

**CSE4001 - Parallel and Distributed Computing, Fall 2019**  
**Vellore Institute of Technology**  
**Instructor: Prof Deebak B D - SCOPE**

**Lab report**

**Title of Lab: OpenMP Reduction and critical clause**

**Assessment #: 3**

**Date: 10/08/2019**

**Author's name: Gagan Deep Singh**

**Registration ID: 17BCI0140**

**Lab section: Friday L59 + L60**

**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.

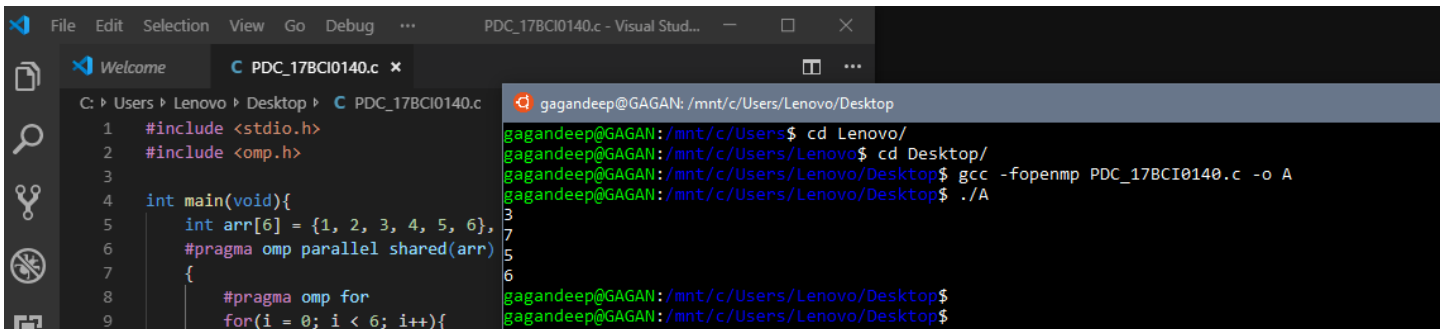
**BRIEF ABOUT YOUR APPROACH:** sum will be in reduction clause as it is to be combined afterwards.

**SOURCE CODE:**

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

int main(void){
    int arr[6] = {1, 2, 3, 4, 5, 6}, i, sum;
    #pragma omp parallel shared(arr) reduction(+: sum)
    {
        #pragma omp for
        for(i = 0; i < 6; i++){
            sum += arr[i];
        }
        printf("%d\n", sum);
    }
}
```

**EXECUTION:**



```
File Edit Selection View Go Debug ... PDC_17BCI0140.c - Visual Stud...
Welcome C PDC_17BCI0140.c x
C:\Users\Lenovo\Desktop\ PDC_17BCI0140.c
1 #include <stdio.h>
2 #include <omp.h>
3
4 int main(void){
5     int arr[6] = {1, 2, 3, 4, 5, 6},
6     #pragma omp parallel shared(arr)
7     {
8         #pragma omp for
9         for(i = 0; i < 6; i++){
gagandeep@GAGAN: /mnt/c/Users/Lenovo/Desktop$ cd Lenovo/
gagandeep@GAGAN: /mnt/c/Users/Lenovo/Desktop$ cd Desktop/
gagandeep@GAGAN: /mnt/c/Users/Lenovo/Desktop$ gcc -fopenmp PDC_17BCI0140.c -o A
gagandeep@GAGAN: /mnt/c/Users/Lenovo/Desktop$ ./A
3
7
5
6
gagandeep@GAGAN: /mnt/c/Users/Lenovo/Desktop$
gagandeep@GAGAN: /mnt/c/Users/Lenovo/Desktop$
```

**RESULTS:** 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.

## 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.

**BRIEF ABOUT YOUR APPROACH:** OpenMP has predefined min and max reduction operators for C, so we will use it.

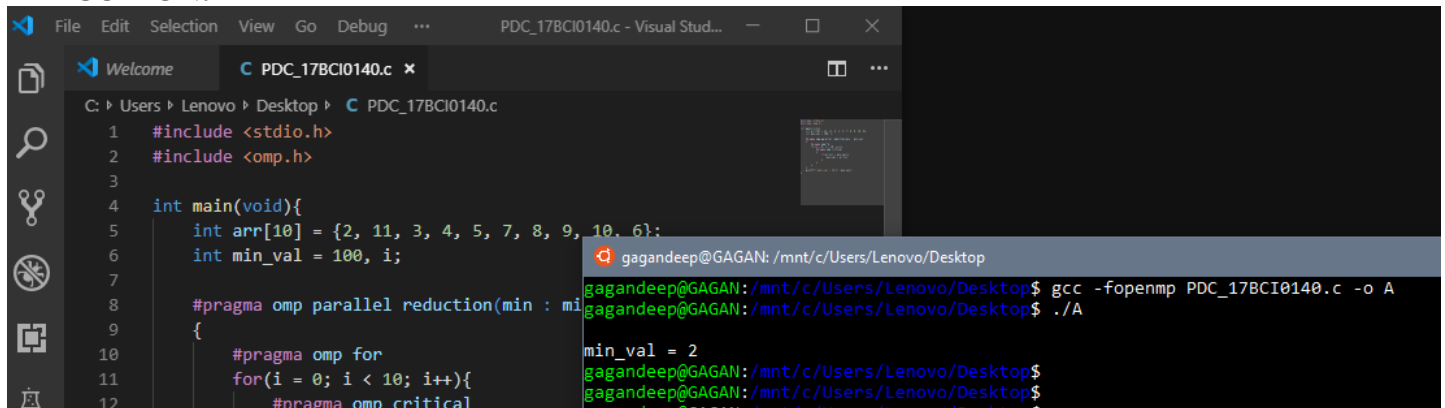
### SOURCE CODE:

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

int main(void){
    int arr[10] = {2, 11, 3, 4, 5, 7, 8, 9, 10, 6};
    int min_val = 100, i;

    #pragma omp parallel reduction(min : min_val)
    {
        #pragma omp for
        for(i = 0; i < 10; i++){
            if(arr[i] < min_val){
                min_val = arr[i];
            }
        }
    }
    printf("\nmin_val = %d\n", min_val);
}
```

