<u>CODE</u>

MPI :

```
#include"mpi.h"

#include<stdio.h>

#include<math.h>

#include<iostream>

using namespace std;

int ROW[24][4], pos = 0;

int combination[5200][4], posn = 0;


void permute(int a[4], int l = 0, int r = 3)

{

        int temp;

        if (l == r)

        {

                for (int i = 0; i <= r; i++)

                        ROW[pos][i] = a[i];

                pos++;

        }

        else

                for (int i = l; i <= r; i++)

                {

                        temp = a[l]; a[l] = a[i]; a[i] = temp;//swap a[l] & a[i]

                        permute(a, l + 1, r);

                        temp = a[l]; a[l] = a[i]; a[i] = temp;//swap a[l] & a[i]

                }

}

void nPr(int a[4], int check[4], int l = 0, int r = 3)

{

        int temp, flag, t_arr[4];
```

```
if (l == r)
{
    for (int j = 0; j <= r; j++)
        if (a[j]>check[j])
            return;

    for (int j = 0; j <= r; j++)
        t_arr[j] = a[j];

    flag = 0;
    for (int i = 0; i<posn; i++) // check for already found case in combination array
    {
        flag = 1;
        for (int j = 0; j<4; j++)
        {
            if (t_arr[j] != combination[i][j])
            {
                flag = 0;
                break;
            }
        }
        if (flag == 1) break;
    }

    if (flag == 0)
    {
        for (int j = 0; j <= r; j++)
            combination[posn][j] = t_arr[j];
        posn++;
        }

    }

    else
        for (int i = l; i <= r; i++)
        {
            temp = a[l]; a[l] = a[i]; a[i] = temp;//swap a[l] & a[i]
            nPr(a, check, l + 1, r);
            temp = a[l]; a[l] = a[i]; a[i] = temp;//swap a[l] & a[i]
        }
}
void nCr(int chosen[], int arr[], int index, int r, int start, int end, int check[4])
{
    if (index == r)
    {
        int data[4];
        for (int i = 0; i < r; i++)
            data[i] = arr[chosen[i]];
        nPr(data, check, 0, 3);
        return;
    }
    for (int i = start; i <= end; i++)
    {
        chosen[index] = i;
        nCr(chosen, arr, index + 1, r, i, end, check);
    }
    return;
}
```

```
void main(int argc, char *argv[])
{
    int rank, size;
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    int i, j, k, q, n = 4, l1, r1[4], flag;
    int arr[24], chosen[5], check[4], r = 4, l = 0, m = -1, num;

    int *A = (int *)malloc(sizeof(int)*n*n);
    int *B = (int *)malloc(sizeof(int) * 96 * n);
    int *B_copy = (int *)malloc(sizeof(int) * 96 * n);
    int *C = (int *)malloc(sizeof(int) * 96);
    int *R = (int *)malloc(sizeof(int) * 24 * n);
    int *CMB = (int *)malloc(sizeof(int)*5200*n);
    int *RES = (int *)malloc(sizeof(int)*5200);
    int *RESULT = (int *)malloc(sizeof(int) * 1300);

    //int A[16] = { 0 ,2 ,0 ,4 ,3 ,0 ,0 ,1 ,0 ,0 ,1 ,0 ,0 ,0 ,4 ,3 };

    if (rank == 0)
    {
        cout << "\n\n\t\t\t PCAP PROJECT - MPI (4 Processes)";
        cout << "\n\n  SUDOKU is a logic-based,combinatorial number-placement puzzle. The objective is to fill a 4x4 grid with digits so that each column, each row, ";
