

Sr. No.	Question	Page No.
1	Simple parallel program to display the thread_id and total number of threads	2
2	Demonstration of if clause in parallel directive	3
3	Demonstrate and analyze shared clause in parallel directive	4
4	Demonstration of private() and firstprivate()	5
5	Demonstration of reduction clause in parallel directive	6
6	Demonstration of threadprivate directive, copyin clause and synchronization directives	6
7	Demonstration of pragma critical directive	8
8	Demonstration of master pragma	9
9	Illustration of OMP_FOR • Program • Output a) schedule (static, chunksize) b) schedule (dynamice, chunksize) c) schedule (guided, chunksize)	9 10 14 19

Q1.Write a C/C++ simple parallel program to display the thread_id and total number of threads. Use these various methods to set number of threads and mention the method of setting the same.

(a) with num-threads(4):
Program:

```
#include<stdio.h>
#include<omp.h>
int main()
{
   int nthreads,tid;
   #pragma omp parallel private(tid) num_threads(4)
   {
      tid=omp_get_thread_num();
      printf("Hello world from thread=%d\n",tid);
      if(tid==0)
      {
         nthreads=omp_get_num_threads();
         printf("Number of threads=%d\n",nthreads);
      }
   }
}
```

Output:

```
Hello world from thread=1
Hello world from thread=0
Number of threads=4
Hello world from thread=3
Hello world from thread=2
```

(b) with omp_set_num_threads()
Program:

```
#include<stdio.h>
#include<omp.h>
int main()
{
   int nthreads,tid;
   omp_set_num_threads(4);
   #pragma omp parallel private(tid)
   {
      tid=omp_get_thread_num();
      printf("Hello world from thread=%d\n",tid);
      if(tid==0)
      {
         nthreads=omp_get_num_threads();
         printf("Number of threads=%d\n",nthreads);
      }
   }
}
```

Output:

```
Hello world from thread=1
Hello world from thread=2
Hello world from thread=0
Number of threads=4
Hello world from thread=3
```

(c) with OMP_NUM_THREADS()

Program:

```
#include<stdio.h>
#include<omp.h>
int main()
{
   int nthreads,tid;
   #pragma omp parallel private(tid)
   {
      tid=omp_get_thread_num();
      printf("Hello world from thread=%d\n",tid);
      if(tid==0)
      {
         nthreads=omp_get_num_threads();
         printf("Number of threads=%d\n",nthreads);
      }
   }
}
```

setting number of threads using - “$export OMP_NUM_THREADS=4” command

Output:

```
Hello world from thread=2
Hello world from thread=0
Number of threads=4
Hello world from thread=1
Hello world from thread=3
```

Q2.Write a program demonstrating the working of if clause in parallel directive.

Program:

```
#include<stdio.h>
#include<omp.h>
int main()
{
   int val;
   printf("Enter 0: for serial 1: for parallel\n>");
   scanf("%d",&val);
   #pragma omp parallel if(val)
   {
      if(omp_in_parallel())
      printf("Parallel val=%d id= %d\n",val, omp_get_thread_num());
```

```
        else
        printf("Serial val=%d id= %d\n",val, omp_get_thread_num());
    }
}
```

Output:

```
[deshabhakt@deshabhakt-pc OpeMP-Assignment-1]$ ./a.out
Enter 0: for serial 1: for parallel
>0
Serial val=0 id= 0
[deshabhakt@deshabhakt-pc OpeMP-Assignment-1]$ ./a.out
Enter 0: for serial 1: for parallel
>1
Parallel val=1 id= 0
Parallel val=1 id= 3
Parallel val=1 id= 2
Parallel val=1 id= 1
```

Q3. Write a program to demonstrate and analyze shared clause in parallel directive.

Program:

```
#include <omp.h>
#include<stdio.h>
#include<stdlib.h>
int main()
{
    int x = 0;
    #pragma omp parallel num_threads(8) shared(x)   //x shared variable --> possibility of race condition
    {
        int tid = omp_get_thread_num();
        x = x + 1;                      //Each thread will add 1 to x
        printf("Thread [%d] value of x is %d \n", tid, x);
    }
}
```

Output:

```
Thread [2] value of x is 2
Thread [1] value of x is 1
Thread [3] value of x is 3
Thread [4] value of x is 4
Thread [6] value of x is 5
Thread [5] value of x is 6
Thread [0] value of x is 8
Thread [7] value of x is 7
```

Analysis:

As -x here is a shared variable, hence each thread tries to access it and therefore, there is a possibility of race condition.

Q4. Write a program demonstrating private() and firstprivate().

Program:

```
#include<stdio.h>
#include<omp.h>
int main()
{
   int i=10;
   printf("Value before pragma i=%d\n",i);
   #pragma omp parallel num_threads(4) firstprivate(i)
   {
      printf("Value after entering pragma i=%d tid=%d\n",i, omp_get_thread_num());
      i=i+omp_get_thread_num();   //adds thread_id to i
      printf("Value after changing value i=%d tid=%d\n",i, omp_get_thread_num());
   }
   printf("Value after having pragma i=%d tid=%d\n",i, omp_get_thread_num());
}

/* shared --> variable is shared to all threads --> all threads update same variable --> race condition possibility
  private --> Each thread creates a private copy of variable for itself and initialize it to "0"
  firstprivate --> Each thread creates a private copy of variable for itself and initialize to same value for which parent var is initialized
*/
```

(a) Output with firstprivate():

```
Value before pragma i=10
Value after entering pragma i=10 tid=0
Value after changing value i=10 tid=0
Value after entering pragma i=10 tid=1
Value after entering pragma i=10 tid=2
Value after changing value i=12 tid=2
Value after changing value i=11 tid=1
Value after entering pragma i=10 tid=3
Value after changing value i=13 tid=3
Value after having pragma i=10 tid=0
```

(b) Output with private():

```
Value before pragma i=10
Value after entering pragma i=0 tid=0
Value after changing value i=0 tid=0
Value after entering pragma i=0 tid=3
Value after changing value i=3 tid=3
Value after entering pragma i=0 tid=2
Value after changing value i=2 tid=2
Value after entering pragma i=0 tid=1
Value after changing value i=1 tid=1
Value after having pragma i=10 tid=0
```

Q5. Demonstration of reduction clause in parallel directive.

