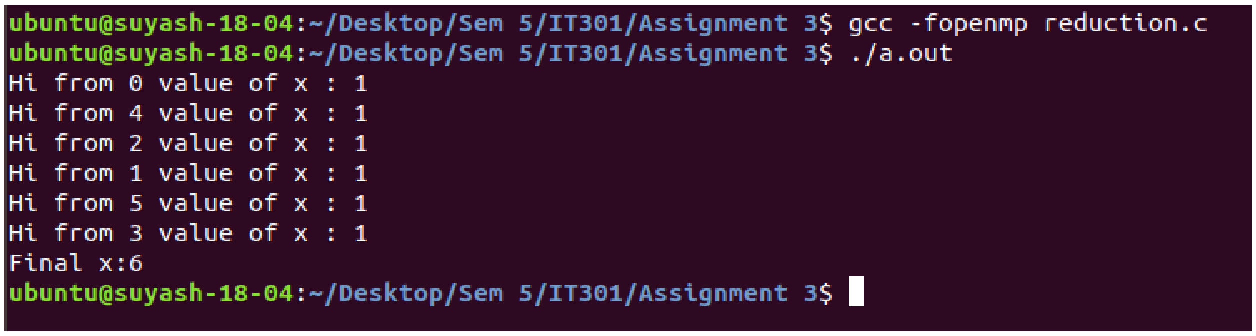
IT301 Assignment 3

NAME: SUYASH CHINTAWAR
ROLL NO.: 191IT109
TOPIC: LAB 3

Note: The codes for problems 1,2 and 3 have not been attached as they were provided in the problem statement itself.

Q1) Demonstration of reduction clause in parallel directive. Write your observation.

SOLUTION:



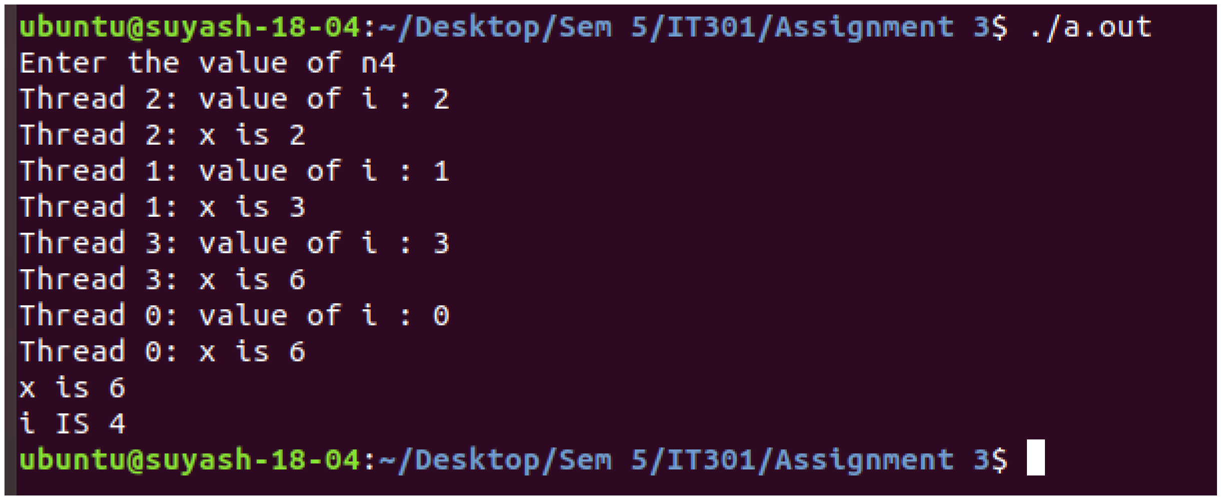
```
ubuntu@suyash-18-04:~/Desktop/Sem 5/IT301/Assignment 3$ gcc -fopenmp reduction.c
ubuntu@suyash-18-04:~/Desktop/Sem 5/IT301/Assignment 3$ ./a.out
Hi from 0 value of x : 1
Hi from 4 value of x : 1
Hi from 2 value of x : 1
Hi from 1 value of x : 1
Hi from 5 value of x : 1
Hi from 3 value of x : 1
Final x:6
ubuntu@suyash-18-04:~/Desktop/Sem 5/IT301/Assignment 3$
```

______________________________**Fig 1.**

Observations: We can see that for each thread, the value of variable ‘x’ after incrementing becomes 1 which infers that the initial value of ‘x’ was zero. Hence, it proves that ‘x’ gets initialized to zero for each thread which in turn implies that ‘x’ becomes private even though none of the default/shared/private clauses has been used. After the complete execution of the pragma construct, the original list item in the reduction clause, i.e. variable ‘x’, is updated with the private copies of each thread by using the ‘+’ operator giving the value of 6 in this case.