        cout << "and each of the four 2x2 subgrids that compose the grid contains all of the digits from 1 to 4. \n\n   The puzzle setter provides a partially completed grid, which ";
        cout << "for a well - posed puzzle has a unique solution completed games are always a type of Latin square with an additional constraint on the contents of individual regions. ";
        cout << "\n\n\n\t Enter the SUDOKU INPUT MATRIX (4x4) -> \n";
        for (i = 0; i < n*n; i++) cin >> A[i];

        int temp[4] = { 1,2,3,4 };
        permute(temp, 0, 3);

        for (i = 0; i<24; i++)
            for (j = 0; j<4; j++)
                R[i * 4 + j] = ROW[i][j];

    }

    MPI_Bcast(A, 16, MPI_INT, 0, MPI_COMM_WORLD); //Broadcasting A
    MPI_Bcast(R, 96, MPI_INT, 0, MPI_COMM_WORLD); //Broadcasting R

    ///////// GENERATING ROWS //////////

    for (i = rank * 4, j = 0; j<4; i++, j++)
        r1[j] = A[i]; //copy row[id] into r1

    l1 = 0; //row index to start writing in B

    for (i = 0; i<24; i++) //possibilities for row 1
    {
        for (j = 0; j<4; j++)
        {
            flag = 1;
            for (k = 0; k<4; k++)
            {
                if (r1[k] != 0 && r1[k] != R[i * 4 + k])
```

```
                    flag = 0;
            }
            if (flag == 0) break;
            C[l1*n + j] = R[i * 4 + j];
        }
        if (flag == 1)
            l1++;
    }
    for (i = l1; i<24; i++)
        for (j = 0; j<4; j++)
            C[i*n + j] = 0;

    MPI_Gather(C, 96, MPI_INT, B, 96, MPI_INT, 0, MPI_COMM_WORLD); //Sending all possible permutations to root  `
    //////////////////////////////////

    if (rank == 0)
    {
        printf("\n\t The Resultant Matrix : \n\t");
        for (int i = 0; i < 384; i++)
        {
            if (B[i] == 0) continue;
            else
            {
                if ((i % 96) == 0) cout << "\n\t ROW " << i / 96 << "\n\t";
                cout << B[i] << " ";
                if ((i + 1) % n == 0)
                    cout << "\n\t";
            }
        }
    /// Combination Array

    for (int i = 0; i < 384; i++)
        B_copy[i] = B[i];

    for (int i = 0; i < 384; i += 4)
    {
        if (i % 96 == 0)
        {
            l = 0; m++;
        }
        if (B[i] != 0) l++;
        check[m] = l;
    }
    num = sizeof(arr) / sizeof(arr[0]);
    for (i = 0; i<24; i++) arr[i] = i + 1;

    nCr(chosen, arr, 0, r, 0, num - 1, check);
    cout << "\n Total Combinations : " << posn << "\n";



    for (i = 0; i < posn; i++)
        for (j = 0; j < 4; j++)
            CMB[i * 4 + j] = combination[i][j];

}

MPI_Bcast(&posn, 1, MPI_INT, 0, MPI_COMM_WORLD); //Broadcasting B
MPI_Bcast(B_copy, 384, MPI_INT, 0, MPI_COMM_WORLD); //Broadcasting B
MPI_Bcast(CMB, posn*4, MPI_INT, 0, MPI_COMM_WORLD); //Broadcasting CMB
```

```
///// GENERATE MATRICES OF ALL COMBINATION AND CHECK IF THEY ARE SUDOKU MATRIX ///////

cout << "\n\t|| Rank : " << rank << "||\n";
for (q = 0; q < 1300; q++)
{
    int id = (rank*1300)+q; //get the index of current thread
    if (id >= posn) { cout << "BREAKING : " << id<<" >= "<<posn; break; }
    int R1[24][4], R2[24][4], R3[24][4], R4[24][4], combo[5200][4], S[4][4];
    int maxm, minm, sum, pro, f, i, j, k;

    RESULT[q] = 0; //initialise RESULT array

    for (i = 0; i < 24; i++)
        for (j = 0; j < 4; j++)
        {
            R1[i][j] = B_copy[(i + 0) * 4 + j];
            R2[i][j] = B_copy[(i + 24) * 4 + j];
            R3[i][j] = B_copy[(i + 48) * 4 + j];
            R4[i][j] = B_copy[(i + 72) * 4 + j];
        }

    for (i = 0; i < posn; i++)
        for (j = 0; j < 4; j++)
            combo[i][j] = CMB[i * 4 + j];

    i = id;

    for (j = 0; j < 4; j++)       //LOAD SUDOKU INTO ARRAY
```

```
        S[0][j] = R1[combo[i][0] - 1][j];
for (j = 0; j < 4; j++)
        S[1][j] = R2[combo[i][1] - 1][j];
for (j = 0; j < 4; j++)
        S[2][j] = R3[combo[i][2] - 1][j];
for (j = 0; j < 4; j++)
        S[3][j] = R4[combo[i][3] - 1][j];

///// CHECK SUDOKU MATRIX /////

//CHECK COLUMN
for (k = 0; k < 4; k++)
{
        f = 1; sum = 0; pro = 1; maxm = 0; minm = 5;
        for (j = 0; j < 4; j++)
        {
                sum += S[j][k];
                pro *= S[j][k];
                if (maxm < S[j][k]) maxm = S[j][k];
                if (minm > S[j][k]) minm = S[j][k];
        }
        if (maxm != 4 || minm != 1 || sum != 10 || pro != 24)
        {
                f = 0; break;
        }
}
if (f == 1) //CHECK BOX
{

        sum = 0; pro = 1; maxm = 0; minm = 5;
```

```
            sum = S[0][0] + S[0][1] + S[1][0] + S[1][1];

            pro = S[0][0] * S[0][1] * S[1][0] * S[1][1];

            minm = (S[0][0] < S[0][1]) ? (S[0][0] < S[1][0]) ? (S[0][0] < S[1][1]) ? S[0][0] : S[1][1] : (S[1][0] < S[1][1]) ? S[1][0] : S[1][1] : (S[0][1] < S[1][0]) ? (S[0][1] < S[1][1]) ? S[0][1] : S[1][1] : (S[1][0] < S[1][1]) ? S[1][0] : S[1][1];

            maxm = (S[0][0] > S[0][1]) ? (S[0][0] > S[1][0]) ? (S[0][0] > S[1][1]) ? S[0][0] : S[1][1] : (S[1][0] > S[1][1]) ? S[1][0] : S[1][1] : (S[0][1] > S[1][0]) ? (S[0][1] > S[1][1]) ? S[0][1] : S[1][1] : (S[1][0] > S[1][1]) ? S[1][0] : S[1][1];


            if (maxm == 4 && minm == 1 && sum == 10 && pro == 24)

            {

                sum = 0; pro = 1; maxm = 0; minm = 5;

                sum = S[2][0] + S[2][1] + S[3][0] + S[3][1];

                pro = S[2][0] * S[2][1] * S[3][0] * S[3][1];

                minm = (S[2][0] < S[2][1]) ? (S[2][0] < S[3][0]) ? (S[2][0] < S[3][1]) ? S[2][0] : S[3][1] : (S[3][0] < S[3][1]) ? S[3][0] : S[3][1] : (S[2][1] < S[3][0]) ? (S[2][1] < S[3][1]) ? S[2][1] : S[3][1] : (S[3][0] < S[3][1]) ? S[3][0] : S[3][1];

                maxm = (S[2][0] > S[2][1]) ? (S[2][0] > S[3][0]) ? (S[2][0] > S[3][1]) ? S[2][0] : S[3][1] : (S[3][0] > S[3][1]) ? S[3][0] : S[3][1] : (S[2][1] > S[3][0]) ? (S[2][1] > S[3][1]) ? S[2][1] : S[3][1] : (S[3][0] > S[3][1]) ? S[3][0] : S[3][1];


                if (maxm == 4 && minm == 1 && sum == 10 && pro == 24)

                {

                    sum = 0; pro = 1; maxm = 0; minm = 5;

                    sum = S[2][2] + S[2][3] + S[3][2] + S[3][3];

                    pro = S[2][2] * S[2][3] * S[3][2] * S[3][3];

                    minm = (S[2][2] < S[2][3]) ? (S[2][2] < S[3][2]) ? (S[2][2] < S[3][3]) ? S[2][2] : S[3][3] : (S[3][2] < S[3][3]) ? S[3][2] : S[3][3] : (S[2][3] < S[3][2]) ? (S[2][3] < S[3][3]) ? S[2][3] : S[3][3] : (S[3][2] < S[3][3]) ? S[3][2] : S[3][3];

                    maxm = (S[2][2] > S[2][3]) ? (S[2][2] > S[3][2]) ? (S[2][2] > S[3][3]) ? S[2][2] : S[3][3] : (S[3][2] > S[3][3]) ? S[3][2] : S[3][3] : (S[2][3] > S[3][2]) ? (S[2][3] > S[3][3]) ? S[2][3] : S[3][3] : (S[3][2] > S[3][3]) ? S[3][2] : S[3][3];


                    if (maxm == 4 && minm == 1 && sum == 10 && pro == 24)
                    {
                        sum = 0; pro = 1; maxm = 0; minm = 5;

                        sum = S[0][2] + S[0][3] + S[1][2] + S[1][3];

                        pro = S[0][2] * S[0][3] * S[1][2] * S[1][3];

                        minm = (S[0][2] < S[0][3]) ? (S[0][2] < S[1][2]) ? (S[0][2] < S[1][3]) ? S[0][2] : S[1][3] : (S[1][2] < S[1][3]) ? S[1][2] : S[1][3] : (S[0][3] < S[1][2]) ? (S[0][3] < S[1][3]) ? S[0][3] : S[1][3] : (S[1][2] < S[1][3]) ? S[1][2] : S[1][3];

                        maxm = (S[0][2] > S[0][3]) ? (S[0][2] > S[1][2]) ? (S[0][2] > S[1][3]) ? S[0][2] : S[1][3] : (S[1][2] > S[1][3]) ? S[1][2] : S[1][3] : (S[0][3] > S[1][2]) ? (S[0][3] > S[1][3]) ? S[0][3] : S[1][3] : (S[1][2] > S[1][3]) ? S[1][2] : S[1][3];


                        if (maxm == 4 && minm == 1 && sum == 10 && pro == 24)

                        {

                            RESULT[q] = 99;

                            printf("\n\n\t** Thread ID : %d -> SUCCESS **", id);

                        }

                    }

                }

            }

        }




    }//end of q loop


    MPI_Gather(RESULT, 1300, MPI_INT, RES, 1300, MPI_INT, 0, MPI_COMM_WORLD); //Sending all possible permutations to root

    ///////////////////////////////////////

    if (rank == 0)

    {


        int R01[24][4], R02[24][4], R03[24][4], R04[24][4];

        for (i = 0; i<24; i++)
```

```
            for (j = 0; j<4; j++)
            {
                R01[i][j] = B_copy[(i + 0) * 4 + j];
                R02[i][j] = B_copy[(i + 24) * 4 + j];
                R03[i][j] = B_copy[(i + 48) * 4 + j];
                R04[i][j] = B_copy[(i + 72) * 4 + j];
            }
        cout << "\n\n\t TOTAL POSSIBLE COMBINATIONS : " << posn;
        for (i = 0; i < posn; i++)
        {
            if (RES[i] == 99)
            {
                cout << "\n\n\t\t -- SUDOKU INPUT -- \n\n\t\t";
                for (j = 0; j < 4; j++)
                {
                    for (k = 0; k < 4; k++)
                        cout << A[j * 4 + k] << " ";
                    cout << "\n\t\t";
                }
                cout << "\n\t\t -- SUDOKU SOLUTION --\n\n\t  THREAD ID : " << i << " | COMBINATION : ";
