

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

Submitted by: Harsh Agarwal (181IT117)

1. Program 1

To understand and analyze shared clause in parallel directive.

```
#include<stdio.h>

#include<omp.h>

int main(){

    int x=0;

    #pragma omp parallel shared(x)

    {

        int tid=omp_get_thread_num();

        x=x+1;

        printf("Thread [%d] => value of x is %d \n",tid,x);

    }

}
```

Analysis:

Here we have initialized the value of x to 0 before entering the shared region. We have used the shared clause with parallel directive which will share the value of x declared before among all the threads and any change made to that value will be reflected across all the threads. Therefore, during the execution the value of x will keep on incrementing by 1 for all threads.

Output:

```
MINGW64:/d/Academics/5th Sem/Parallel Computing/Lab/Lab2
harsh@EDUCATION MINGW64 /d/Academics/5th Sem/Parallel Computing/Lab/Lab2
$ gcc -fopenmp q1.c

harsh@EDUCATION MINGW64 /d/Academics/5th Sem/Parallel Computing/Lab/Lab2
$ ./a.exe
Thread [6] => value of x is 1
Thread [0] => value of x is 6
Thread [7] => value of x is 7
Thread [3] => value of x is 3
Thread [2] => value of x is 4
Thread [4] => value of x is 5
Thread [5] => value of x is 2
Thread [1] => value of x is 2

harsh@EDUCATION MINGW64 /d/Academics/5th Sem/Parallel Computing/Lab/Lab2
$
```

2. Program 2 - Learn the concept of private(), firstprivate()

```
#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) private(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());

}
```

Analysis:

Here private and firstprivate clauses are used with parallel directives. In case of parallel, the earlier initialized value of i will be discarded and no value will be shared between different threads. In case of first parallel, the earlier declared value of i will be copied to all the threads when the control goes to parallel region. But even here the values between different threads will not be shared.

Output (private):

```
harsh@EDUCATION MINGW64 /d/Academics/5th Sem/Parallel Computing/Lab/Lab2
$ gcc -fopenmp q2.c

harsh@EDUCATION MINGW64 /d/Academics/5th Sem/Parallel Computing/Lab/Lab2
$ ./a.exe
Value before pragma i=10
Value after entering pragma i=4 tid=0
Value after changing value i=4 tid=0
Value after entering pragma i=0 tid=1
Value after changing value i=1 tid=1
Value after entering pragma i=0 tid=2
Value after changing value i=2 tid=2
Value after entering pragma i=0 tid=3
Value after changing value i=3 tid=3
Value after having pragma i=10 tid=0
```

Output (firstprivate):

```
harsh@EDUCATION MINGW64 /d/Academics/5th Sem/Parallel Computing/Lab/Lab2
$ gcc -fopenmp q2.c

harsh@EDUCATION MINGW64 /d/Academics/5th Sem/Parallel Computing/Lab/Lab2
$ ./a.exe
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=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 entering pragma i=10 tid=2
Value after changing value i=12 tid=2
Value after having pragma i=10 tid=0
```

