NAME: MOUNVI PODAPATI
REG. NO: 19BCE0396
SLOT: L15 + L16
FACULTY: PROF. DEEPAK B.D
DATED: 1ST SEPTEMBER 2021

LAB ASSESSMENT – 3

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.

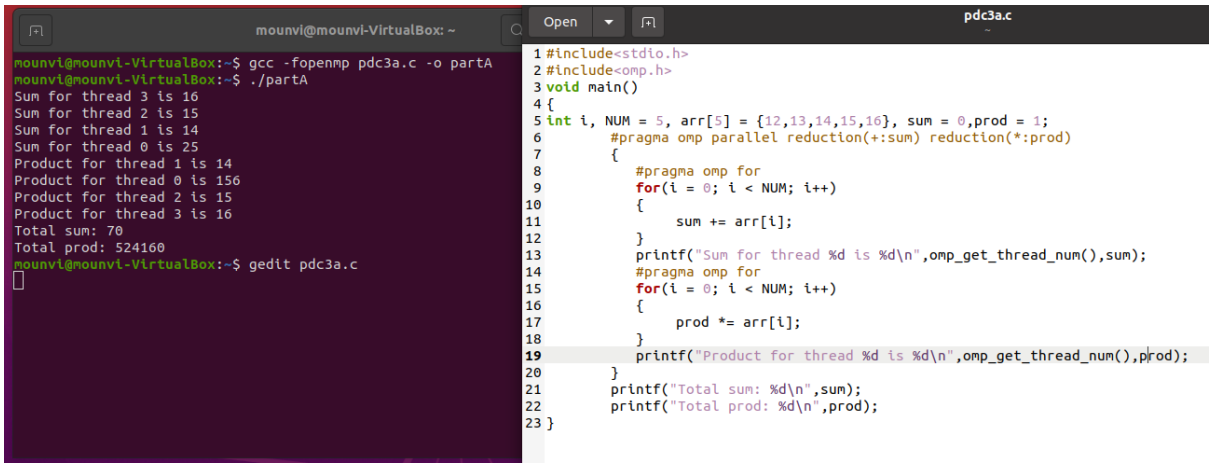
BRIEF ABOUT THE APPROACH:

- Sum and Prod will be a part of the reduction clause because they are required to be combined afterwards.
- #pragma omp for is used for invoking a parallelized approach to for loop

SOURCE CODE:

```
#include<stdio.h>
#include<omp.h>
void main()
{
int i, NUM = 5, arr[5] = {12,13,14,15,16}, sum = 0,prod = 1;
        #pragma omp parallel reduction(+:sum) reduction(*:prod)
        {
          #pragma omp for
          for(i = 0; i < NUM; i++)
          {
                sum += arr[i];
          }
          printf("Sum for thread %d is %d\n",omp_get_thread_num(),sum);
          #pragma omp for
          for(i = 0; i < NUM; i++)
          {
                prod *= arr[i];
          }
          printf("Product for thread %d is %d\n",omp_get_thread_num(),prod);
        }
        printf("Total sum: %d\n",sum);
        printf("Total prod: %d\n",prod);
}
```

EXECUTION:



REMARKS:

Using reduction clause, sum and prod variables that are private to each thread are combined afterwards to return the right result by being a subject to a reduction operation at the end of parallel code block.

SCENARIO II:

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

BRIEF ABOUT THE APPROACH:

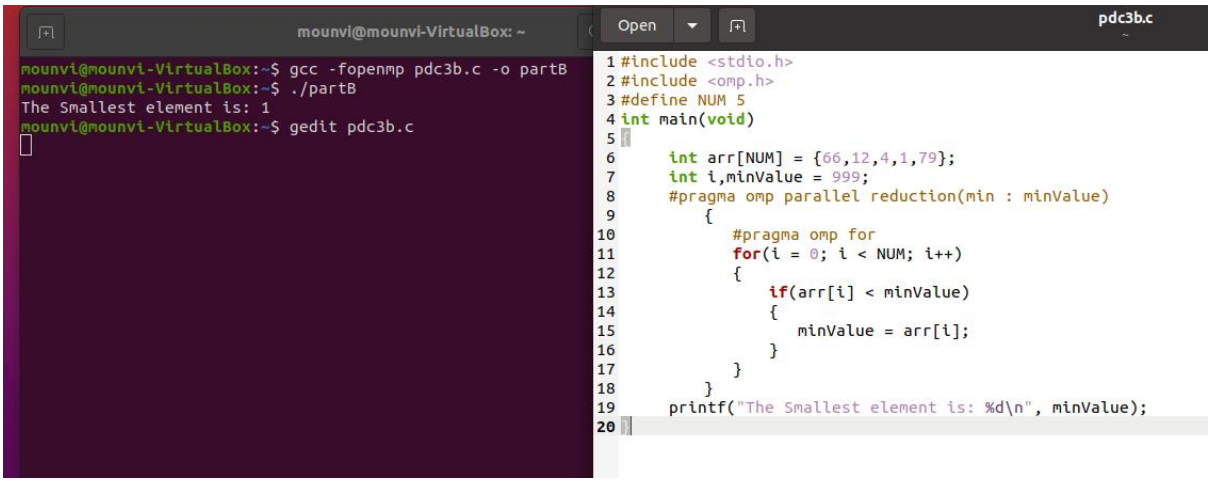
- By using the predefined reduction operation *min* by OpenMP to return the smallest element in a list
- #pragma omp for is used to invoke a parallelized approach to loop through the list using a *for* loop

