**Name: Varun Menon**

**Reg No: 19BCE1438**

**Course Code: CSE4001**

**Faculty: Dr. Harini S**

**Lab Experiment 4**

1. Sample for Reduction

Code:

#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]);

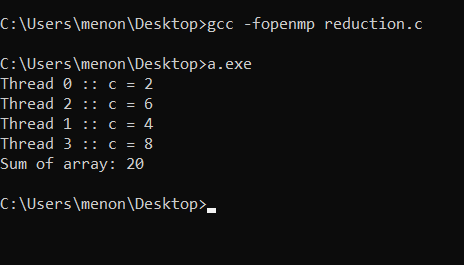
    }

    printf("Sum of array: %d\n", sum);

    return 0;

}

Output:



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

Code:

#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);

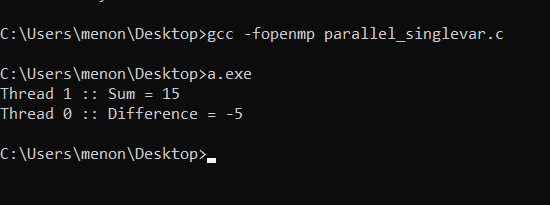
        }

    }

    return 0;

}

Output:



1. 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]);

            }

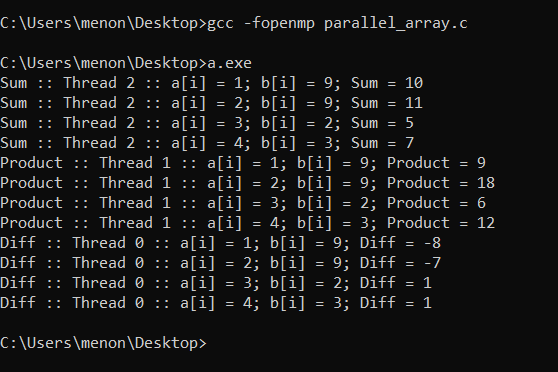
        }

    }

    return 0;

}

Output:



1. 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;

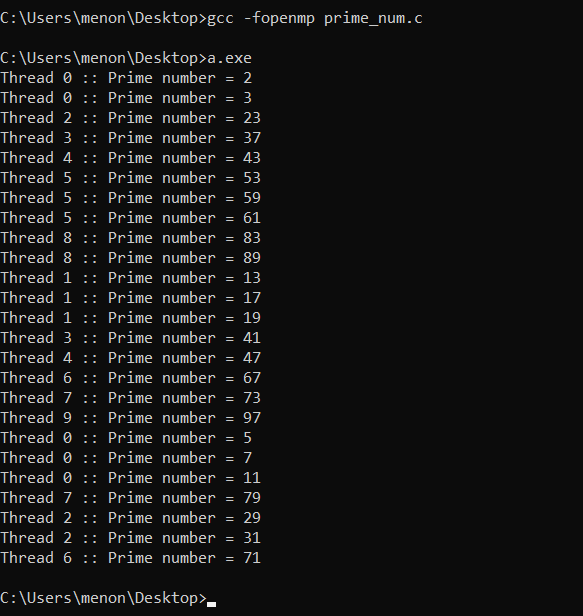
        c = 2;

    }

    return 0;

}

Output:



1. 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:

