

NATIONAL INSTITUTE OF TECHNOLOGY KARNATAKA SURATHKAL  
DEPARTMENT OF INFORMATION TECHNOLOGY  
IT 301 Parallel Computing LAB 2  
19th August 2020  
Faculty: Dr. Geetha V and Mrs. Tanmayee

---

**Name:** Chinmayi C. Ramakrishna

**Roll No.:** 181IT113

**Program 1:**

To understand and analyze shared clause in parallel directive.

```
chinmayi@chinmayi-HP-Pavilion-Laptop-15-cs3xxx: ~  
chinmayi@chinmayi-HP-Pavilion-Laptop-15-cs3xxx:~$ subl shared.c  
chinmayi@chinmayi-HP-Pavilion-Laptop-15-cs3xxx:~$ gcc -o shared -fopenmp shared.c  
chinmayi@chinmayi-HP-Pavilion-Laptop-15-cs3xxx:~$ ./shared  
  
Thread [7]  
value of x is 3  
  
Thread [6]  
value of x is 1  
  
Thread [2]  
value of x is 2  
  
Thread [5]  
value of x is 2  
  
Thread [4]  
value of x is 1  
  
Thread [1]  
value of x is 1  
  
Thread [3]  
value of x is 1  
  
Thread [0]  
value of x is 2  
chinmayi@chinmayi-HP-Pavilion-Laptop-15-cs3xxx:~$
```

Value 1 is shared by three threads 1,4 and 6.

Value 2 is shared by three threads 0,2 and 5.

Value 3 is shared by thread 7.

## 2. Program 2

Learn the concept of `private ()`, `firstprivate ()`

### `private()`

```
chinmayi@chinmayi-HP-Pavilion-Laptop-15-cs3xxx: ~  
chinmayi@chinmayi-HP-Pavilion-Laptop-15-cs3xxx:~$ subl learn.c  
chinmayi@chinmayi-HP-Pavilion-Laptop-15-cs3xxx:~$ gcc -o learn -fopenmp learn.c  
chinmayi@chinmayi-HP-Pavilion-Laptop-15-cs3xxx:~$ ./learn  
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  
chinmayi@chinmayi-HP-Pavilion-Laptop-15-cs3xxx:~$
```

Each thread 0, 1, 2 and 3 has its own instance of variable `i=0`

### `firstprivate()`

```
chinmayi@chinmayi-HP-Pavilion-Laptop-15-cs3xxx: ~  
chinmayi@chinmayi-HP-Pavilion-Laptop-15-cs3xxx:~$ gcc -o learn -fopenmp learn.c  
chinmayi@chinmayi-HP-Pavilion-Laptop-15-cs3xxx:~$ ./learn  
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=2  
Value after changing value i=12 tid=2  
Value after entering pragma i=10 tid=3  
Value after changing value i=13 tid=3  
Value after entering pragma i=10 tid=1  
Value after changing value i=11 tid=1  
Value after having pragma i=10 tid=0  
chinmayi@chinmayi-HP-Pavilion-Laptop-15-cs3xxx:~$
```

Every thread 0, 1, 2 and 3 has its own instance of the variable and the variable is initialized with the value of the variable. The threads 0, 1, 2 and 3 have `i=10`.

### 3. Program 3

Learn the working of lastprivate () clause:

```
chinmayi@chinmayi-HP-Pavilion-Laptop-15-cs3xxx: ~  
chinmayi@chinmayi-HP-Pavilion-Laptop-15-cs3xxx:~$ subl lastprivate.c  
chinmayi@chinmayi-HP-Pavilion-Laptop-15-cs3xxx:~$ gcc -o lastprivate -fopenmp lastprivate.c  
chinmayi@chinmayi-HP-Pavilion-Laptop-15-cs3xxx:~$ ./lastprivate  
Enter the value of n: 5  
Thread 0: value of i : 0  
Thread 0: x is 0  
Thread 4: value of i : 4  
Thread 4: x is 4  
Thread 2: value of i : 2  
Thread 2: x is 6  
Thread 3: value of i : 3  
Thread 3: x is 9  
Thread 1: value of i : 1  
Thread 1: x is 10  
x is: 10  
i is: 5  
chinmayi@chinmayi-HP-Pavilion-Laptop-15-cs3xxx:~$ ./lastprivate  
Enter the value of n: 8  
Thread 6: value of i : 6  
Thread 6: x is 6  
Thread 3: value of i : 3  
Thread 3: x is 9  
Thread 0: value of i : 0  
Thread 0: x is 9  
Thread 2: value of i : 2  
Thread 2: x is 11  
Thread 7: value of i : 7  
Thread 7: x is 18  
Thread 4: value of i : 4  
Thread 4: x is 22  
Thread 1: value of i : 1  
Thread 1: x is 23  
Thread 5: value of i : 5  
Thread 5: x is 28  
x is: 28  
i is: 8  
chinmayi@chinmayi-HP-Pavilion-Laptop-15-cs3xxx:~$
```

The value of x at the next iteration is i of the current iteration+ value of x at the previous iteration. The variable that is set equal to the private version of a particular thread executes the final iteration or the last section.