Program:

```
#include <stdio.h>
#include <omp.h>
void main()
{
   int x = 0;
   #pragma omp parallel num_threads(6) reduction(+:x)
   {
      int id = omp_get_thread_num();
      int threads = omp_get_num_threads();
      x = x + 1;
      printf("Hi from %d\n Value of x : %d\n", id, x);
   }
   printf("Final x:%d\n", x);
}
```

Output:

```
Hi from 1
 Value of x : 1
Hi from 2
 Value of x : 1
Hi from 3
 Value of x : 1
Hi from 4
 Value of x : 1
Hi from 5
 Value of x : 1
Hi from 0
 Value of x : 1
Final x:6
```

Q6. Execute following code and observe the working of threadprivate directive, copyin clause and synchronization directives

Program:

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

/*copyin(x) copies initialized value of x (from outside the pragma directive) in all threads  (inside the pragma directive)*/

/*initializing x globally works same
as (using copyin(x) clause and initializing x in main function)*/

int tid, x;            //here x is not initialized
```

```
#pragma omp threadprivate(x,tid)          // creates copy of variables for each thread
void main()
{
   x = 10;            // x initialized here
   #pragma omp parallel num_threads(4)   // to use copyin(x)--> x must be threadprivate
   {
      tid = omp_get_thread_num();
      #pragma omp master          // updates only master (i.e 0) thread
      {
         printf("Parallel Region 1 \n");
         x = x + 1;
      }
      #pragma omp barrier         //all threads wait here until remaining threads reach here
      if (tid == 1)
         x = x + 2;
      printf("Thread % d Value of x is %d\n", tid, x);
   }
   #pragma omp parallel num_threads(4)
   {
      #pragma omp master
      {
      printf("Parallel Region 2 \n");         //executed only once because of #pragma omp master
command
      }
      #pragma omp barrier
      printf("Thread %d Value of x is %d\n", tid, x);
   }
   printf("Value of x in Main Region is %d\n", x);
}
```

(a) Output with copyin():

```
Parallel Region 1
Thread  0 Value of x is 11
Thread  2 Value of x is 10
Thread  1 Value of x is 12
Thread  3 Value of x is 10
Parallel Region 2
Thread 1 Value of x is 12
Thread 0 Value of x is 11
Thread 3 Value of x is 10
Thread 2 Value of x is 10
Value of x in Main Region is 11
```

(b) Output without copyin():

```
Parallel Region 1
Thread  3 Value of x is 0
Thread  0 Value of x is 11
Thread  1 Value of x is 2
Thread  2 Value of x is 0
```

Parallel Region 2
Thread 0 Value of x is 11
Thread 3 Value of x is 0
Thread 2 Value of x is 0
Thread 1 Value of x is 2
Value of x in Main Region is 11

(c) Output without copyin() and declaring x globally:
Parallel Region 1
Thread 0 Value of x is 11
Thread 1 Value of x is 12
Thread 2 Value of x is 10
Thread 3 Value of x is 10
Parallel Region 2
Thread 2 Value of x is 10
Thread 3 Value of x is 10
Thread 0 Value of x is 11
Thread 1 Value of x is 12
Value of x in Main Region is 11

Analysis:
1) copyin() copies initialized value of x (from outside the pragma directive) in all threads (inside the pragma directive). i.e. copyin() copies value from master to all slave threads.
2) copyin() works same as declaring variable globally.
3) For copyin() to work properly, the variable passed to copyin() function must be declared threadprivate().

Q7. Demonstration of pragma critical directive
Program:

```
#include <stdlib.h>
#include <stdio.h>
#include <omp.h>
int main()
{
   int a[100][100], mx = -1, lmx = -1, i, j;
   for (j = 0; j < 100; j++)
      for (i = 0; i < 100; i++)
         a[i][j] = 1 + (int)(10.0 * rand() / (RAND_MAX + 1.0));
   #pragma omp parallel private(i) firstprivate(lmx) num_threads(1000)
   {
      #pragma omp for
      for (j = 0; j < 100; j++)
         for (i = 0; i < 100; i++)
            lmx = (lmx > a[i][j]) ? lmx : a[i][j];
   #pragma omp critical
      mx = (mx > lmx) ? mx : lmx;
   }
   printf("max value of a is %d\n", mx);
}
```

(a) Output with pragma critical directive:
max value of a is 10
(b) Output without pragma critical directive:
max value of a is 10

Q8. Demonstration of master pragma
Program:

```
#include <stdio.h>
#include <omp.h>
#include <stdlib.h>
#define N 4
int main()
{
   int a[N],i;
   #pragma omp parallel num_threads(N) shared(a) private(i)
   {
      i=omp_get_thread_num();
      a[i] = i;
      #pragma omp master
         printf("YOU SHOULD ONLY SEE THIS ONCE\n");
      printf("a[%d]=%d\n",i,a[i]);
   }
}
```

Output:

```
a[1]=1
a[2]=2
YOU SHOULD ONLY SEE THIS ONCE
a[0]=0
a[3]=3
```

Q9. Illustration of OMP_FOR

Program:

```
#include <omp.h>
#include <stdio.h>
#include <stdlib.h>
#define CHUNKSIZE 10
#define N 29
int main(int argc, char *argv[])
{
   int nthreads, tid, i, chunk;
   float a[N], b[N], c[N];
   for (i = 0; i < N; i++)
      a[i] = b[i] = i * 1.0;
   chunk = CHUNKSIZE;
   #pragma omp parallel shared(a, b, c, nthreads, chunk) private(i, tid)
   {
```

```
        tid = omp_get_thread_num();
        if (tid == 0)
        {
           nthreads = omp_get_num_threads();
           printf("Number of threads = %d\n", nthreads);
        }
        printf("Thread %d starting...\n", tid);
        #pragma omp for schedule(static, chunk)
        for (i = 0; i < N; i++)
        {
           c[i] = a[i] + b[i];
           printf("Thread %d:c[% d] = % f\n", tid, i, c[i]);
        }
    }
}
```

Output:

I) schedule(static,chunk)
a) Number of threads =5 and chunk size=10

```
Thread 1 starting...
Thread 1:c[ 10] =  20.000000
Thread 1:c[ 11] =  22.000000
Thread 1:c[ 12] =  24.000000
Thread 1:c[ 13] =  26.000000
Thread 1:c[ 14] =  28.000000
Thread 1:c[ 15] =  30.000000
Thread 1:c[ 16] =  32.000000
Thread 1:c[ 17] =  34.000000
Thread 1:c[ 18] =  36.000000
Thread 1:c[ 19] =  38.000000
Thread 1:c[ 60] =  120.000000
Thread 1:c[ 61] =  122.000000
Thread 1:c[ 62] =  124.000000
Thread 1:c[ 63] =  126.000000
Thread 1:c[ 64] =  128.000000
Thread 1:c[ 65] =  130.000000
Thread 1:c[ 66] =  132.000000
Thread 1:c[ 67] =  134.000000
Thread 1:c[ 68] =  136.000000
Thread 1:c[ 69] =  138.000000
Number of threads = 5
Thread 0 starting...
Thread 0:c[ 0] =  0.000000
Thread 0:c[ 1] =  2.000000
Thread 0:c[ 2] =  4.000000
Thread 0:c[ 3] =  6.000000
Thread 0:c[ 4] =  8.000000
Thread 0:c[ 5] =  10.000000
```

```
Thread 0:c[ 6] =  12.000000
Thread 0:c[ 7] =  14.000000
Thread 0:c[ 8] =  16.000000
Thread 0:c[ 9] =  18.000000
Thread 0:c[ 50] =  100.000000
Thread 0:c[ 51] =  102.000000
Thread 0:c[ 52] =  104.000000
Thread 0:c[ 53] =  106.000000
Thread 0:c[ 54] =  108.000000
Thread 0:c[ 55] =  110.000000
Thread 0:c[ 56] =  112.000000
Thread 0:c[ 57] =  114.000000
Thread 0:c[ 58] =  116.000000
Thread 0:c[ 59] =  118.000000
Thread 2 starting...
Thread 2:c[ 20] =  40.000000
Thread 2:c[ 21] =  42.000000
Thread 2:c[ 22] =  44.000000
Thread 2:c[ 23] =  46.000000
Thread 2:c[ 24] =  48.000000
Thread 2:c[ 25] =  50.000000
Thread 2:c[ 26] =  52.000000
Thread 2:c[ 27] =  54.000000
Thread 2:c[ 28] =  56.000000
Thread 2:c[ 29] =  58.000000
Thread 2:c[ 70] =  140.000000
Thread 2:c[ 71] =  142.000000
Thread 2:c[ 72] =  144.000000
Thread 2:c[ 73] =  146.000000
Thread 2:c[ 74] =  148.000000
Thread 2:c[ 75] =  150.000000
Thread 2:c[ 76] =  152.000000
Thread 2:c[ 77] =  154.000000
Thread 2:c[ 78] =  156.000000
Thread 2:c[ 79] =  158.000000
Thread 3 starting...
Thread 3:c[ 30] =  60.000000
Thread 3:c[ 31] =  62.000000
Thread 3:c[ 32] =  64.000000
Thread 3:c[ 33] =  66.000000
Thread 3:c[ 34] =  68.000000
Thread 3:c[ 35] =  70.000000
Thread 3:c[ 36] =  72.000000
Thread 3:c[ 37] =  74.000000
Thread 3:c[ 38] =  76.000000
Thread 3:c[ 39] =  78.000000
Thread 3:c[ 80] =  160.000000
Thread 3:c[ 81] =  162.000000
Thread 3:c[ 82] =  164.000000
```

Thread 3:c[83] = 166.000000
Thread 3:c[84] = 168.000000
Thread 3:c[85] = 170.000000
Thread 3:c[86] = 172.000000
Thread 3:c[87] = 174.000000
Thread 3:c[88] = 176.000000
Thread 3:c[89] = 178.000000
Thread 4 starting...
Thread 4:c[40] = 80.000000
Thread 4:c[41] = 82.000000
Thread 4:c[42] = 84.000000
Thread 4:c[43] = 86.000000
Thread 4:c[44] = 88.000000
Thread 4:c[45] = 90.000000
Thread 4:c[46] = 92.000000
Thread 4:c[47] = 94.000000
Thread 4:c[48] = 96.000000
Thread 4:c[49] = 98.000000
Thread 4:c[90] = 180.000000
Thread 4:c[91] = 182.000000
Thread 4:c[92] = 184.000000
Thread 4:c[93] = 186.000000
Thread 4:c[94] = 188.000000
Thread 4:c[95] = 190.000000
Thread 4:c[96] = 192.000000
Thread 4:c[97] = 194.000000
Thread 4:c[98] = 196.000000
Thread 4:c[99] = 198.000000

b) Number of threads =5 and chunk size =25

Thread 1 starting...
Thread 1:c[25] = 50.000000
Thread 1:c[26] = 52.000000
Thread 1:c[27] = 54.000000
Thread 1:c[28] = 56.000000
Thread 1:c[29] = 58.000000
Thread 1:c[30] = 60.000000
Thread 1:c[31] = 62.000000
Thread 1:c[32] = 64.000000
Thread 1:c[33] = 66.000000
Thread 1:c[34] = 68.000000
Thread 1:c[35] = 70.000000
Thread 1:c[36] = 72.000000
Thread 1:c[37] = 74.000000
Thread 1:c[38] = 76.000000
Thread 1:c[39] = 78.000000
Thread 1:c[40] = 80.000000
Thread 1:c[41] = 82.000000
Thread 1:c[42] = 84.000000