### EXECUTION:

The screenshot shows a Visual Studio Code editor with a C file named PDC\_17BCI0140.c. The code is as follows:

```
1 #include <stdio.h>
2 #include <omp.h>
3
4 int main(void){
5     int arr[10] = {2, 11, 3, 4, 5, 7, 8, 9, 10, 6};
6     int min_val = 100, i;
7
8     #pragma omp parallel reduction(min : min_val)
9     {
10         #pragma omp for
11         for(i = 0; i < 10; i++){
12             #pragma omp critical
```

The terminal output shows the following commands and results:

```
gagandeep@GAGAN: /mnt/c/Users/Lenovo/Desktop$ gcc -fopenmp PDC_17BCI0140.c -o A
gagandeep@GAGAN: /mnt/c/Users/Lenovo/Desktop$ ./A
min_val = 2
gagandeep@GAGAN: /mnt/c/Users/Lenovo/Desktop$
```

### RESULTS:

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.

### 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.

**BRIEF ABOUT YOUR APPROACH:** The code will be same as previous one, all comparisons required in the loop will be in critical section.

#### SOURCE CODE:

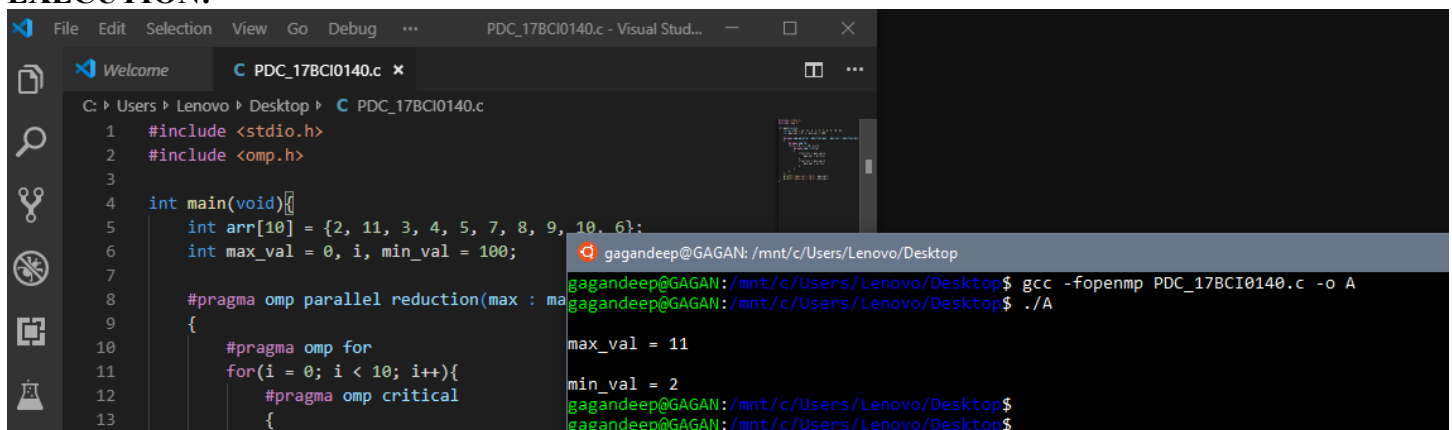
```
#include <stdio.h>
#include <omp.h>

int main(void){
    int arr[10] = {2, 11, 3, 4, 5, 7, 8, 9, 10, 6};
    int max_val = 0, i, min_val = 100;

    #pragma omp parallel reduction(max : max_val) reduction(min: min_val)
    {
        #pragma omp for
        for(i = 0; i < 10; i++){
            #pragma omp critical
            {
                if(arr[i] > max_val){
                    max_val = arr[i];
                }
                if(arr[i] < min_val){
                    min_val = arr[i];
                }
            }
        }
    }

    printf("\nmax_val = %d\n", max_val);
    printf("\nmin_val = %d\n", min_val);
}
```

#### EXECUTION:



```
gagandeep@GAGAN: /mnt/c/Users/Lenovo/Desktop
gagandeep@GAGAN:/mnt/c/Users/Lenovo/Desktop$ gcc -fopenmp PDC_17BCI0140.c -o A
gagandeep@GAGAN:/mnt/c/Users/Lenovo/Desktop$ ./A
max_val = 11
min_val = 2
gagandeep@GAGAN:/mnt/c/Users/Lenovo/Desktop$
```

**RESULTS:** The use of 'critical' section demands to execute computation on one thread at a time, as critical section.