

Fall Sem 2021-22

Assignment: 3

Date : 7/09/21

Name : Ishan Sagar Jogalekar

Reg no : 19BCE2250

Course: Parallel and distributed computing LAB - CSE4001

Slot : L55+L56

### **SCENARIO – I**

Write a simple OpenMP program to employ a '*reduction*' clause to express the reduction of a for loop. In order to specify the reduction in OpenMP, we must provide

1. An operation (+ / \* / o)
2. A reduction variable (sum / product / reduction). This variable holds the result of the computation.

**Ans:**

**SUM Operation -**

#### **BRIEF ABOUT YOUR APPROACH:**

sum will be in reduction clause as it is to be combined afterwards.

#### **SOURCE CODE:**

```
#include <stdio.h>
#include <omp.h>
void main(){
    printf("***Sum of elements in array using reduction in OpenMP***\n");
    printf("19BCE2250 - Ishan Jogalekar\n");
    int arr[6]={1,2,3,4,5,6};
    int sum=0,i;
    //Parallel OpenMP block with shared and reduction clause
    #pragma omp parallel shared(arr,sum)
    {
```

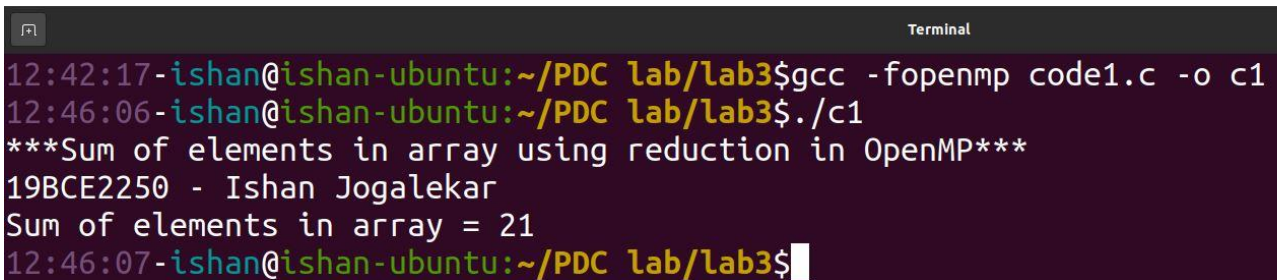
```

    #pragma omp for reduction(+:sum)
    for (i = 0; i < 6; i++) //For loop to iterate between each element of array
    {
        sum += arr[i]; //Calculating Sum
    }
}

printf("Sum of elements in array = %d\n",sum);
}

```

## **EXECUTION:**



```

12:42:17-ishan@ishan-ubuntu:~/PDC lab/lab3$gcc -fopenmp code1.c -o c1
12:46:06-ishan@ishan-ubuntu:~/PDC lab/lab3$./c1
***Sum of elements in array using reduction in OpenMP***
19BCE2250 - Ishan Jogalekar
Sum of elements in array = 21
12:46:07-ishan@ishan-ubuntu:~/PDC lab/lab3$

```

## **RESULT:**

The reduction clause specifies that one or more variables that are private to each thread are the subject of a reduction operation at the end of the parallel region. We will get output as sum of elements in array.

## **Multiplication Operation –**

### **BRIEF ABOUT YOUR APPROACH:**

Multiplication will be in reduction clause as it is to be combines afterwards. Also shared block is used to provide array and multiplication variable as shared variables within processors.

### **SOURCE CODE:**

```

#include <stdio.h>
#include <omp.h>
void main(){
    printf("***Product of elements in array using reduction in OpenMP***\n");
    printf("19BCE2250 - Ishan Jogalekar\n");
    int arr[6]={1,2,3,4,5,6};
    int mul=1,i;
    //Parallel OpenMP block with shared and reduction clause
    #pragma omp parallel shared(arr,mul)
    {

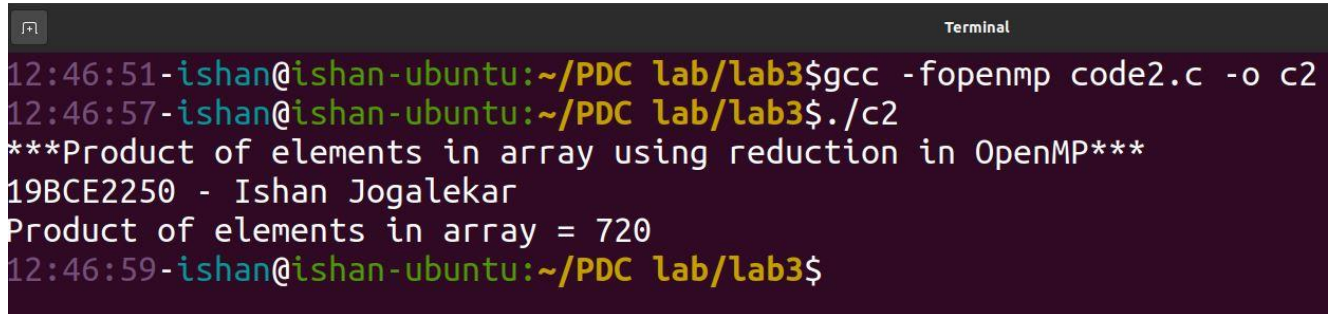
```

```

#pragma omp for reduction(*:mul)
for (i = 0; i < 6; i++) //For loop to iterate between each element of array
{
    mul *= arr[i]; //Calculating Sum
}
}
printf("Product of elements in array = %d\n",mul);
}