                cout << " " << combination[i][0] << " " << combination[i][1] << " " << combination[i][2] << " " << combination[i][3] << " |\n\n\t\t";
                for (j = 0; j < 4; j++)
                    cout << R01[combination[i][0] - 1][j] << " "; cout << "\n\t\t";
                for (j = 0; j < 4; j++)
                    cout << R02[combination[i][1] - 1][j] << " "; cout << "\n\t\t";
                for (j = 0; j < 4; j++)
                    cout << R03[combination[i][2] - 1][j] << " "; cout << "\n\t\t";
                for (j = 0; j < 4; j++)
                    cout << R04[combination[i][3] - 1][j] << " "; cout << "\n\t\t";
                cout <<
"\n\n^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^";

            }

        }


    }

    MPI_Finalize();

}

/*

0 2 0 4

3 0 0 1

0 0 1 0

0 0 4 0


1 0 0 0

0 2 0 0

0 0 3 0

0 0 0 0

*/
```

OPENCL :

SUDOKU_PARALLEL_OPENCL.cpp

```
#include <stdio.h>
#include <CL/cl.h>
#include <stdlib.h>
#include <conio.h>
#include <string.h>
#include<iostream>
using namespace std;
#define MAX_SOURCE_SIZE (0x100000)

int ROW[24][4], pos = 0;
int combination[5200][4], posn = 0;


void permute(int a[4], int l = 0, int r = 3)
```

```
{
    int temp;
    if (l == r)
    {
        for (int i = 0; i <= r; i++)
            ROW[pos][i] = a[i];
        pos++;
    }
    else
        for (int i = l; i <= r; i++)
        {
            temp = a[l]; a[l] = a[i]; a[i] = temp;//swap a[l] & a[i]
            permute(a, l + 1, r);
            temp = a[l]; a[l] = a[i]; a[i] = temp;//swap a[l] & a[i]
        }
}
void nPr(int a[4], int check[4], int l = 0, int r = 3)
{
    int temp, flag, t_arr[4];

    if (l == r)
    {
        for (int j = 0; j <= r; j++)
            if (a[j]>check[j])
                return;

        for (int j = 0; j <= r; j++)
            t_arr[j] = a[j];

        flag = 0;
        for (int i = 0; i<posn; i++) // check for already found case in combination array
        {
            flag = 1;
            for (int j = 0; j<4; j++)
            {
                if (t_arr[j] != combination[i][j])
                {
                    flag = 0;
                    break;
                }
            }
            if (flag == 1) break;
        }

        if (flag == 0)
        {
            for (int j = 0; j <= r; j++)
                combination[posn][j] = t_arr[j];
            posn++;
        }

    }

    else
        for (int i = l; i <= r; i++)
        {
            temp = a[l]; a[l] = a[i]; a[i] = temp;//swap a[l] & a[i]
            nPr(a, check, l + 1, r);
            temp = a[l]; a[l] = a[i]; a[i] = temp;//swap a[l] & a[i]
        }
}
void nCr(int chosen[], int arr[], int index, int r, int start, int end, int check[4])
{
    if (index == r)
    {
        int data[4];
        for (int i = 0; i < r; i++)
            data[i] = arr[chosen[i]];
        nPr(data, check, 0, 3);
        return;
    }
    for (int i = start; i <= end; i++)
    {
        chosen[index] = i;
        nCr(chosen, arr, index + 1, r, i, end, check);
    }
