Computer Science & Engineering

CSE4001

Parallel and Distributed Computing

LAB ASSIGNMENT 3

Submitted to **Prof. DEEBAK B.D.**

TOPIC: PROBLEMS USING OPENMP

NAME: PUNIT MIDDHA

REG.NO: 19BCE2060

SLOT: L55+L56

DATE: 07/09/2021

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.

Example:

Let's parallelize the following for loop:

```
sum = 0;
#pragma omp parallel for shared(sum, a) reduction(+: sum)
for (auto i = 0; i < 9; i++)
{
   sum += a[i]
}
```

Each thread has sumloc, which is a local copy of the reduction variable. The threads then perform the following computations

- **Thread 1**

sumloc_1 = a[0] + a[1] + a[2]

- **Thread 2**

sumloc_2 = a[3] + a[4] + a[5]

- **Thread 3**

sumloc_3 = a[6] + a[7] + a[8]

In the end, when the treads join together, OpenMP reduces local copies to the shared reduction variable

sum = sumloc_1 + sumloc_2 + sumloc_3

1. Show the complete code execution of above logic using C/C++

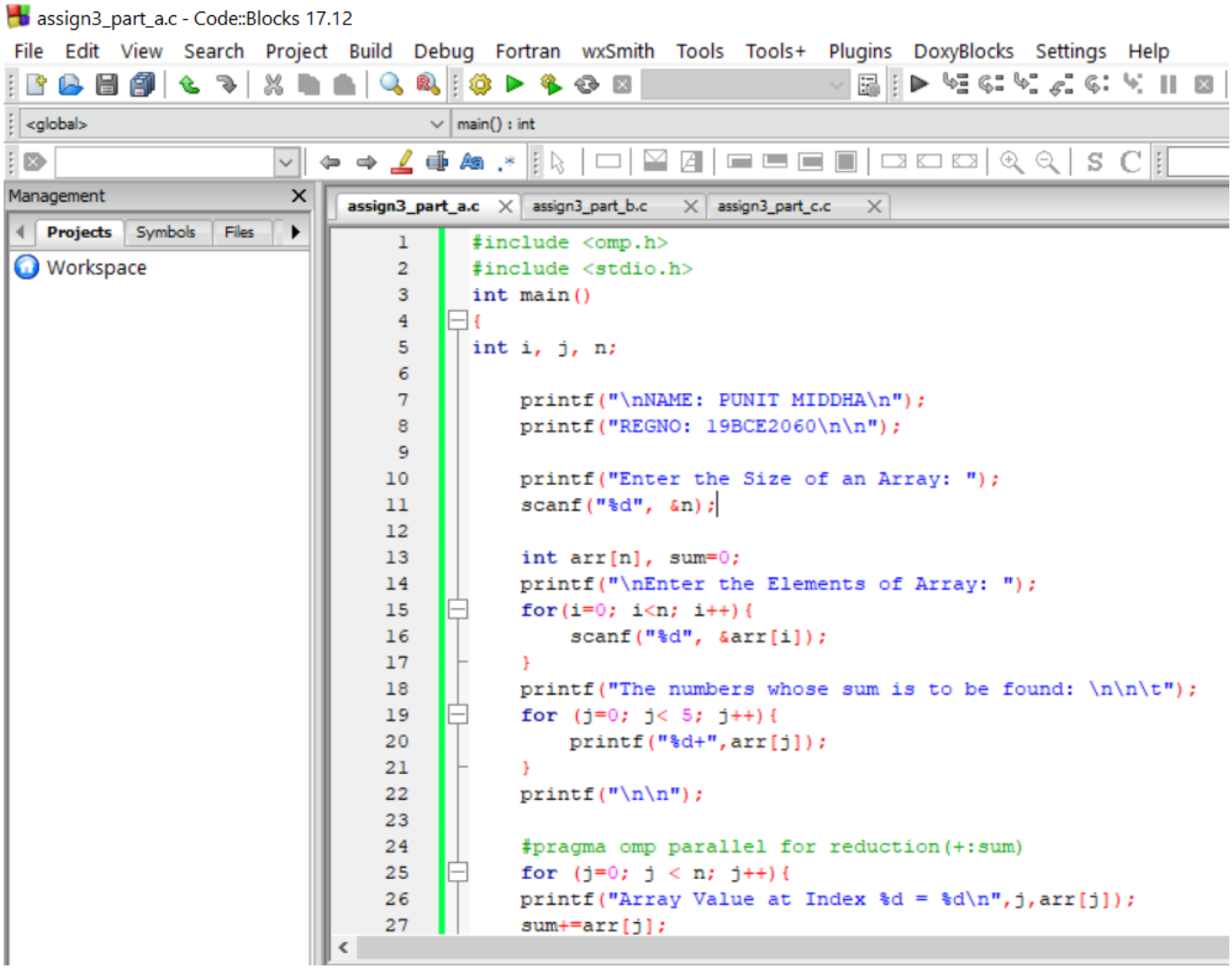
BRIEF ABOUT YOUR APPROACH:

- Firstly, taking the number of array element(n) then taking the input of array elements separated with space.
- Initialize the variable Sum as '0'
- #pragma omp parallel for reduction(+:sum) sum is passed in the reduction clause with operator '+'.

SOURCE CODE:

```
#include <omp.h>

#include <stdio.h>

int main()

{

int i, j, n;


    printf("\nNAME: PUNIT MIDDHA\n");

    printf("REGNO: 19BCE2060\n\n");


    printf("Enter the Size of an Array: ");

    scanf("%d", &n);


    int arr[n], sum=0;

    printf("\nEnter the Elements of Array: ");

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

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

    }

    printf("The numbers whose sum is to be found: \n\n\t");

    for (j=0; j< 5; j++){

        printf("%d+",arr[j]);

    }

    printf("\n\n");


    #pragma omp parallel for reduction(+:sum)

    for (j=0; j < n; j++){

    printf("Array Value at Index %d = %d\n",j,arr[j]);
```

```
    sum+=arr[j];
    }
    printf("\n\n\tSum of given numbers is = %d\n",sum);


}
```



EXECUTION:

```
"C:\Users\Punit Middha\Desktop\PDC\assign3_part_a.exe"

NAME: PUNIT MIDDHA
REGNO: 19BCE2060

Enter the Size of an Array: 6

Enter the Elements of Array: 10 20 30 45 75 32
The numbers whose sum is to be found:

        10+20+30+45+75+

Array Value at Index 1 = 20
Array Value at Index 2 = 30
Array Value at Index 5 = 32
Array Value at Index 0 = 10
Array Value at Index 4 = 75
Array Value at Index 3 = 45


        Sum of given numbers is = 212

Process returned 0 (0x0)   execution time : 19.847 s
Press any key to continue.
```

REMARKS:

- As you can see from compilation window, sum is a shared variable.
- According to the given array, we are getting the desired output of Sum in the given array with the help of shared, private and reduction clauses.

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.

