Name: Varun Menon **Course Code: CSE4001**

Reg No: 19BCE1438 **Faculty: Dr. Harini S**

Lab Experiment 4

1. Sample for Reduction

   Code:

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

int main()
{
    omp_set_num_threads(4);
    int a[] = {1,2,3,4};
    int b[] = {1,2,3,4};
    int c[4];
    int sum = 0;

    #pragma omp parallel for reduction(+:sum)
    for(int i=0; i<4; i++)
    {
        c[i] = a[i] + b[i];
        sum += c[i];
        printf("Thread %d :: c = %d\n", omp_get_thread_num(), c[i]);
    }
    printf("Sum of array: %d\n", sum);
    return 0;
}
```

Output:

```
C:\Users\menon\Desktop>gcc -fopenmp reduction.c

C:\Users\menon\Desktop>a.exe
Thread 0 :: c = 2
Thread 2 :: c = 6
Thread 1 :: c = 4
Thread 3 :: c = 8
Sum of array: 20

C:\Users\menon\Desktop>
```

2. Parallel : c = a+b; c= a-b

Code:

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

int main()
{
    omp_set_num_threads(4);
    int a = 5;
    int b = 10;
    int c;

    #pragma omp parallel sections private(c)
    {
        #pragma omp section
        {
            c = a + b;
            printf("Thread %d :: Sum = %d\n", omp_get_thread_num(), c);
        }
        #pragma omp section
        {
            c = a - b;
            printf("Thread %d :: Difference = %d\n",
omp_get_thread_num(), c);
        }
    }
    return 0;
}
```

Output:

```
C:\Users\menon\Desktop>gcc -fopenmp parallel_singlevar.c

C:\Users\menon\Desktop>a.exe
Thread 1 :: Sum = 15
Thread 0 :: Difference = -5

C:\Users\menon\Desktop>
```

3. Parallel : c[i] = a [i]+b[i] ; c [i] =a[i]*b[i] ; c[i]-b[i]

Code:

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

int main()
{
    int a[] = {1,2,3,4};
    int b[] = {9,9,2,3};
    int sum[4], diff[4], mul[4];

    #pragma omp parallel sections
    {
        #pragma omp section
        {
            for(int i=0; i<4; i++)
            {
                sum[i] = a[i] + b[i];
                printf("Sum :: Thread %d :: a[i] = %d; b[i] = %d; Sum =
%d\n", omp_get_thread_num(), a[i], b[i], sum[i]);
            }
        }
        #pragma omp section
        {
            for(int i=0; i<4; i++)
            {
                diff[i] = a[i] - b[i];
                printf("Diff :: Thread %d :: a[i] = %d; b[i] = %d; Diff
= %d\n", omp_get_thread_num(), a[i], b[i], diff[i]);
            }
        }
        #pragma omp section
        {
            for(int i=0; i<4; i++)
            {
                mul[i] = a[i] * b[i];
                printf("Product :: Thread %d :: a[i] = %d; b[i] = %d;
Product = %d\n", omp_get_thread_num(), a[i], b[i], mul[i]);
            }
        }
    }
    return 0;
}
```

Output:

```
C:\Users\menon\Desktop>gcc -fopenmp parallel_array.c

C:\Users\menon\Desktop>a.exe
Sum :: Thread 2 :: a[i] = 1; b[i] = 9; Sum = 10
Sum :: Thread 2 :: a[i] = 2; b[i] = 9; Sum = 11
Sum :: Thread 2 :: a[i] = 3; b[i] = 2; Sum = 5
Sum :: Thread 2 :: a[i] = 4; b[i] = 3; Sum = 7
Product :: Thread 1 :: a[i] = 1; b[i] = 9; Product = 9
Product :: Thread 1 :: a[i] = 2; b[i] = 9; Product = 18
Product :: Thread 1 :: a[i] = 3; b[i] = 2; Product = 6
Product :: Thread 1 :: a[i] = 4; b[i] = 3; Product = 12
Diff :: Thread 0 :: a[i] = 1; b[i] = 9; Diff = -8
Diff :: Thread 0 :: a[i] = 2; b[i] = 9; Diff = -7
Diff :: Thread 0 :: a[i] = 3; b[i] = 2; Diff = 1
Diff :: Thread 0 :: a[i] = 4; b[i] = 3; Diff = 1

C:\Users\menon\Desktop>
```

4. Implement listing of prime numbers < N

Code:

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

int main()
{
    omp_set_num_threads(10);
    int n = 100;
    int c = 2;
    int num_of_div = 0; // number of divisors

    #pragma omp parallel for firstprivate(num_of_div, c, n)
    for(int i=2; i<=n; i++)
    {
        while(c != i)
        {
            if(i%c == 0)
            {
                num_of_div++;
                break;
            }
            c++;
        }
        if(num_of_div == 0)
        {
            printf("Thread %d :: Prime number = %d\n",
omp_get_thread_num(), i);
        }
        num_of_div = 0;
        c = 2;
    }
    return 0;
}
```