```
Thread 1:c[ 43] =  86.000000
Thread 1:c[ 44] =  88.000000
Thread 1:c[ 45] =  90.000000
Thread 1:c[ 46] =  92.000000
Thread 1:c[ 47] =  94.000000
Thread 1:c[ 48] =  96.000000
Thread 1:c[ 49] =  98.000000
Thread 3 starting...
Thread 3:c[ 75] =  150.000000
Thread 3:c[ 76] =  152.000000
Thread 3:c[ 77] =  154.000000
Thread 3:c[ 78] =  156.000000
Thread 3:c[ 79] =  158.000000
Thread 3:c[ 80] =  160.000000
Thread 3:c[ 81] =  162.000000
Thread 3:c[ 82] =  164.000000
Thread 3:c[ 83] =  166.000000
Thread 3:c[ 84] =  168.000000
Thread 3:c[ 85] =  170.000000
Thread 3:c[ 86] =  172.000000
Thread 3:c[ 87] =  174.000000
Thread 3:c[ 88] =  176.000000
Thread 3:c[ 89] =  178.000000
Thread 3:c[ 90] =  180.000000
Thread 3:c[ 91] =  182.000000
Thread 3:c[ 92] =  184.000000
Thread 3:c[ 93] =  186.000000
Thread 3:c[ 94] =  188.000000
Thread 3:c[ 95] =  190.000000
Thread 3:c[ 96] =  192.000000
Thread 3:c[ 97] =  194.000000
Thread 3:c[ 98] =  196.000000
Thread 3:c[ 99] =  198.000000
Thread 4 starting...
Number of threads = 5
Thread 0 starting...
Thread 0:c[ 0] =  0.000000
Thread 0:c[ 1] =  2.000000
Thread 0:c[ 2] =  4.000000
Thread 0:c[ 3] =  6.000000
Thread 0:c[ 4] =  8.000000
Thread 0:c[ 5] =  10.000000
Thread 0:c[ 6] =  12.000000
Thread 0:c[ 7] =  14.000000
Thread 0:c[ 8] =  16.000000
Thread 0:c[ 9] =  18.000000
Thread 0:c[ 10] =  20.000000
Thread 0:c[ 11] =  22.000000
Thread 0:c[ 12] =  24.000000
```

Thread 0:c[13] = 26.000000
Thread 0:c[14] = 28.000000
Thread 0:c[15] = 30.000000
Thread 0:c[16] = 32.000000
Thread 0:c[17] = 34.000000
Thread 0:c[18] = 36.000000
Thread 0:c[19] = 38.000000
Thread 0:c[20] = 40.000000
Thread 0:c[21] = 42.000000
Thread 0:c[22] = 44.000000
Thread 0:c[23] = 46.000000
Thread 0:c[24] = 48.000000
Thread 2 starting...
Thread 2:c[50] = 100.000000
Thread 2:c[51] = 102.000000
Thread 2:c[52] = 104.000000
Thread 2:c[53] = 106.000000
Thread 2:c[54] = 108.000000
Thread 2:c[55] = 110.000000
Thread 2:c[56] = 112.000000
Thread 2:c[57] = 114.000000
Thread 2:c[58] = 116.000000
Thread 2:c[59] = 118.000000
Thread 2:c[60] = 120.000000
Thread 2:c[61] = 122.000000
Thread 2:c[62] = 124.000000
Thread 2:c[63] = 126.000000
Thread 2:c[64] = 128.000000
Thread 2:c[65] = 130.000000
Thread 2:c[66] = 132.000000
Thread 2:c[67] = 134.000000
Thread 2:c[68] = 136.000000
Thread 2:c[69] = 138.000000
Thread 2:c[70] = 140.000000
Thread 2:c[71] = 142.000000
Thread 2:c[72] = 144.000000
Thread 2:c[73] = 146.000000
Thread 2:c[74] = 148.000000

II) schedule(dynamic,chunk)
a) Number of threads = 5 and chunk size =10
Thread 2 starting...
Thread 2:c[0] = 0.000000
Thread 2:c[1] = 2.000000
Thread 2:c[2] = 4.000000
Thread 1 starting...
Thread 3 starting...
Thread 3:c[10] = 20.000000
Thread 3:c[11] = 22.000000