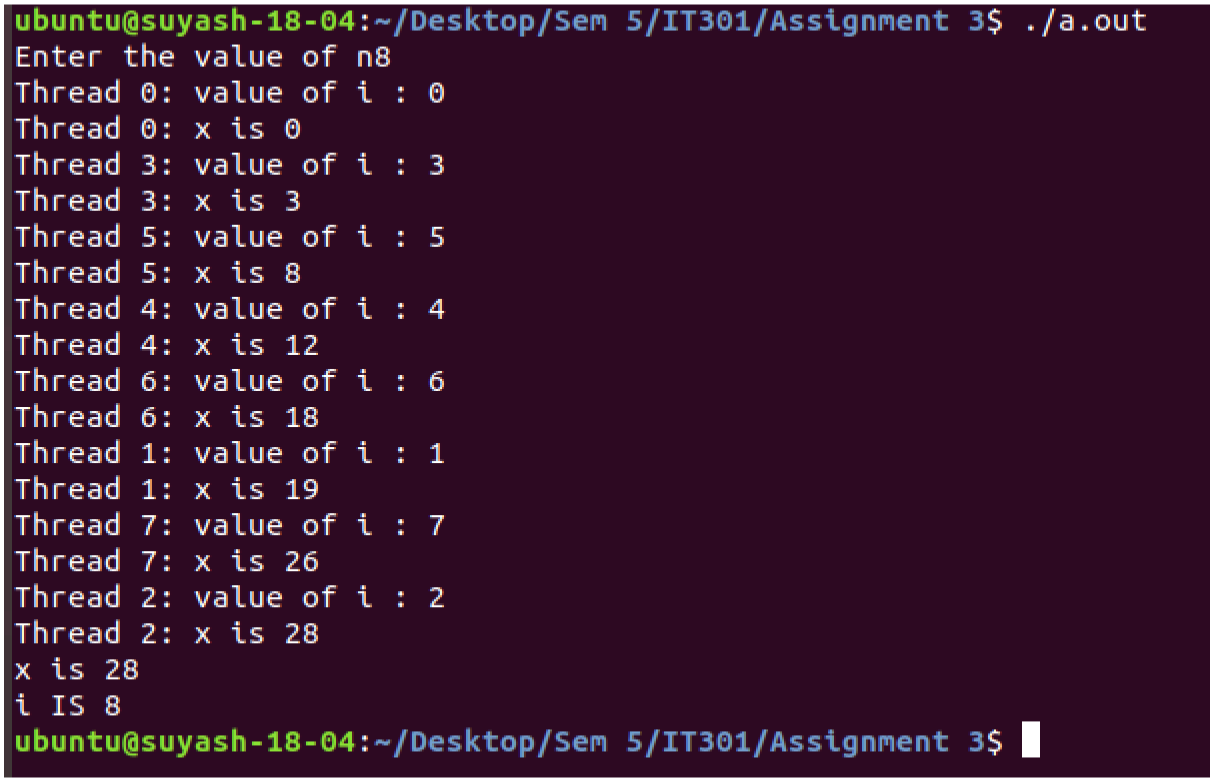
Q2) Demonstration of lastprivate(). Write your observation.

SOLUTION:



```
ubuntu@suyash-18-04:~/Desktop/Sem 5/IT301/Assignment 3$ ./a.out
Enter the value of n4
Thread 2: value of i : 2
Thread 2: x is 2
Thread 1: value of i : 1
Thread 1: x is 3
Thread 3: value of i : 3
Thread 3: x is 6
Thread 0: value of i : 0
Thread 0: x is 6
x is 6
i IS 4
ubuntu@suyash-18-04:~/Desktop/Sem 5/IT301/Assignment 3$
```