    return;
}


int main(void)
{
    int i,j,k,n=4;
    cout << "\n\n\t\t\t PCAP PROJECT - OpenCL ";
    cout << "\n\n  SUDOKU is a logic-based,combinatorial number-placement puzzle. The objective is to fill a 4x4 grid with digits so that each column, each row, ";
    cout << "and each of the four 2x2 subgrids that compose the grid contains all of the digits from 1 to 4. \n\n   The puzzle setter provides a partially completed grid, which ";
    cout << "for a well - posed puzzle has a unique solution completed games are always a type of Latin square with an additional constraint on the contents of individual regions. ";

    cout << "\n\n\n\t Enter the SUDOKU INPUT MATRIX -> \n";
    int *A = (int *)malloc(sizeof(int)*n*n);
    int *B = (int *)malloc(sizeof(int)*96*n);
    int *R = (int *)malloc(sizeof(int)*24*n);
    //int A[16] = { 0 ,2 ,0 ,4 ,3 ,0 ,0 ,1 ,0 ,0 ,1 ,0 ,0 ,0 ,4 ,3 };
    for (i = 0; i < n*n; i++) cin >> A[i];

    int temp[4] = { 1,2,3,4 };
    permute(temp, 0, 3);

    for (i = 0; i<24; i++)
        for (j = 0; j<4; j++)
            R[i * 4 + j] = ROW[i][j];

    FILE *fp1;
    char *source_str_1;
    size_t source_size_1;
    fp1 = fopen("SUDOKU_PARALLEL_OPENCL.cl", "r");
    source_str_1 = (char*)malloc(MAX_SOURCE_SIZE);
    source_size_1 = fread(source_str_1, 1, MAX_SOURCE_SIZE, fp1);
    fclose(fp1);

    FILE *fp2;
    char *source_str_2;
    size_t source_size_2;
    fp2 = fopen("SUDOKU_PARALLEL.cl", "r");
    source_str_2 = (char*)malloc(MAX_SOURCE_SIZE);
    source_size_2 = fread(source_str_2, 1, MAX_SOURCE_SIZE, fp2);
    fclose(fp2);

    cl_platform_id platform_id = NULL;
    cl_device_id device_id = NULL;
    cl_uint ret_num_devices;
    cl_uint ret_num_platforms;

    cl_int ret = clGetPlatformIDs(1, &platform_id, &ret_num_platforms);
    ret = clGetDeviceIDs(platform_id, CL_DEVICE_TYPE_CPU, 1, &device_id, &ret_num_devices);
    cl_context context = clCreateContext(NULL, 1, &device_id, NULL, NULL, &ret);

    cl_command_queue command_queue = clCreateCommandQueue(context, device_id,
CL_QUEUE_PROFILING_ENABLE, &ret);


    cl_mem a_mem_obj = clCreateBuffer(context, CL_MEM_READ_ONLY, n*n * sizeof(int), NULL, &ret);
    cl_mem b_mem_obj = clCreateBuffer(context, CL_MEM_READ_WRITE, 96*n * sizeof(int), NULL, &ret);
    cl_mem r_mem_obj = clCreateBuffer(context, CL_MEM_READ_ONLY, 24*n * sizeof(int), NULL, &ret);

    ret = clEnqueueWriteBuffer(command_queue, a_mem_obj, CL_TRUE, 0, n*n * sizeof(int), A, 0, NULL,
NULL);
    ret = clEnqueueWriteBuffer(command_queue, r_mem_obj, CL_TRUE, 0, 24*n * sizeof(int), R, 0, NULL,
NULL);
```

```
    cl_program program_1 = clCreateProgramWithSource(context, 1, (const char **)&source_str_1,
(const size_t *)&source_size_1, &ret);
    ret = clBuildProgram(program_1, 1, &device_id, NULL, NULL, NULL);
    cl_program program_2 = clCreateProgramWithSource(context, 1, (const char **)&source_str_2,
(const size_t *)&source_size_2, &ret);
    ret = clBuildProgram(program_2, 1, &device_id, NULL, NULL, NULL);