```
Thread 3:c[ 12] =  24.000000
Thread 1:c[ 20] =  40.000000
Number of threads = 5
Thread 0 starting...
Thread 2:c[ 3] =  6.000000
Thread 2:c[ 4] =  8.000000
Thread 2:c[ 5] =  10.000000
Thread 2:c[ 6] =  12.000000
Thread 2:c[ 7] =  14.000000
Thread 2:c[ 8] =  16.000000
Thread 2:c[ 9] =  18.000000
Thread 2:c[ 40] =  80.000000
Thread 2:c[ 41] =  82.000000
Thread 2:c[ 42] =  84.000000
Thread 2:c[ 43] =  86.000000
Thread 2:c[ 44] =  88.000000
Thread 2:c[ 45] =  90.000000
Thread 2:c[ 46] =  92.000000
Thread 2:c[ 47] =  94.000000
Thread 2:c[ 48] =  96.000000
Thread 2:c[ 49] =  98.000000
Thread 2:c[ 50] =  100.000000
Thread 2:c[ 51] =  102.000000
Thread 2:c[ 52] =  104.000000
Thread 2:c[ 53] =  106.000000
Thread 2:c[ 54] =  108.000000
Thread 2:c[ 55] =  110.000000
Thread 2:c[ 56] =  112.000000
Thread 2:c[ 57] =  114.000000
Thread 2:c[ 58] =  116.000000
Thread 2:c[ 59] =  118.000000
Thread 2:c[ 60] =  120.000000
Thread 2:c[ 61] =  122.000000
Thread 2:c[ 62] =  124.000000
Thread 2:c[ 63] =  126.000000
Thread 2:c[ 64] =  128.000000
Thread 2:c[ 65] =  130.000000
Thread 2:c[ 66] =  132.000000
Thread 2:c[ 67] =  134.000000
Thread 2:c[ 68] =  136.000000
Thread 2:c[ 69] =  138.000000
Thread 4 starting...
Thread 1:c[ 21] =  42.000000
Thread 1:c[ 22] =  44.000000
Thread 2:c[ 70] =  140.000000
Thread 2:c[ 71] =  142.000000
Thread 2:c[ 72] =  144.000000
Thread 2:c[ 73] =  146.000000
Thread 2:c[ 74] =  148.000000
```

Thread 2:c[75] = 150.000000
Thread 2:c[76] = 152.000000
Thread 2:c[77] = 154.000000
Thread 2:c[78] = 156.000000
Thread 2:c[79] = 158.000000
Thread 2:c[90] = 180.000000
Thread 2:c[91] = 182.000000
Thread 2:c[92] = 184.000000
Thread 2:c[93] = 186.000000
Thread 2:c[94] = 188.000000
Thread 2:c[95] = 190.000000
Thread 2:c[96] = 192.000000
Thread 2:c[97] = 194.000000
Thread 2:c[98] = 196.000000
Thread 2:c[99] = 198.000000
Thread 3:c[13] = 26.000000
Thread 3:c[14] = 28.000000
Thread 3:c[15] = 30.000000
Thread 3:c[16] = 32.000000
Thread 3:c[17] = 34.000000
Thread 3:c[18] = 36.000000
Thread 3:c[19] = 38.000000
Thread 4:c[80] = 160.000000
Thread 4:c[81] = 162.000000
Thread 4:c[82] = 164.000000
Thread 4:c[83] = 166.000000
Thread 4:c[84] = 168.000000
Thread 4:c[85] = 170.000000
Thread 4:c[86] = 172.000000
Thread 4:c[87] = 174.000000
Thread 4:c[88] = 176.000000
Thread 4:c[89] = 178.000000
Thread 1:c[23] = 46.000000
Thread 1:c[24] = 48.000000
Thread 1:c[25] = 50.000000
Thread 1:c[26] = 52.000000
Thread 1:c[27] = 54.000000
Thread 1:c[28] = 56.000000
Thread 1:c[29] = 58.000000
Thread 0:c[30] = 60.000000
Thread 0:c[31] = 62.000000
Thread 0:c[32] = 64.000000
Thread 0:c[33] = 66.000000
Thread 0:c[34] = 68.000000
Thread 0:c[35] = 70.000000
Thread 0:c[36] = 72.000000
Thread 0:c[37] = 74.000000
Thread 0:c[38] = 76.000000
Thread 0:c[39] = 78.000000

b) Number of threads = 8 and chunk size = 10