Fig 2. num_threads = 4

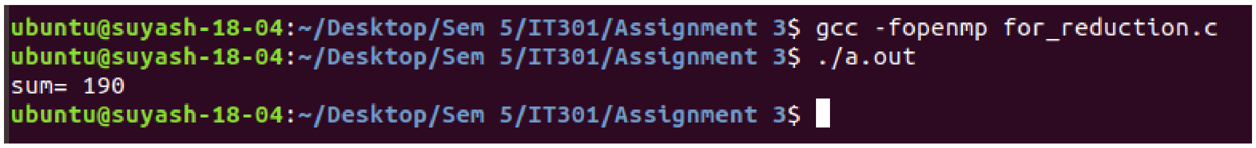


```
ubuntu@suyash-18-04:~/Desktop/Sem 5/IT301/Assignment 3$ ./a.out
Enter the value of n8
Thread 0: value of i : 0
Thread 0: x is 0
Thread 3: value of i : 3
Thread 3: x is 3
Thread 5: value of i : 5
Thread 5: x is 8
Thread 4: value of i : 4
Thread 4: x is 12
Thread 6: value of i : 6
Thread 6: x is 18
Thread 1: value of i : 1
Thread 1: x is 19
Thread 7: value of i : 7
Thread 7: x is 26
Thread 2: value of i : 2
Thread 2: x is 28
x is 28
i IS 8
ubuntu@suyash-18-04:~/Desktop/Sem 5/IT301/Assignment 3$
```

Fig 3. num_threads = 8

Observations: The lastprivate clause retains the last computed value by a thread and assigns it to the master thread after executing the pragma construct. That is, lastprivate in the pragma for construct makes the variable private, but its value gets retained after every iteration. Similarly, the value of the variables also gets retained after coming out of both the pragma constructs and the last computed values gets assigned in the master thread. We can also observe that it is not necessary that the threads execute in ascending order of their thread ids, for eg. in Fig 3, Thread with ID 2 executes at the last and the result obtained after the execution of thread 2 is retained by the master thread in the main program.

Q3) Demonstration of reduction clause with ‘for’.
<u>SOLUTION:</u>



```
ubuntu@suyash-18-04:~/Desktop/Sem 5/IT301/Assignment 3$ gcc -fopenmp for_reduction.c
ubuntu@suyash-18-04:~/Desktop/Sem 5/IT301/Assignment 3$ ./a.out
sum= 190
ubuntu@suyash-18-04:~/Desktop/Sem 5/IT301/Assignment 3$ 
```

Fig 4. reduction clause with ‘for’

<u>Observations:</u> The array ‘a’ is initialized with the values a[i]=i, where 0<=i<20. Hence, the expected output after the summation of all the elements of the array is 190 (which is obtained as in Fig 4). As the number of threads is 6 (given in code in problem statement) and the schedule clause specifies the chunk size as 5, each thread executes 5 iterations of the for loop and their partial sums are added together in the ‘sum’ variable giving us 190 as the final result. Although the number of threads are set to 6, only 4 of them suffice to perform addition of the array (as each thread perform 5 iterations of the for loop).