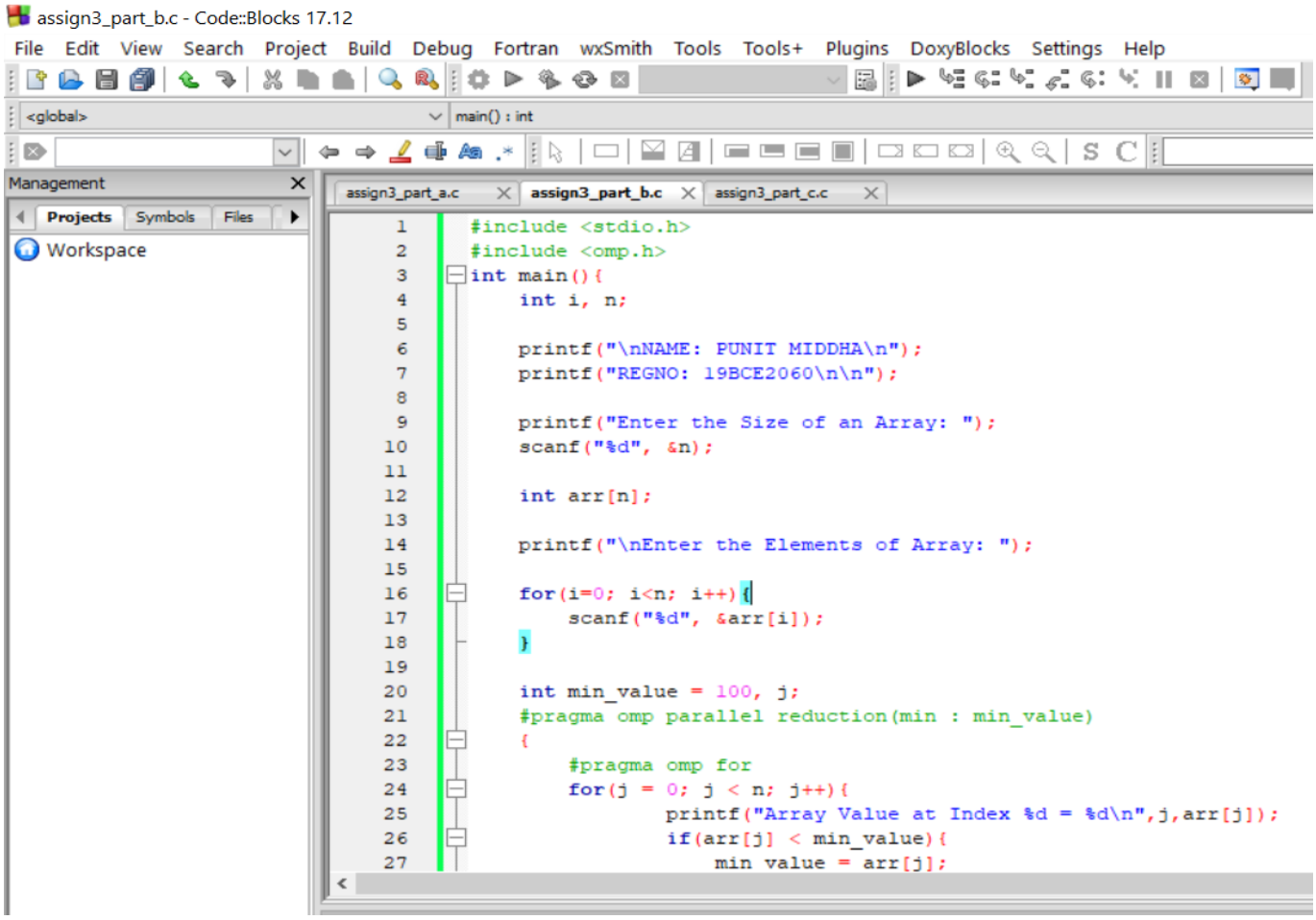
BRIEF ABOUT YOUR APPROACH:

- Firstly, taking the number of array element(n) then taking the input of array elements separated with space.
- For this case I would define variable min_value for minimum value.
- min : min_value is passed in reduction clause for evaluating min value.
- Initialise min_value to 1st element of array, if array value is less than min_value then we'll redefine min_value as that value.

SOURCE CODE:

```
#include <stdio.h>

#include <omp.h>

int main(){

    int i, n;

    printf("\nNAME: PUNIT MIDDHA\n");

    printf("REGNO: 19BCE2060\n\n");

    printf("Enter the Size of an Array: ");

    scanf("%d", &n);

    int arr[n];


    printf("\nEnter the Elements of Array: ");

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

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

```
    }
    int min_value = 100, j;
    #pragma omp parallel reduction(min : min_value)
    {
        #pragma omp for
        for(j = 0; j < n; j++){
                printf("Array Value at Index %d = %d\n",j,arr[j]);
                if(arr[j] < min_value){
                    min_value = arr[j];
            }
        }
    }
    printf("\n\tMinimum value = %d\n", min_value);
}
```



EXECUTION:

```
"C:\Users\Punit Middha\Desktop\PDC\assign3_part_b.exe"

NAME: PUNIT MIDDHA
REGNO: 19BCE2060

Enter the Size of an Array: 7

Enter the Elements of Array: 8 5 12 64 2 47 31
Array Value at Index 3 = 64
Array Value at Index 5 = 47
Array Value at Index 6 = 31
Array Value at Index 0 = 8
Array Value at Index 1 = 5
Array Value at Index 4 = 2
Array Value at Index 2 = 12