```
Thread 1 starting...
Thread 1:c[ 0] =  0.000000
Thread 1:c[ 1] =  2.000000
Thread 1:c[ 2] =  4.000000
Thread 1:c[ 3] =  6.000000
Thread 1:c[ 4] =  8.000000
Thread 1:c[ 5] =  10.000000
Thread 1:c[ 6] =  12.000000
Thread 1:c[ 7] =  14.000000
Thread 1:c[ 8] =  16.000000
Thread 1:c[ 9] =  18.000000
Thread 1:c[ 10] =  20.000000
Thread 1:c[ 11] =  22.000000
Thread 1:c[ 12] =  24.000000
Thread 1:c[ 13] =  26.000000
Thread 1:c[ 14] =  28.000000
Thread 1:c[ 15] =  30.000000
Thread 1:c[ 16] =  32.000000
Thread 1:c[ 17] =  34.000000
Thread 1:c[ 18] =  36.000000
Thread 1:c[ 19] =  38.000000
Thread 1:c[ 20] =  40.000000
Thread 1:c[ 21] =  42.000000
Thread 1:c[ 22] =  44.000000
Thread 1:c[ 23] =  46.000000
Thread 1:c[ 24] =  48.000000
Thread 1:c[ 25] =  50.000000
Thread 1:c[ 26] =  52.000000
Thread 1:c[ 27] =  54.000000
Thread 1:c[ 28] =  56.000000
Thread 1:c[ 29] =  58.000000
Thread 1:c[ 30] =  60.000000
Thread 1:c[ 31] =  62.000000
Thread 1:c[ 32] =  64.000000
Thread 1:c[ 33] =  66.000000
Thread 1:c[ 34] =  68.000000
Thread 1:c[ 35] =  70.000000
Thread 1:c[ 36] =  72.000000
Thread 1:c[ 37] =  74.000000
Thread 1:c[ 38] =  76.000000
Thread 1:c[ 39] =  78.000000
Thread 1:c[ 40] =  80.000000
Thread 1:c[ 41] =  82.000000
Thread 1:c[ 42] =  84.000000
Thread 1:c[ 43] =  86.000000
Thread 1:c[ 44] =  88.000000
Thread 1:c[ 45] =  90.000000
Thread 1:c[ 46] =  92.000000
```

Thread 1:c[47] = 94.000000
Thread 1:c[48] = 96.000000
Thread 1:c[49] = 98.000000
Thread 1:c[50] = 100.000000
Thread 1:c[51] = 102.000000
Thread 1:c[52] = 104.000000
Thread 1:c[53] = 106.000000
Thread 1:c[54] = 108.000000
Thread 1:c[55] = 110.000000
Thread 1:c[56] = 112.000000
Thread 1:c[57] = 114.000000
Thread 1:c[58] = 116.000000
Thread 1:c[59] = 118.000000
Thread 1:c[60] = 120.000000
Thread 1:c[61] = 122.000000
Thread 1:c[62] = 124.000000
Thread 1:c[63] = 126.000000
Thread 1:c[64] = 128.000000
Thread 1:c[65] = 130.000000
Thread 1:c[66] = 132.000000
Thread 1:c[67] = 134.000000
Thread 1:c[68] = 136.000000
Thread 1:c[69] = 138.000000
Thread 1:c[70] = 140.000000
Thread 1:c[71] = 142.000000
Thread 1:c[72] = 144.000000
Thread 1:c[73] = 146.000000
Thread 1:c[74] = 148.000000
Thread 1:c[75] = 150.000000
Thread 1:c[76] = 152.000000
Thread 1:c[77] = 154.000000
Thread 1:c[78] = 156.000000
Thread 1:c[79] = 158.000000
Thread 1:c[80] = 160.000000
Thread 1:c[81] = 162.000000
Thread 1:c[82] = 164.000000
Thread 1:c[83] = 166.000000
Thread 1:c[84] = 168.000000
Thread 1:c[85] = 170.000000
Thread 1:c[86] = 172.000000
Thread 1:c[87] = 174.000000
Thread 1:c[88] = 176.000000
Thread 1:c[89] = 178.000000
Thread 1:c[90] = 180.000000
Thread 1:c[91] = 182.000000
Thread 1:c[92] = 184.000000
Thread 1:c[93] = 186.000000
Thread 1:c[94] = 188.000000
Thread 1:c[95] = 190.000000

Thread 1:c[96] = 192.000000
Thread 1:c[97] = 194.000000
Thread 1:c[98] = 196.000000
Thread 1:c[99] = 198.000000
Thread 2 starting...
Thread 5 starting...
Thread 6 starting...
Thread 7 starting...
Thread 4 starting...
Number of threads = 8
Thread 0 starting...
Thread 3 starting...

III) schedule(guided,chunk)
a) number of threads =5 and chunk size=10
Thread 2 starting...
Thread 2:c[0] = 0.000000
Thread 2:c[1] = 2.000000
Thread 2:c[2] = 4.000000
Thread 2:c[3] = 6.000000
Thread 2:c[4] = 8.000000
Thread 2:c[5] = 10.000000
Thread 2:c[6] = 12.000000
Thread 2:c[7] = 14.000000
Thread 2:c[8] = 16.000000
Thread 2:c[9] = 18.000000
Thread 2:c[10] = 20.000000
Thread 2:c[11] = 22.000000
Thread 2:c[12] = 24.000000
Thread 2:c[13] = 26.000000
Thread 2:c[14] = 28.000000
Thread 2:c[15] = 30.000000
Thread 2:c[16] = 32.000000
Thread 2:c[17] = 34.000000
Thread 2:c[18] = 36.000000
Thread 2:c[19] = 38.000000
Thread 2:c[20] = 40.000000
Thread 2:c[21] = 42.000000
Thread 2:c[22] = 44.000000
Thread 2:c[23] = 46.000000
Thread 2:c[24] = 48.000000
Thread 2:c[25] = 50.000000
Thread 2:c[26] = 52.000000
Thread 2:c[27] = 54.000000
Thread 2:c[28] = 56.000000
Thread 2:c[29] = 58.000000
Thread 2:c[30] = 60.000000
Thread 2:c[31] = 62.000000
Thread 2:c[32] = 64.000000

Thread 2:c[33] = 66.000000
Thread 2:c[34] = 68.000000
Thread 2:c[35] = 70.000000
Thread 2:c[36] = 72.000000
Thread 2:c[37] = 74.000000
Thread 2:c[38] = 76.000000
Thread 2:c[39] = 78.000000
Thread 2:c[40] = 80.000000
Thread 2:c[41] = 82.000000
Thread 2:c[42] = 84.000000
Thread 2:c[43] = 86.000000
Thread 2:c[44] = 88.000000
Thread 2:c[45] = 90.000000
Thread 2:c[46] = 92.000000
Thread 2:c[47] = 94.000000
Thread 2:c[48] = 96.000000
Thread 2:c[49] = 98.000000
Thread 2:c[50] = 100.000000
Thread 2:c[51] = 102.000000
Thread 2:c[52] = 104.000000
Thread 2:c[53] = 106.000000
Thread 2:c[54] = 108.000000
Thread 2:c[55] = 110.000000
Thread 2:c[56] = 112.000000
Thread 2:c[57] = 114.000000
Thread 2:c[58] = 116.000000
Thread 2:c[59] = 118.000000
Thread 2:c[60] = 120.000000
Thread 2:c[61] = 122.000000
Thread 2:c[62] = 124.000000
Thread 2:c[63] = 126.000000
Thread 2:c[64] = 128.000000
Thread 2:c[65] = 130.000000
Thread 2:c[66] = 132.000000
Thread 2:c[67] = 134.000000
Thread 2:c[68] = 136.000000
Thread 2:c[69] = 138.000000
Thread 2:c[70] = 140.000000
Thread 2:c[71] = 142.000000
Thread 2:c[72] = 144.000000
Thread 2:c[73] = 146.000000
Thread 2:c[74] = 148.000000
Thread 2:c[75] = 150.000000
Thread 2:c[76] = 152.000000
Thread 2:c[77] = 154.000000
Thread 2:c[78] = 156.000000
Thread 2:c[79] = 158.000000
Thread 2:c[80] = 160.000000
Thread 2:c[81] = 162.000000

Thread 2:c[82] = 164.000000
Thread 2:c[83] = 166.000000
Thread 2:c[84] = 168.000000
Thread 2:c[85] = 170.000000
Thread 2:c[86] = 172.000000
Thread 2:c[87] = 174.000000
Thread 2:c[88] = 176.000000
Thread 2:c[89] = 178.000000
Thread 2:c[90] = 180.000000
Thread 2:c[91] = 182.000000
Thread 2:c[92] = 184.000000
Thread 2:c[93] = 186.000000
Thread 2:c[94] = 188.000000
Thread 2:c[95] = 190.000000
Thread 2:c[96] = 192.000000
Thread 2:c[97] = 194.000000
Thread 2:c[98] = 196.000000
Thread 2:c[99] = 198.000000
Thread 1 starting...
Thread 3 starting...
Thread 4 starting...
Number of threads = 5
Thread 0 starting...

b) number of threads =5 and chunk size=5

Number of threads = 5
Thread 0 starting...
Thread 0:c[0] = 0.000000
Thread 0:c[1] = 2.000000
Thread 0:c[2] = 4.000000
Thread 0:c[3] = 6.000000
Thread 0:c[4] = 8.000000
Thread 0:c[5] = 10.000000
Thread 0:c[6] = 12.000000
Thread 0:c[7] = 14.000000
Thread 0:c[8] = 16.000000
Thread 0:c[9] = 18.000000
Thread 0:c[10] = 20.000000
Thread 0:c[11] = 22.000000
Thread 0:c[12] = 24.000000
Thread 0:c[13] = 26.000000
Thread 0:c[14] = 28.000000
Thread 0:c[15] = 30.000000
Thread 0:c[16] = 32.000000
Thread 0:c[17] = 34.000000
Thread 0:c[18] = 36.000000
Thread 0:c[19] = 38.000000
Thread 0:c[20] = 40.000000
Thread 0:c[21] = 42.000000

Thread 0:c[22] = 44.000000
Thread 0:c[23] = 46.000000
Thread 0:c[24] = 48.000000
Thread 0:c[25] = 50.000000
Thread 0:c[26] = 52.000000
Thread 0:c[27] = 54.000000
Thread 0:c[28] = 56.000000
Thread 0:c[29] = 58.000000
Thread 0:c[30] = 60.000000
Thread 0:c[31] = 62.000000
Thread 0:c[32] = 64.000000
Thread 0:c[33] = 66.000000
Thread 0:c[34] = 68.000000
Thread 0:c[35] = 70.000000
Thread 0:c[36] = 72.000000
Thread 0:c[37] = 74.000000
Thread 0:c[38] = 76.000000
Thread 0:c[39] = 78.000000
Thread 0:c[40] = 80.000000
Thread 0:c[41] = 82.000000
Thread 0:c[42] = 84.000000
Thread 0:c[43] = 86.000000
Thread 0:c[44] = 88.000000
Thread 0:c[45] = 90.000000
Thread 0:c[46] = 92.000000
Thread 0:c[47] = 94.000000
Thread 0:c[48] = 96.000000
Thread 0:c[49] = 98.000000
Thread 0:c[50] = 100.000000
Thread 0:c[51] = 102.000000
Thread 0:c[52] = 104.000000
Thread 0:c[53] = 106.000000
Thread 0:c[54] = 108.000000
Thread 0:c[55] = 110.000000
Thread 0:c[56] = 112.000000
Thread 0:c[57] = 114.000000
Thread 0:c[58] = 116.000000
Thread 0:c[59] = 118.000000
Thread 0:c[60] = 120.000000
Thread 0:c[61] = 122.000000
Thread 0:c[62] = 124.000000
Thread 0:c[63] = 126.000000
Thread 0:c[64] = 128.000000
Thread 0:c[65] = 130.000000
Thread 0:c[66] = 132.000000
Thread 0:c[67] = 134.000000
Thread 0:c[68] = 136.000000
Thread 0:c[69] = 138.000000
Thread 0:c[70] = 140.000000

Thread 0:c[71] = 142.000000
Thread 0:c[72] = 144.000000
Thread 0:c[73] = 146.000000
Thread 0:c[74] = 148.000000
Thread 0:c[75] = 150.000000
Thread 0:c[76] = 152.000000
Thread 0:c[77] = 154.000000
Thread 0:c[78] = 156.000000
Thread 0:c[79] = 158.000000
Thread 0:c[80] = 160.000000
Thread 0:c[81] = 162.000000
Thread 0:c[82] = 164.000000
Thread 0:c[83] = 166.000000
Thread 0:c[84] = 168.000000
Thread 0:c[85] = 170.000000
Thread 0:c[86] = 172.000000
Thread 0:c[87] = 174.000000
Thread 0:c[88] = 176.000000
Thread 0:c[89] = 178.000000
Thread 0:c[90] = 180.000000
Thread 0:c[91] = 182.000000
Thread 0:c[92] = 184.000000
Thread 0:c[93] = 186.000000
Thread 0:c[94] = 188.000000
Thread 0:c[95] = 190.000000
Thread 0:c[96] = 192.000000
Thread 0:c[97] = 194.000000
Thread 0:c[98] = 196.000000
Thread 0:c[99] = 198.000000
Thread 2 starting...
Thread 4 starting...
Thread 1 starting...
Thread 3 starting...

c) number of threads =8 and chunk size =5

Thread 1 starting...
Thread 1:c[0] = 0.000000
Thread 1:c[1] = 2.000000
Thread 1:c[2] = 4.000000
Thread 1:c[3] = 6.000000
Thread 1:c[4] = 8.000000
Thread 1:c[5] = 10.000000
Thread 7 starting...
Thread 7:c[13] = 26.000000
Thread 7:c[14] = 28.000000
Thread 7:c[15] = 30.000000
Thread 5 starting...
Thread 3 starting...
Thread 5:c[24] = 48.000000