```

## **EXECUTION:**



```

12:46:51-ishan@ishan-ubuntu:~/PDC lab/lab3$gcc -fopenmp code2.c -o c2
12:46:57-ishan@ishan-ubuntu:~/PDC lab/lab3$./c2
***Product of elements in array using reduction in OpenMP***
19BCE2250 - Ishan Jogalekar
Product of elements in array = 720
12:46:59-ishan@ishan-ubuntu:~/PDC lab/lab3$

```

## **RESULT:**

The reduction clause specifies that one or more variables that are private to each thread are the subject of a reduction operation at the end of the parallel region. We will get output as product of elements in array.

## **SCENARIO – II**

Write an OpenMP program to find the smallest element in a list of numbers using OpenMP REDUCTION clause.

### **Description**

Largest element in a list of numbers is found using OpenMP PARALLEL DO directive and REDUCTION clause. Reductions are a sufficiently common type of operation. OpenMP includes a reduction data scope clause just to handle the variable. In reduction, we repeatedly apply a binary operator to a variable and some other value, and store the result back in the variable. In this example we have added the clause REDUCTION ( MAX : LargeNumber), which tells the compiler that LargeNumber is the target of a sum reduction operation.

**Ans:**

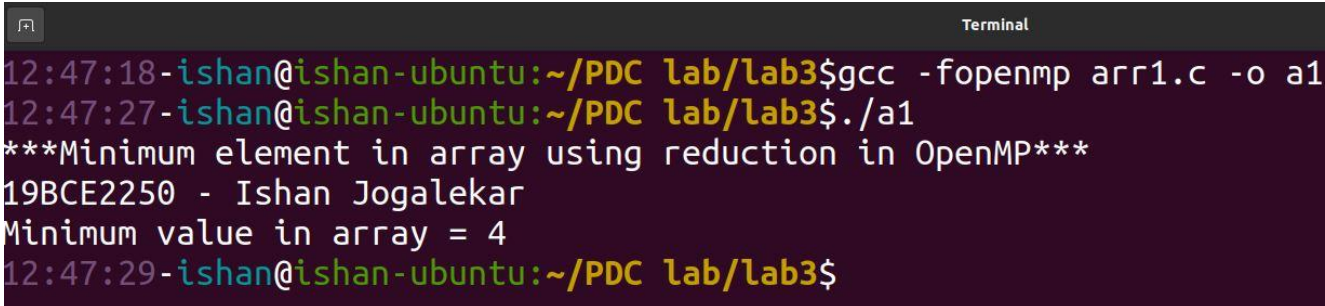
## **BRIEF ABOUT YOUR APPROACH:**

OpenMP has predefined min and max reduction operators for C, so we will use it. It is min operator in reduction clause.

## **SOURCE CODE:**

```
#include <stdio.h>
#include <omp.h>
int main(void){
    printf("***Minimum element in array using reduction in OpenMP***\n");
    printf("19BCE2250 - Ishan Jogalekar\n");
    int arr[10] = {10, 11, 65, 4, 51, 73, 8, 9, 12, 6};
    int MinValue = 100, i;
    #pragma omp parallel reduction(min : MinValue)
    {
        #pragma omp for
        for(i = 0; i < 10; i++){
            if(arr[i] < MinValue){
                MinValue = arr[i];
            }
        }
    }
    printf("Minimum value in array = %d\n", MinValue);
}
```

## **EXECUTION:**



```
Terminal
12:47:18-ishan@ishan-ubuntu:~/PDC lab/lab3$gcc -fopenmp arr1.c -o a1
12:47:27-ishan@ishan-ubuntu:~/PDC lab/lab3$./a1
***Minimum element in array using reduction in OpenMP***
19BCE2250 - Ishan Jogalekar
Minimum value in array = 4
12:47:29-ishan@ishan-ubuntu:~/PDC lab/lab3$
```

## **RESULT:**

One can use reduction clause in OpenMP in order to carry out various operations like addition, multiplication, logical OR, logical AND, etc. This clause specifies that one or more variables that are private to each thread are the subject of a reduction operation at the end of the parallel region.