        Minimum value = 2

Process returned 0 (0x0)   execution time : 27.504 s
Press any key to continue.
```

REMARKS:

- Variable min_value is a shared variable in this scenario.
- Here min : min_value represents variable min_value will store minimum value.
- We got the desired output as expected , i.e., for given case min=2.

P.T.O.

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)
        }
```

BRIEF ABOUT YOUR APPROACH:

- Firstly, taking the number of array element(n) then taking the input of array elements separated with space.
- For this case we are using Critical clause for each if block.
- The two if blocks are separate and evaluate separate values while working on same array, in order to avoid any conflict due to threads we use Critical clause.
- We will be using #pragma omp parallel reduction(max : max_value) reduction(min: min_value) in this question because we have to get maximum and minimum values both in same code.

SOURCE CODE:

```
#include <stdio.h>

#include <omp.h>

int main(void){

    int i, n;

    printf("\nNAME: PUNIT MIDDHA\n");

    printf("REGNO: 19BCE2060\n\n");

    printf("Enter the Size of an Array: ");

    scanf("%d", &n);


    int arr[n], sum=0;
```

```
    printf("\nEnter the Elements of Array: ");


    for(i=0; i<n; i++){
        scanf("%d", &arr[i]);
    }
    int max_value, j, min_value;
    #pragma omp parallel reduction(max : max_value) reduction(min: min_value)
    {
        #pragma omp for
        for(j = 0; j < n; j++){
            printf("Array Value at Index %d = %d\n",j,arr[j]);
            #pragma omp critical
            {
                if(arr[j] > max_value){
                    max_value = arr[j];
                }
                if(arr[j] < min_value){
                    min_value = arr[j];
                }
            }
        }
    }
    printf("\n\tMaximum value = %d\n", max_value);
    printf("\n\tMinimum value = %d\n", min_value);
}
```

assign3_part_c.c - Code::Blocks 17.12

File Edit View Search Project Build Debug Fortran wxSmith Tools Tools+ Plugins DoxyBlocks Settings Help

<global> main(void) : int

Management

Projects Symbols Files

Workspace

assign3_part_a.c assign3_part_b.c assign3_part_c.c

```c
#include <stdio.h>
#include <omp.h>
int main(void){
    int i, n;

    printf("\nNAME: PUNIT MIDDHA\n");
    printf("REGNO: 19BCE2060\n\n");

    printf("Enter the Size of an Array: ");
    scanf("%d", &n);

    int arr[n], sum=0;

    printf("\nEnter the Elements of Array: ");

    for(i=0; i<n; i++){
        scanf("%d", &arr[i]);
    }

    int max_value, j, min_value;
    #pragma omp parallel reduction(max : max_value) reduction(min: min_value)
    {
        #pragma omp for
        for(j = 0; j < n; j++){
            printf("Array Value at Index %d = %d\n",j,arr[j]);
            #pragma omp critical
            {
```

EXECUTION:

"C:\Users\Punit Middha\Desktop\PDC\assign3_part_c.exe"

```
NAME: PUNIT MIDDHA
REGNO: 19BCE2060

Enter the Size of an Array: 10

Enter the Elements of Array: 15 21 14 84 36 95 75 62 05 88
Array Value at Index 5 = 95
Array Value at Index 8 = 5
Array Value at Index 0 = 15
Array Value at Index 1 = 21
Array Value at Index 7 = 62
Array Value at Index 6 = 75
Array Value at Index 4 = 36
Array Value at Index 9 = 88
Array Value at Index 2 = 14
Array Value at Index 3 = 84

        Maximum value = 95

        Minimum value = 5

Process returned 0 (0x0)   execution time : 22.187 s
Press any key to continue.
```

REMARKS:

- Here, we can see that the threads of Min and Max work separately due to critical clause.
- we get to display Final outputs Max and Min as per the given input array.
- In this case, max_value and min_value are variables used to store maximum and minimum values respectively and are initialised separately and We pass them both under shared clause.
- In order to pass two different variables under reduction, we can write reduction clause twice.
- We got the desired output as expected , i.e., Maximum value = 95 & Minimum value = 5.