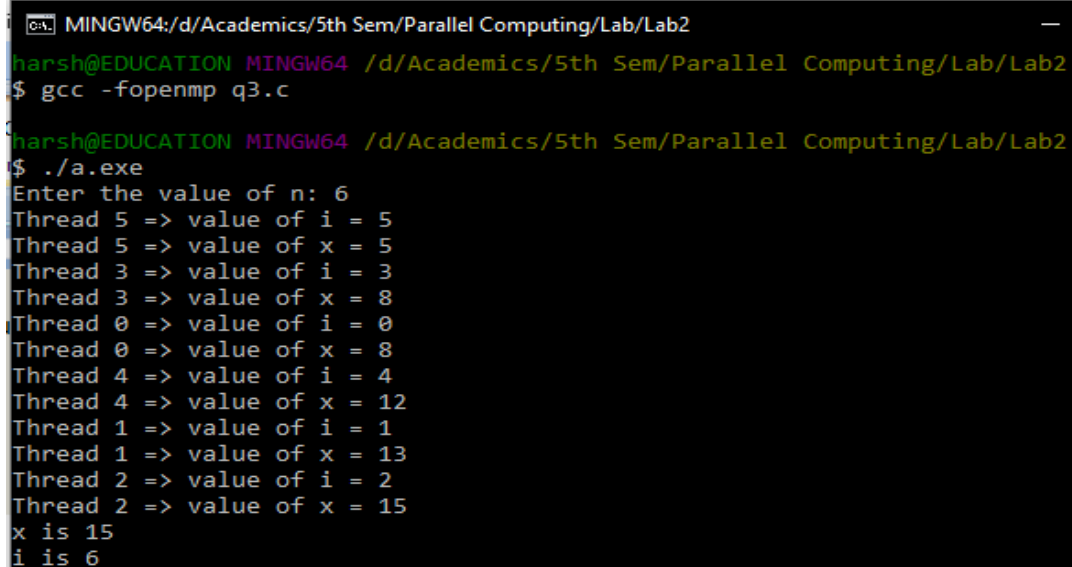
3. Program 3 - Learn the working of lastprivate() clause:

```
#include<stdio.h>
#include<omp.h>
void main(){
    int x=0,i,n;
    printf("Enter the value of n: ");
    scanf("%d",&n);
    #pragma omp parallel
    {
        int id=omp_get_thread_num();
        #pragma omp for lastprivate(i)
        for(i=0;i<n;i++){
            printf("Thread %d => value of i = %d\n",id,i);
            x=x+i;
            printf("Thread %d => value of x = %d\n",id,x);
        }
    }
    printf("x is %d\n",x);
    printf("i is %d\n",i);
}
```

Analysis:

We have not initialized *i* with any value before entering the parallel region. Therefore I will take random values for each thread inside the parallel region. By using the lastprivate clause with for directive, we are ensuring that the value of *i* when printed after coming out of the loop will return the last value of *i*. It will give the last iteration of *i*. Since the total number of iterations is *n*, the value of *i* will be equal to *n*.

Output:



```
MINGW64:/d/Academics/5th Sem/Parallel Computing/Lab/Lab2
harsh@EDUCATION MINGW64 /d/Academics/5th Sem/Parallel Computing/Lab/Lab2
$ gcc -fopenmp q3.c
harsh@EDUCATION MINGW64 /d/Academics/5th Sem/Parallel Computing/Lab/Lab2
$ ./a.exe
Enter the value of n: 6
Thread 5 => value of i = 5
Thread 5 => value of x = 5
Thread 3 => value of i = 3
Thread 3 => value of x = 8
Thread 0 => value of i = 0
Thread 0 => value of x = 8
Thread 4 => value of i = 4
Thread 4 => value of x = 12
Thread 1 => value of i = 1
Thread 1 => value of x = 13
Thread 2 => value of i = 2
Thread 2 => value of x = 15
x is 15
i is 6
```

4. Demonstration of reduction clause in parallel directive.

```
#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();

        x=x+1;

        printf("Thread %d => Value of x = %d\n",id,x);

    }

    printf("Final value of x = %d\n",x);

}
```

Analysis:

We use the reduction clause when we want to use an operator on the values of that are obtained individually after computation inside the parallel regions. An alternative way to solve this problem is by storing the results obtained from each thread and then performing the operations on them using a loop. But, by using the reduction clause, we can perform this in single pass. Here we are calculating the sum of values of x obtained from each thread and the final value is displayed after the loop.