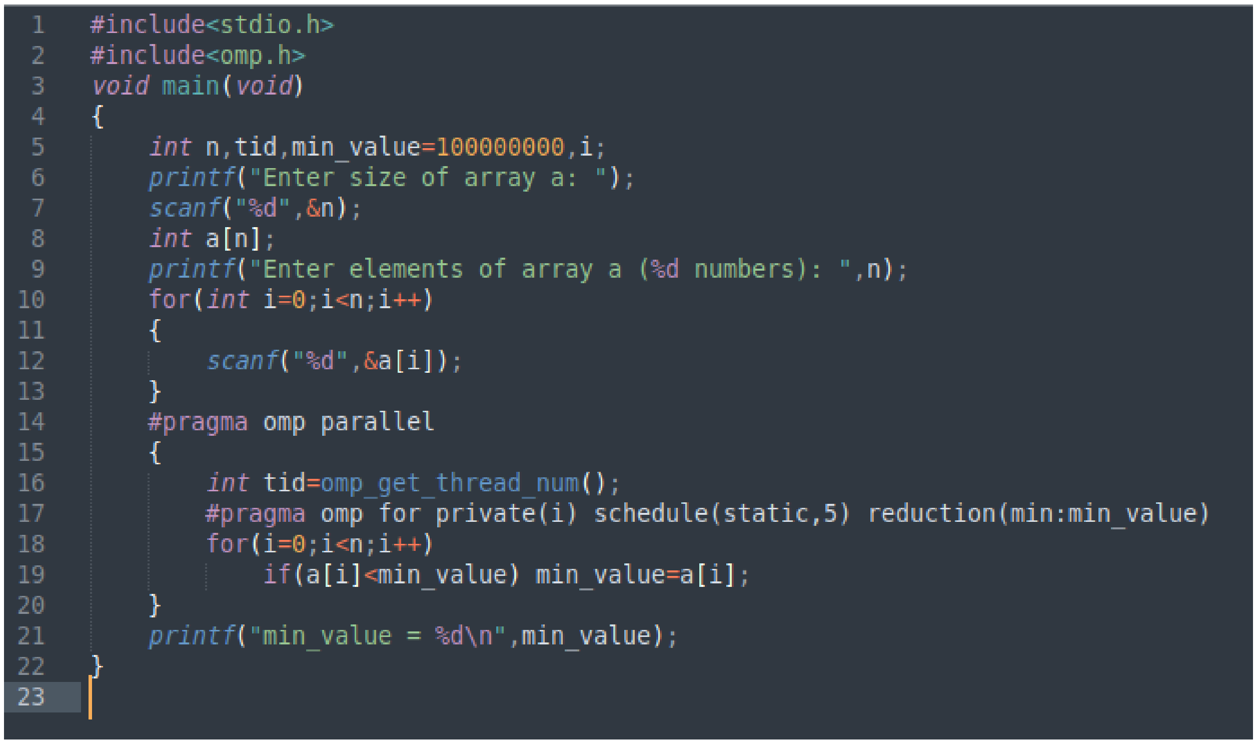
Q4) Programming Exercise

Write a parallel program to find the minimum number in a given array. Use ‘for’ directive for the same along with reduction clause. Write code, execution results and your observation.

SOLUTION:

<u>Code:</u>

(continued on next page...)

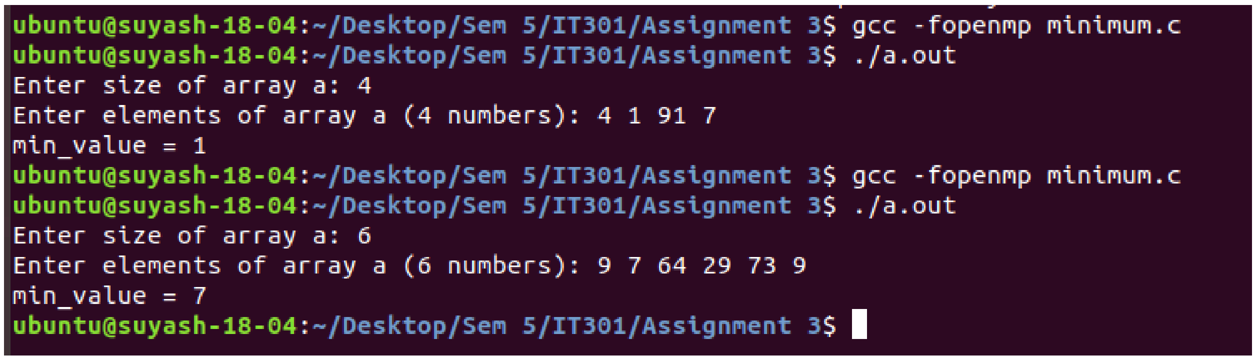


```c
#include<stdio.h>
#include<omp.h>
void main(void)
{
    int n,tid,min_value=100000000,i;
    printf("Enter size of array a: ");
    scanf("%d",&n);
    int a[n];
    printf("Enter elements of array a (%d numbers): ",n);
    for(int i=0;i<n;i++)
    {
        scanf("%d",&a[i]);
    }
    #pragma omp parallel
    {
        int tid=omp_get_thread_num();
        #pragma omp for private(i) schedule(static,5) reduction(min:min_value)
        for(i=0;i<n;i++)
            if(a[i]<min_value) min_value=a[i];
    }
    printf("min_value = %d\n",min_value);
}

```

Fig 5. Code to find minimum in array

Output:



```
ubuntu@suyash-18-04:~/Desktop/Sem 5/IT301/Assignment 3$ gcc -fopenmp minimum.c
ubuntu@suyash-18-04:~/Desktop/Sem 5/IT301/Assignment 3$ ./a.out
Enter size of array a: 4
Enter elements of array a (4 numbers): 4 1 91 7
min_value = 1
ubuntu@suyash-18-04:~/Desktop/Sem 5/IT301/Assignment 3$ gcc -fopenmp minimum.c
ubuntu@suyash-18-04:~/Desktop/Sem 5/IT301/Assignment 3$ ./a.out
Enter size of array a: 6
Enter elements of array a (6 numbers): 9 7 64 29 73 9
min_value = 7
ubuntu@suyash-18-04:~/Desktop/Sem 5/IT301/Assignment 3$
```

Fig 6. Output

(continued on next page...)

Observations:

We use the reduction on the ‘min’ operator present in openmp on the variable. Compute the minimum value as usual by comparing the current minimum value with a[i] and update if a[i]<current_minimum. As the schedule clause divides the work in round robin fashion, every thread gets to do 5 iterations of the for loop. The partial results obtained by each thread, i.e. the minimum values found by each thread in those 5 iterations are then reduced by the ‘min’ operator present in openmp giving us the minimum value in the whole array.

THANK YOU