```
Thread 2 starting...
Number of threads = 8
Thread 0 starting...
Thread 0:c[ 51] =  102.000000
Thread 0:c[ 52] =  104.000000
Thread 0:c[ 53] =  106.000000
Thread 0:c[ 54] =  108.000000
Thread 0:c[ 55] =  110.000000
Thread 0:c[ 56] =  112.000000
Thread 0:c[ 57] =  114.000000
Thread 0:c[ 58] =  116.000000
Thread 0:c[ 59] =  118.000000
Thread 0:c[ 60] =  120.000000
Thread 0:c[ 61] =  122.000000
Thread 0:c[ 62] =  124.000000
Thread 0:c[ 63] =  126.000000
Thread 0:c[ 64] =  128.000000
Thread 0:c[ 65] =  130.000000
Thread 0:c[ 66] =  132.000000
Thread 0:c[ 67] =  134.000000
Thread 0:c[ 68] =  136.000000
Thread 0:c[ 69] =  138.000000
Thread 0:c[ 70] =  140.000000
Thread 0:c[ 71] =  142.000000
Thread 0:c[ 72] =  144.000000
Thread 0:c[ 73] =  146.000000
Thread 0:c[ 74] =  148.000000
Thread 0:c[ 75] =  150.000000
Thread 0:c[ 76] =  152.000000
Thread 0:c[ 77] =  154.000000
Thread 0:c[ 78] =  156.000000
Thread 0:c[ 79] =  158.000000
Thread 0:c[ 80] =  160.000000
Thread 0:c[ 81] =  162.000000
Thread 0:c[ 82] =  164.000000
Thread 0:c[ 83] =  166.000000
Thread 0:c[ 84] =  168.000000
Thread 0:c[ 85] =  170.000000
Thread 0:c[ 86] =  172.000000
Thread 0:c[ 87] =  174.000000
Thread 0:c[ 88] =  176.000000
Thread 0:c[ 89] =  178.000000
Thread 0:c[ 90] =  180.000000
Thread 0:c[ 91] =  182.000000
Thread 0:c[ 92] =  184.000000
Thread 0:c[ 93] =  186.000000
Thread 0:c[ 94] =  188.000000
Thread 0:c[ 95] =  190.000000
Thread 0:c[ 96] =  192.000000
```