Output:

```
harsh@EDUCATION MINGW64 /d/Academics/5th Sem/Parallel Computing/Lab/Lab2
$ gcc -fopenmp q4.c

harsh@EDUCATION MINGW64 /d/Academics/5th Sem/Parallel Computing/Lab/Lab2
$ ./a.exe
Thread 5 => Value of x = 1
Thread 3 => Value of x = 1
Thread 2 => Value of x = 1
Thread 1 => Value of x = 1
Thread 0 => Value of x = 1
Thread 4 => Value of x = 1
Final value of x = 6
```

5. Programming exercise

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

```
#include<stdio.h>

#include<omp.h>

void main(){

    printf("Enter the size of array: \n");

    int n;

    scanf("%d",&n);

    int a[n];

    printf("Enter the array elements: \n");

    for(int i = 0;i<n;i++){

        scanf("%d",&a[i]);

    }

    int sum=0;

    #pragma omp parallel for reduction (+:sum)

    for (int i=0;i<n;i++){

        sum=sum+a[i];

    }

    printf("Sum of the array is : %d",sum);

}
```

Analysis:

Using the reduction clause with parallel for directive, we can traverse the array elements using for loop and at the same time increment the value of sum variable. At the end, the sum variable will store the sum of all the elements because + operator is used with the reduction clause.

Output:

```
harsh@EDUCATION MINGW64 /d/Academics/5th Sem/Parallel Computing/Lab/Lab2
$ gcc -fopenmp q5_1.c

harsh@EDUCATION MINGW64 /d/Academics/5th Sem/Parallel Computing/Lab/Lab2
$ ./a.exe
Enter the size of array:
5
Enter the array elements:
1 4 2 1 3
Sum of the array is : 11
```

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

```
#include <stdio.h>

#include <omp.h>

int main(void){

    int n;

    printf("Enter the size of arrays: \n");

    scanf("%d", &n);

    int A[n], B[n], C[n];

    printf("Enter the array B elements: \n");

    for (int a = 0; a < n; a++){

        scanf("%d", &B[a]);

    }

    printf("Enter the array C elements: \n");

    for (int a = 0; a < n; a++){

        scanf("%d", &C[a]);

    }

    #pragma omp parallel shared(A)
```

```

{

    #pragma omp for

    for (int i = 0; i < n; i++){

        A[i] = B[i] + C[i];

    }

}

printf("The elements of array A are: \n");

for (int a = 0; a < n; a++){

    printf("%d ", A[a]);

}

}

```

Analysis:

The array A is shared among the arrays B and C. When we are iterating over B and C in a for loop, we can add their value and store it in our shared array A. Since A is shared, the changes in A in one thread will be taken into account when its value will be changed in another thread.

Output:

```

harsh@EDUCATION MINGW64 /d/Academics/5th Sem/Parallel Computing/Lab/Lab2
$ gcc -fopenmp q5_2.c

harsh@EDUCATION MINGW64 /d/Academics/5th Sem/Parallel Computing/Lab/Lab2
$ ./a.exe
Enter the size of arrays:
4
Enter the array B elements:
1 1 3 2
Enter the array C elements:
2 5 6 1
The elements of array A are:
3 6 9 3

```


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

```
#include <stdio.h>

#include <omp.h>

int main(){

    int n, cur_max = 0;

    printf("Enter the size of arrays: \n");

    scanf("%d", &n);

    int array[n];

    printf("Enter the array elements: \n");

    for (int a = 0; a < n; a++){

        scanf("%d", &array[a]);

    }

    #pragma omp parallel for

    for (int i = 0; i < n; i = i + 1){

        if (array[i] > cur_max)

            #pragma omp critical

            if(array[i] > cur_max)

                cur_max = array[i];

    }

    printf("Largest number = %d\n", cur_max);

}
```

Analysis:

Assign the cur_max as 0 in the beginning. We will traverse the array elements using for loop and if the array element is greater the cur_max, we will enter into critical region and update

our cur_max variable. The critical region will ensure that the update operation is being performed by a single thread at a time.

Output:

```
harsh@EDUCATION MINGW64 /d/Academics/5th Sem/Parallel Computing/Lab/Lab2
$ gcc -fopenmp q5_3.c

harsh@EDUCATION MINGW64 /d/Academics/5th Sem/Parallel Computing/Lab/Lab2
$ ./a.exe
Enter the size of arrays:
5
Enter the array elements:
3 10 1 4 5
Largest number = 10
```