## 4. Program 4

Demonstration of reduction clause in parallel directive.

```
chinmayi@chinmayi-HP-Pavilion-Laptop-15-cs3xxx: ~  
chinmayi@chinmayi-HP-Pavilion-Laptop-15-cs3xxx:~$ subl reduction.c  
chinmayi@chinmayi-HP-Pavilion-Laptop-15-cs3xxx:~$ gcc -o reduction -fopenmp reduction.c  
chinmayi@chinmayi-HP-Pavilion-Laptop-15-cs3xxx:~$ ./reduction  
Hi from 0  
  value of x : 1  
Hi from 4  
  value of x : 1  
Hi from 2  
  value of x : 1  
Hi from 1  
  value of x : 1  
Hi from 3  
  value of x : 1  
Hi from 5  
  value of x : 1  
Final x:6  
chinmayi@chinmayi-HP-Pavilion-Laptop-15-cs3xxx:~$
```

Reduction clause specifies one or more thread-private variables that are subject to a reduction operation at the end of the parallel region. + operator needs to be specified to perform the reduction.

## 5. Programming exercise

### 1. Write a parallel program to calculate the sum of elements in an array.

Code:

```
sum.c
1 #include<stdio.h>
2 #include<omp.h>
3 #include<stdlib.h>
4 int main(int argc , char **argv)
5 {
6     double *Array, sum;
7     int array_size, i, threads, number;
8     if( argc != 3 )
9     {
10         printf("Very Few Arguments\n ");
11         printf("Syntax : exec <threads> <array-size>\n");
12         exit(-1);
13     }
14     threads=atoi(argv[1]);
15     if ((threads!=1) && (threads!=2) && (threads!=4) && (threads!=8) && (threads!= 16) )
16     {
17         printf("\nNumber of threads should be 1,2,4,8 or 16 for the execution of program. \n\n");
18         exit(-1);
19     }
20     array_size=atoi(argv[2]);
21     if (array_size <= 0) {
22         printf("\nArray Size Should Be Of Positive Value ");
23         exit(1);
24     }
25     printf("\nThreads      : %d ",threads);
26     printf("\nArray Size   : %d ",array_size);
27     Array = (double *) malloc(sizeof(double) * array_size);
28     printf("\nArray elements: ");
29     for (i = 0; i < array_size; i++) {
30         scanf("%d",&number);
31         Array[i] = number;
32     }
33     sum=0.0;
34     omp_set_num_threads(threads);
35     #pragma omp parallel for
36     for (i = 0; i < array_size; i++)
37     {
38         #pragma omp critical
39         sum = sum + Array[i];
40     }
41     free(Array);
42     printf("\nThe Sum Of Elements Of The Array Is: %lf\n", sum);
43     return 0;
44 }
```

Output:

```
chinmayi@chinmayi-HP-Pavilion-Laptop-15-cs3xxx: ~
chinmayi@chinmayi-HP-Pavilion-Laptop-15-cs3xxx:~$ subl sum.c
chinmayi@chinmayi-HP-Pavilion-Laptop-15-cs3xxx:~$ gcc -o sum -fopenmp sum.c
chinmayi@chinmayi-HP-Pavilion-Laptop-15-cs3xxx:~$ ./sum 4 6

Threads      : 4
Array Size   : 6
Array elements: 2 5 7 1 9 13

The Sum Of Elements Of The Array Is: 37.000000
chinmayi@chinmayi-HP-Pavilion-Laptop-15-cs3xxx:~$
```

2. Write a parallel program to calculate the  $a[i]=b[i]+c[i]$ , for all elements in array  $b[]$  and  $c[]$ .

Code:

```
~/sumtwoarrays.c - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
19      }
20      exit(-1);
21      array_size=atoi(argv[2]);
22      if (array_size <= 0) {
23          printf("\nArray Size Should Be Of Positive Value ");
24          exit(1);
25      }
26
27      printf("\nThreads      : %d ",threads);
28      printf("\nArray Size   : %d ",array_size);
29      Array = (double *) malloc(sizeof(double) * array_size);
30      Array1 = (double *) malloc(sizeof(double) * array_size);
31      Array2 = (double *) malloc(sizeof(double) * array_size);
32      printf("\nArray1 elements: ");
33
34      for (i = 0; i < array_size; i++) {
35          scanf("%d",&number);
36          Array1[i] = number;
37      }
38      printf("\nArray2 elements: ");
39
40      for (i = 0; i < array_size; i++) {
41          scanf("%d",&number);
42          Array2[i] = number;
43      }
44      omp_set_num_threads(threads);
45      #pragma omp parallel for
46      for (i = 0; i < array_size; i++)
47      {
48          #pragma omp critical
49          Array[i] = Array1[i] + Array2[i];
50      }
51
52      printf("\nThe Sums Of Elements Of Array1 and Array2 are: |");
53      for (i = 0; i < array_size; i++)
54      {
55          printf("%lf ",Array[i]);
56      }
57      printf("\n");
58      free(Array1);
59      free(Array2);
60      free(Array);
61      return 0;
62
63 }
```

Output:

```
chinmayi@chinmayi-HP-Pavilion-Laptop-15-cs3xxx: ~
chinmayi@chinmayi-HP-Pavilion-Laptop-15-cs3xxx:~$ subl sumtwoarrays.c
chinmayi@chinmayi-HP-Pavilion-Laptop-15-cs3xxx:~$ gcc -o sumtwoarrays -fopenmp sumtwoarrays.c
chinmayi@chinmayi-HP-Pavilion-Laptop-15-cs3xxx:~$ ./sumtwoarrays 4 4

Threads      : 4
Array Size   : 4
Array1 elements: 6 18 12 24
Array2 elements: 4 20 16 8

The Sums Of Elements Of Array1 and Array2 are: 10.000000 38.000000 28.000000 32.000000
chinmayi@chinmayi-HP-Pavilion-Laptop-15-cs3xxx:~$
```

### 3. Write a parallel program to find the largest among all elements in an array.

Code:

```
~/largest.c - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
1 #include<stdio.h>
2 #include<omp.h>
3 #include<stdlib.h>
4 int main(int argc , char **argv)
5 {
6     double *Array, largest;
7     int array_size, i, threads, number;
8     if( argc != 3 )
9     {
10         printf("Very Few Arguments\n");
11         printf("Syntax : exec <Threads> <array-size>\n");
12         exit(-1);
13     }
14     threads=atoi(argv[1]);
15     if ((threads!=1) && (threads!=2) && (threads!=4) && (threads!=8) && (threads!=16) )
16     {
17         printf("\nNumber of threads should be 1,2,4,8 or 16 for the execution of program. \n\n");
18         exit(-1);
19     }
20     array_size=atoi(argv[2]);
21     if (array_size <= 0) {
22         printf("\nArray Size Should Be Of Positive Value ");
23         exit(1);
24     }
25     printf("\nThreads      : %d ", threads);
26     printf("\nArray Size   : %d ", array_size);
27     Array = (double *) malloc(sizeof(double) * array_size);
28     printf("\nArray elements: ");
29     for (i = 0; i < array_size; i++) {
30         scanf("%d", &number);
31         Array[i] = number;
32     }
33     largest=Array[0];
34     omp_set_num_threads(threads);
35     #pragma omp parallel for
36     for (i = 0; i < array_size; i++)
37     {
38         #pragma omp critical
39         if(Array[i]>largest) largest = Array[i];
40     }
41     printf("\nThe Largest Element in the Array Is: %lf\n", largest);
42     free(Array);
43     return 0;
44 }
```

Output:

```
chinmayi@chinmayi-HP-Pavillon-Laptop-15-cs3xxx: ~
chinmayi@chinmayi-HP-Pavillon-Laptop-15-cs3xxx:~$ subl largest.c
chinmayi@chinmayi-HP-Pavillon-Laptop-15-cs3xxx:~$ gcc -o largest -fopenmp largest.c
chinmayi@chinmayi-HP-Pavillon-Laptop-15-cs3xxx:~$ ./largest 4 6

Threads      : 4
Array Size   : 6
Array elements: 3 7 1 2 8 8

The Largest Element in the Array Is: 8.000000
chinmayi@chinmayi-HP-Pavillon-Laptop-15-cs3xxx:~$
```