Output:

```
C:\Users\menon\Desktop>gcc -fopenmp prime_num.c

C:\Users\menon\Desktop>a.exe
Thread 0 :: Prime number = 2
Thread 0 :: Prime number = 3
Thread 2 :: Prime number = 23
Thread 3 :: Prime number = 37
Thread 4 :: Prime number = 43
Thread 5 :: Prime number = 53
Thread 5 :: Prime number = 59
Thread 5 :: Prime number = 61
Thread 8 :: Prime number = 83
Thread 8 :: Prime number = 89
Thread 1 :: Prime number = 13
Thread 1 :: Prime number = 17
Thread 1 :: Prime number = 19
Thread 3 :: Prime number = 41
Thread 4 :: Prime number = 47
Thread 6 :: Prime number = 67
Thread 7 :: Prime number = 73
Thread 9 :: Prime number = 97
Thread 0 :: Prime number = 5
Thread 0 :: Prime number = 7
Thread 0 :: Prime number = 11
Thread 7 :: Prime number = 79
Thread 2 :: Prime number = 29
Thread 2 :: Prime number = 31
Thread 6 :: Prime number = 71

C:\Users\menon\Desktop>
```

5. Implement "Sudoku solving algorithm (2*2) (16 cells)

Code:

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

int main()
{
    omp_set_num_threads(4);
    int board[4][4] = {{4,1,3,4},{4,3,1,2},{1,2,4,3},{3,4,2,1}}; //
changed for the second output case
    int checksum = 0;
    int flag = 0;
    int sum = 10;

    printf("Puzzle\n");
    for(int i=0; i<4; i++)
    {
        for(int j=0; j<4; j++)
        {
            printf("%d", board[i][j]);
            if (j == 1)
            {
                printf(" || ");
            } else if (j == 3)
            {
                continue;
            } else
            {
                printf(" | ");
            }

        }
        if (i == 1)
        {
            printf("\n--------------");
        }
        printf("\n");
    }

    printf("\n");

    printf("The sudoku is solved correctly if the sum of all the rows
and the columns are equal to exactly 10 each.\nWe check the
same:\n\n");

    #pragma omp parallel sections firstprivate(checksum, flag)
lastprivate(flag)
```

```
    {
        #pragma omp section
        {
            for(int i=0; i<4; i++)
            {
                for(int j=0; j<4; j++)
                {
                    checksum += board[i][j];
                }
                if(checksum == sum)
                {
                    printf("Thread %d :: Row %d equals 10\n",
omp_get_thread_num(), i+1);
                }
                else
                {
                    printf("Thread %d :: Row %d DOESN'T EQUAL 10 --
ERROR SPOTTED\n", omp_get_thread_num(), i+1);
                    flag = 1;
                }
                checksum = 0;
            }
        }

        #pragma omp section
        {
            for(int j=0; j<4; j++)
            {
                for(int i=0; i<4; i++)
                {
                    checksum += board[i][j];
                }
                if(checksum == sum)
                {
                    printf("Thread %d :: Column %d equals 10\n",
omp_get_thread_num(), j+1);
                }
                else
                {
                    printf("Thread %d :: Column %d DOESN'T EQUAL 10 --
ERROR SPOTTED\n", omp_get_thread_num(), j+1);
                    flag = 1;
                }
                checksum = 0;
            }
        }
    }
    printf("\n\n");
```

```
    if(flag == 0)
    {
        printf("Sudoku Solved\n");
    }
    else
    {
        printf("Sudoku Incorrect\n");
    }
    return 0;
}
```

Output:

```
C:\Users\menon\Desktop>gcc -fopenmp sudoku.c

C:\Users\menon\Desktop>a.exe
Puzzle
2 | 1 || 3 | 4
4 | 3 || 1 | 2
-------------
1 | 2 || 4 | 3
3 | 4 || 2 | 1

The sudoku is solved correctly if the sum of all the rows and the columns are equal to exactly 10 each.
We check the same:

Thread 1 :: Row 1 equals 10
Thread 1 :: Row 2 equals 10
Thread 1 :: Row 3 equals 10
Thread 1 :: Row 4 equals 10
Thread 0 :: Column 1 equals 10
Thread 0 :: Column 2 equals 10
Thread 0 :: Column 3 equals 10
Thread 0 :: Column 4 equals 10


Sudoku Solved
```

```
C:\Users\menon\Desktop>gcc -fopenmp sudoku.c

C:\Users\menon\Desktop>a.exe
Puzzle
4 | 1 || 3 | 4
4 | 3 || 1 | 2
-------------
1 | 2 || 4 | 3
3 | 4 || 2 | 1

The sudoku is solved correctly if the sum of all the rows and the columns are equal to exactly 10 each.
We check the same:

Thread 0 :: Row 1 DOESN'T EQUAL 10 -- ERROR SPOTTED
Thread 0 :: Row 2 equals 10
Thread 1 :: Column 1 DOESN'T EQUAL 10 -- ERROR SPOTTED
Thread 1 :: Column 2 equals 10
Thread 0 :: Row 3 equals 10
Thread 0 :: Row 4 equals 10
Thread 1 :: Column 3 equals 10
Thread 1 :: Column 4 equals 10


Sudoku Incorrect
```