SOURCE CODE:

```
#include <stdio.h>
#include <omp.h>
#define NUM 5
int main(void)
{
   int arr[NUM] = {66,12,4,1,79};
   int i,minValue = 999;
   #pragma omp parallel reduction(min : minValue)
         {
           #pragma omp for
           for(i = 0; i < NUM; i++)
           {
```

```
                    if(arr[i] < minValue)
                    {
                      minValue = arr[i];
                    }
                }
            }
    printf("The Smallest element is: %d\n", minValue);
}
```

EXECUTION:



REMARKS:

Using the reduction operator min, the smallest value in a list is found in a parallelized approach using a reduction clause in which 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.

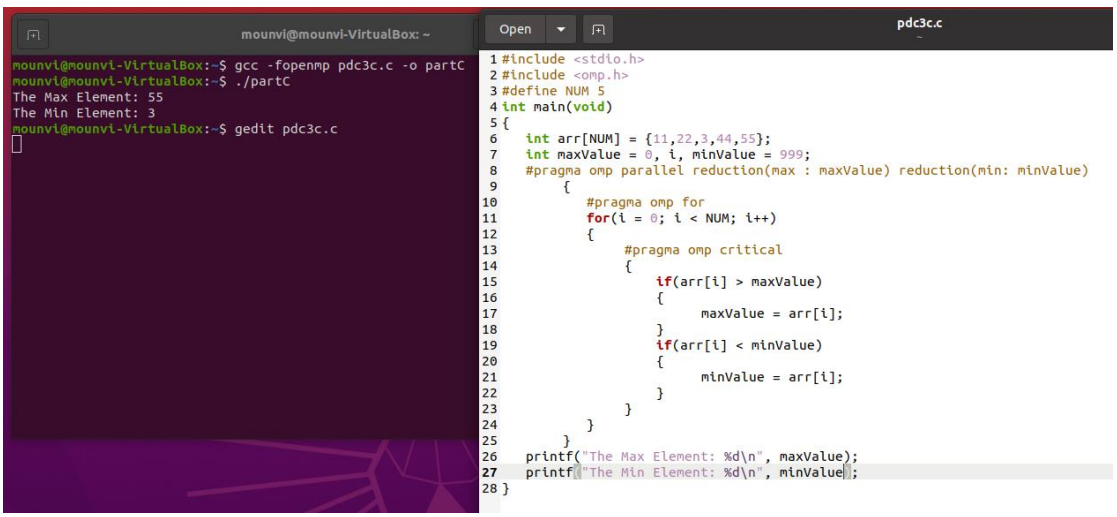
BRIEF ABOUT THE APPROACH:

- *critical* section is used to execute computation on one thread at a time as Max & Min value is easily prone to change by another thread after comparing with Array [Index].
- Execution is similar to the code as in the previous scenario but all comparisons will be required in the loop will be in critical section.

SOURCE CODE:

```
#include <stdio.h>
#include <omp.h>
#define NUM 5
int main(void)
{
  int arr[NUM] = {11,22,3,44,55};
  int maxValue = 0, i, minValue = 999;
  #pragma omp parallel reduction(max : maxValue) reduction(min: minValue)
        {
          #pragma omp for
          for(i = 0; i < NUM; i++)
          {
                #pragma omp critical
                {
                  if(arr[i] > maxValue)
                  {
                        maxValue = arr[i];
                  }
                  if(arr[i] < minValue)
                  {
                        minValue = arr[i];
                  }
                }
          }
        }
  printf("The Max Element: %d\n", maxValue);
  printf("The Min Element: %d\n", minValue);
}
```

EXECUTION:



REMARKS:

The critical section has been explored by finding the min and max elements in a list in a parallelized approach and using a reduction clause and noted that The use of ‘critical’ section demands to execute computation on one thread at a time