For given scenario we used min operator in order to find minimum value. Hence, we used min operator in reduction clause.

### SCENARIO – III

Write an OpenMP program to find the Max and Min elements in a list of numbers using OpenMP Critical clause to understand:

Hint: As Max & Min value is easily prone to change by another thread after comparing with Array [Index], the use of '*critical*' section is highly demanded to execute such computation on one thread at a time.

#### Example:

```
#pragma omp critical
{
    if (Array Index is greater/lesser than Max/Min)
}
```

**Ans:**

#### BRIEF ABOUT YOUR APPROACH:

The code will be same as previous one, all comparisons required in the loop will be in critical section. Using max function to find maximum value and min for minimum value with the help of if condition.

#### SOURCE CODE:

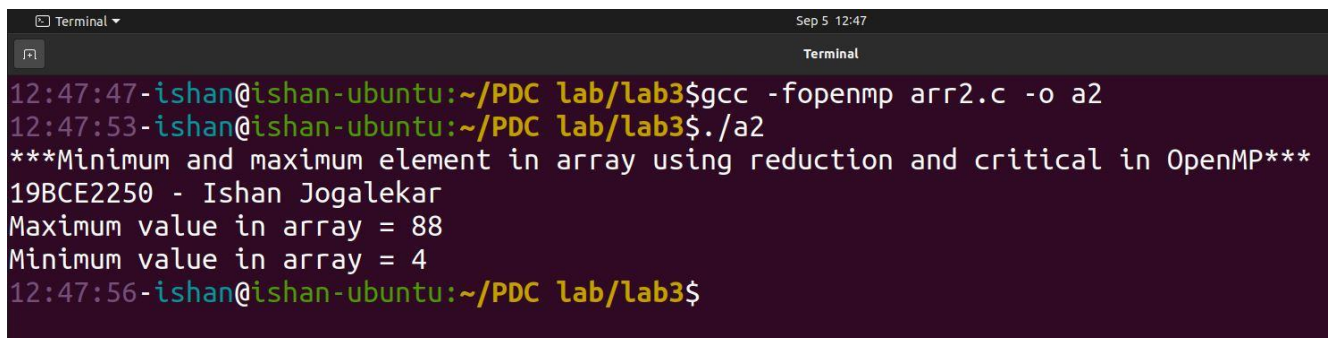
```
#include <stdio.h>
#include <omp.h>
int main(void){
    printf("***Minimum and maximum element in array using reduction and
critical in OpenMP***\n");
    printf("19BCE2250 - Ishan Jogalekar\n");
    int arr[10] = {12, 11, 31, 4, 5, 7, 88, 19, 10, 65};
    int MaxValue = 0, i, MinValue = 100;
    #pragma omp parallel reduction(max : MaxValue) reduction(min:
MinValue)
    {
        #pragma omp for
        for(i = 0; i < 10; i++){
            #pragma omp critical
            {
                if(arr[i] > MaxValue){
```

```

        MaxValue = arr[i];
    }
    if(arr[i] < MinValue){
        MinValue = arr[i];
    }
}
}
}
printf("Maximum value in array = %d\n", MaxValue);
printf("Minimum value in array = %d\n", MinValue);
}

```

### **EXECUTION:**



```

Terminal
Sep 5 12:47
12:47:47-ishan@ishan-ubuntu:~/PDC lab/lab3$gcc -fopenmp arr2.c -o a2
12:47:53-ishan@ishan-ubuntu:~/PDC lab/lab3$./a2
***Minimum and maximum element in array using reduction and critical in OpenMP***
19BCE2250 - Ishan Jogalekar
Maximum value in array = 88
Minimum value in array = 4
12:47:56-ishan@ishan-ubuntu:~/PDC lab/lab3$

```

### **RESULT:**

The use of 'critical' section demands to execute computation on one thread at a time, as critical section. Hence using min and max operator within critical section with the help of if conditional statements in order to find maximum and minimum elements in array.