```
Thread 0:c[ 97] =  194.000000
Thread 0:c[ 98] =  196.000000
Thread 0:c[ 99] =  198.000000
Thread 3:c[ 34] =  68.000000
Thread 3:c[ 35] =  70.000000
Thread 3:c[ 36] =  72.000000
Thread 3:c[ 37] =  74.000000
Thread 3:c[ 38] =  76.000000
Thread 3:c[ 39] =  78.000000
Thread 3:c[ 40] =  80.000000
Thread 3:c[ 41] =  82.000000
Thread 3:c[ 42] =  84.000000
Thread 2:c[ 43] =  86.000000
Thread 2:c[ 44] =  88.000000
Thread 2:c[ 45] =  90.000000
Thread 2:c[ 46] =  92.000000
Thread 2:c[ 47] =  94.000000
Thread 7:c[ 16] =  32.000000
Thread 7:c[ 17] =  34.000000
Thread 7:c[ 18] =  36.000000
Thread 7:c[ 19] =  38.000000
Thread 7:c[ 20] =  40.000000
Thread 7:c[ 21] =  42.000000
Thread 7:c[ 22] =  44.000000
Thread 7:c[ 23] =  46.000000
Thread 2:c[ 48] =  96.000000
Thread 2:c[ 49] =  98.000000
Thread 2:c[ 50] =  100.000000
Thread 4 starting...
Thread 1:c[ 6] =  12.000000
Thread 1:c[ 7] =  14.000000
Thread 1:c[ 8] =  16.000000
Thread 1:c[ 9] =  18.000000
Thread 1:c[ 10] =  20.000000
Thread 1:c[ 11] =  22.000000
Thread 1:c[ 12] =  24.000000
Thread 5:c[ 25] =  50.000000
Thread 5:c[ 26] =  52.000000
Thread 5:c[ 27] =  54.000000
Thread 5:c[ 28] =  56.000000
Thread 5:c[ 29] =  58.000000
Thread 5:c[ 30] =  60.000000
Thread 5:c[ 31] =  62.000000
Thread 5:c[ 32] =  64.000000
Thread 5:c[ 33] =  66.000000
Thread 6 starting...
```

Analysis:

a) Static:

i) OpenMP divides the iterations into chunks of size chunk size and it distributes the chunks to threads in a circular order.

ii)When no chunk size is specified, OpenMP divides iterations into chunks that are approximately equal in size and it distributes at most one chunk to each thread.

b) Dynamic:

i) OpenMP divides the iterations into chunks of size chunk-size. Each thread executes a chunk of iterations and then requests another chunk until there are no more chunks available.

ii) There is no particular order in which the chunks are distributed to the threads. The order changes each time when we execute the for loop.

iii)If we do not specify chunk-size, it defaults to one.

c) Guided:

Almost similar to dynamic scheduling, only differes in chunk size. In Guided scheduling the chunk size is proportional to number of threads.

========== **THE END** ==========