NAME	JACOB JOHN
REGISTER NO.	16BCE2205
E-MAIL	jacob.john2016@vitstudent.ac.in

LAB ASSESSMENT #3

SCENARIO – 1

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:

The code employs a reduction clause which are a simple mechanism to improve the performance by removing synchronization points and at the price of relaxed consistence of memory view. The reduction clause performs a reduction operation on the variables that appear in the list.

Each thread has a local copy of temp variable. The threads then perform a sum computation as such:
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

The code segmented below also uses a shared variable that runs into a race condition. This is because the threads compete to obtain the temp value. However, the reduction clause prevents it from doing so as clearly shown in the output of the program.

Code:

```c
/*OpenMP program to employ reduction clause to express the reduction of a for loop. */

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

int main()
{
    int sum = 0;
```

```
    int temp = 0;
    int a[10];
    clock_t t;

    printf("Enter a value: \n");
    for(int j = 0; j < 9; j ++)
        scanf("%d",&a[j]);

    omp_set_num_threads(3);
    t = clock();
    #pragma omp parallel for shared(temp, a) reduction(+:sum)
        for (int i = 0; i < 9; i++)
        {
            temp = a[i];
            sum += temp;
            printf("Thread %d is in execution and temp is = %d\n",
omp_get_thread_num(),temp);
            printf("Value of temp is = %d (race condition) and sum = %d\n", temp, sum);
        }
    t = clock() - t;

    printf("Value of sum is: %d\n",sum);
    double time_taken = ((double)t) / CLOCKS_PER_SEC;
    printf("Running the program parallelly took %f seconds to execute \n", time_taken);
}
```

Output:

```
Last login: Thu Aug 16 16:06:21 on ttys000
Jacobs-MacBook-Pro:~ jacobjohn$ cd Codes/
Jacobs-MacBook-Pro:Codes jacobjohn$ cd Parallel-and-Distributed-Computing/
Jacobs-MacBook-Pro:Parallel-and-Distributed-Computing jacobjohn$ gcc-8 -fopenmp reduction_clause.c
Jacobs-MacBook-Pro:Parallel-and-Distributed-Computing jacobjohn$ ./a.out
Enter a value:
1
2
3
45
5
6
7
8
9
Thread 1 is in execution and temp is = 45
Thread 0 is in execution and temp is = 1
Thread 2 is in execution and temp is = 7
Value of temp is = 7 (race condition) and sum = 45
Value of temp is = 7 (race condition) and sum = 1
Value of temp is = 7 (race condition) and sum = 7
Thread 1 is in execution and temp is = 5
Thread 0 is in execution and temp is = 2
Thread 2 is in execution and temp is = 8
Value of temp is = 8 (race condition) and sum = 50
Value of temp is = 8 (race condition) and sum = 3
Value of temp is = 8 (race condition) and sum = 15
Thread 1 is in execution and temp is = 6
Thread 0 is in execution and temp is = 3
Thread 2 is in execution and temp is = 9
Value of temp is = 9 (race condition) and sum = 56
Value of temp is = 9 (race condition) and sum = 6
Value of temp is = 9 (race condition) and sum = 24
Value of sum is: 86
Running the program parallelly took 0.000850 seconds to execute
Jacobs-MacBook-Pro:Parallel-and-Distributed-Computing jacobjohn$
```

Execution

The results produced the race condition as described above:

Thread 1 is in execution and temp is = 6
Thread 0 is in execution and temp is = 3
Thread 2 is in execution and temp is = 9
Value of temp is = 9 (race condition) and sum = 56
Value of temp is = 9 (race condition) and sum = 6
Value of temp is = 9 (race condition) and sum = 24
Value of sum is: 86

SCENARIO – 2

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:

The code employs a reduction clause once again. The code below also uses a min reduction clause which is called as a function. In FORTRAN, an explicit function is provided to perform min reduction but in C this is not the case.

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. It also prevents a race condition that would result in an incorrect minimum value. A Critical Section code segment has been implemented that accesses shared variables and has to be executed as an atomic action. It means that in a group of cooperating processes, at a given point of time, only one process must be executing its critical section. This is done by specifying *#pragma omp parallel*, followed by the shared variables.

Code:

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

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

int min(int x,int y)
{
    if(x<y)
        return x;
    else
        return y;
}

int main()
{
    int min_val = 9999;
    int a[10];
    int temp = 0;
    clock_t t;

    printf("Enter a value: \n");
```

```
    for (int j = 0; j < 9; j++)
        scanf("%d", &a[j]);

    omp_set_num_threads(3);
    t = clock();
    #pragma omp parallel for shared(temp, a) reduction(min:min_val)
    for (int i = 0; i < 9; i++)
    {
        temp = a[i];
        min_val = min(min_val,temp);
        printf("Thread %d is in execution and temp is = %d\n", omp_get_thread_num(), temp);
        printf("Value of temp is = %d (race condition) and min_val = %d\n", temp, min_val);
    }
    t = clock() - t;

    printf("Value of min is: %d\n", min_val);
    double time_taken = ((double)t) / CLOCKS_PER_SEC;
    printf("Running the program parallelly took %f seconds to execute \n", time_taken);
}
```

Output:

```
Jacobs-MacBook-Pro:Parallel-and-Distributed-Computing jacobjohn$ gcc-8 -fopenmp smallest_element_reduction.c
Jacobs-MacBook-Pro:Parallel-and-Distributed-Computing jacobjohn$ ./a.out
Enter a value:
10
2
4
5
69
1
3
4
7
Thread 1 is in execution and temp is = 5
Thread 2 is in execution and temp is = 3
Thread 0 is in execution and temp is = 10
Value of temp is = 3 (race condition) and min_val = 5
Value of temp is = 3 (race condition) and min_val = 3
Value of temp is = 3 (race condition) and min_val = 10
Thread 1 is in execution and temp is = 69
Thread 2 is in execution and temp is = 4
Thread 0 is in execution and temp is = 2
Value of temp is = 2 (race condition) and min_val = 5
Value of temp is = 2 (race condition) and min_val = 3
Value of temp is = 2 (race condition) and min_val = 2
Thread 1 is in execution and temp is = 1
Thread 2 is in execution and temp is = 7
Thread 0 is in execution and temp is = 4
Value of temp is = 4 (race condition) and min_val = 1
Value of temp is = 4 (race condition) and min_val = 3
Value of temp is = 4 (race condition) and min_val = 2
Value of min is: 1
Running the program parallelly took 0.000910 seconds to execute
Jacobs-MacBook-Pro:Parallel-and-Distributed-Computing jacobjohn$
```

Execution

Minimum value has been successfully implemented and the program has produced an output free from race conditions.

Value of temp is = 4 (race condition) and min_val = 1
Value of temp is = 4 (race condition) and min_val = 3
Value of temp is = 4 (race condition) and min_val = 2
Value of min is: 1

SCENARIO – 3

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 employs a reduction clause which are a simple mechanism to improve the performance by removing synchronization points and at the price of relaxed consistence of memory view.

Again, a critical section has been defined to avoid race conditions among the variables.

Code:

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

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

int min(int x,int y)
{
    if(x<y)
        return x;
    else
        return y;
}

int max(int x, int y)
{
    if(x>y)
        return x;
    else
        return y;
}

int main()
{
    int min_val = 9999;
    int max_val = -9999;
    int a[10];
    int temp = 0;
    clock_t t;

    printf("Enter a value: \n");
    for (int j = 0; j < 9; j++)
        scanf("%d", &a[j]);
```

```
    omp_set_num_threads(3);
    t = clock();
    #pragma omp parallel for shared(temp, a) reduction(min:min_val) reduction(max:max_val)
    for (int i = 0; i < 9; i++)
    {
        temp = a[i];
        min_val = min(min_val,temp);
        max_val = max(max_val,temp);
        printf("Thread %d is in execution and temp is = %d\n", omp_get_thread_num(), temp);
        printf("Value of temp is = %d (race condition) and min_val = %d\n", temp, min_val);
        printf("Value of temp is = %d (race condition) and max_val = %d\n", temp, max_val);
    }
    t = clock() - t;

    printf("Value of min is: %d\n", min_val);
    printf("Value of max is: %d\n", max_val);
    double time_taken = ((double)t) / CLOCKS_PER_SEC;
    printf("Running the program parallelly took %f seconds to execute \n", time_taken);
}
```

Output

```
Jacobs-MacBook-Pro:Parallel-and-Distributed-Computing jacobjohn$ gcc-8 -fopenmp min_max_reduction.c
Jacobs-MacBook-Pro:Parallel-and-Distributed-Computing jacobjohn$ ./a.out
Enter a value:
10
2
4
85
139
14
3
6
1
Thread 1 is in execution and temp is = 85
Thread 0 is in execution and temp is = 10
Thread 2 is in execution and temp is = 3
Value of temp is = 10 (race condition) and min_val = 85
Value of temp is = 10 (race condition) and min_val = 10
Value of temp is = 10 (race condition) and min_val = 3
Value of temp is = 10 (race condition) and max_val = 85
Value of temp is = 10 (race condition) and max_val = 10
Value of temp is = 10 (race condition) and max_val = 3
Thread 1 is in execution and temp is = 139
Thread 0 is in execution and temp is = 2
Thread 2 is in execution and temp is = 6
Value of temp is = 6 (race condition) and min_val = 85
Value of temp is = 6 (race condition) and min_val = 2
Value of temp is = 6 (race condition) and min_val = 3
Value of temp is = 6 (race condition) and max_val = 139
Value of temp is = 6 (race condition) and max_val = 10
Value of temp is = 6 (race condition) and max_val = 6
Thread 1 is in execution and temp is = 14
Thread 0 is in execution and temp is = 4
Thread 2 is in execution and temp is = 1
Value of temp is = 1 (race condition) and min_val = 14
Value of temp is = 1 (race condition) and min_val = 2
Value of temp is = 1 (race condition) and min_val = 1
Value of temp is = 1 (race condition) and max_val = 139
Value of temp is = 1 (race condition) and max_val = 10
Value of temp is = 1 (race condition) and max_val = 6
Value of min is: 1
Value of max is: 139
Running the program parallelly took 0.001028 seconds to execute
Jacobs-MacBook-Pro:Parallel-and-Distributed-Computing jacobjohn$
```

Execution

The results reflect the desired result:

Value of temp is = 1 (race condition) and max_val = 139
Value of temp is = 1 (race condition) and max_val = 10
Value of temp is = 1 (race condition) and max_val = 6
Value of min is: 1

Value of max is: 139
Running the program parallelly took 0.001028 seconds to execute