    cl_kernel kernel_1 = clCreateKernel(program_1, "SUDOKU", &ret);
    cl_kernel kernel_2 = clCreateKernel(program_2, "LOADCHECK", &ret);

    //Set the arguments of the kernel
    ret = clSetKernelArg(kernel_1, 0, sizeof(cl_mem), (void *)&a_mem_obj);
    ret = clSetKernelArg(kernel_1, 1, sizeof(cl_mem), (void *)&b_mem_obj);
    ret = clSetKernelArg(kernel_1, 2, sizeof(cl_mem), (void *)&r_mem_obj);
    ret = clSetKernelArg(kernel_1, 3, sizeof(cl_mem), (void *)&n);


    size_t global_item_size[2] = { n,n };
    size_t local_item_size[2] = { 1,4 };

    //Execute the kernel on the device
    cl_event event;
    ret = clEnqueueNDRangeKernel(command_queue, kernel_1, 2, NULL, global_item_size,
local_item_size, 0, NULL, &event);


    ret = clEnqueueReadBuffer(command_queue, b_mem_obj, CL_TRUE, 0, 96*n*sizeof(int), B, 0, NULL,
NULL);
    printf("\n\t The Resultant Matrix : \n\t");
    for (int i = 0; i < 384; i++)
    {
        if (B[i] == 0) continue;
        else
        {
            if ((i % 96) == 0) cout << "\n\t ROW " << i / 96 << "\n\t";
            cout << B[i] << " ";
            if ((i + 1) % n == 0)
                cout << "\n\t";
        }
    }
    /// Combination Array
    int arr[24], chosen[5], check[4], r = 4, l = 0, m = -1, num;

    for (int i = 0; i < 384; i += 4)
    {
        if (i % 96 == 0)
        {
            l = 0; m++;
        }
        if (B[i] != 0) l++;
        check[m] = l;
    }
    num = sizeof(arr) / sizeof(arr[0]);
    for (i = 0; i<24; i++) arr[i] = i + 1;

    nCr(chosen, arr, 0, r, 0, num - 1, check);
    cout << "\n Total Combinations : " << posn<<"\n";

    ////////////////

        int *CMB = (int *)malloc(sizeof(int)*posn*n);
        int *RES = (int *)malloc(sizeof(int)*posn);

        for (i = 0; i < posn; i++)
            for (j = 0; j < 4; j++)
                CMB[i * 4 + j] = combination[i][j];

        cl_mem combo_mem_obj = clCreateBuffer(context, CL_MEM_READ_WRITE, posn*n *
sizeof(int), NULL, &ret);
        cl_mem res_mem_obj = clCreateBuffer(context, CL_MEM_READ_WRITE, posn * sizeof(int),
NULL, &ret);

        ret = clEnqueueWriteBuffer(command_queue, b_mem_obj, CL_TRUE, 0, 96 * n * sizeof(int), B,
0, NULL, NULL);
        ret = clEnqueueWriteBuffer(command_queue, combo_mem_obj, CL_TRUE, 0, posn*n *
sizeof(int), CMB, 0, NULL, NULL);

        ret = clSetKernelArg(kernel_2, 0, sizeof(cl_mem), (void *)&b_mem_obj);
        ret = clSetKernelArg(kernel_2, 1, sizeof(cl_mem), (void *)&combo_mem_obj);
        ret = clSetKernelArg(kernel_2, 2, sizeof(cl_mem), (void *)&res_mem_obj);
        ret = clSetKernelArg(kernel_2, 3, sizeof(cl_mem), (void *)&posn);

        size_t global_item_size_k2 = posn;
        size_t local_item_size_k2 = 1;
        cout <<
"\n^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^\n";
        ret = clEnqueueNDRangeKernel(command_queue, kernel_2, 1, NULL, &global_item_size_k2,
&local_item_size_k2, 0, NULL, NULL);

        ret = clEnqueueReadBuffer(command_queue, res_mem_obj, CL_TRUE, 0, posn * sizeof(int),
RES, 0, NULL, NULL);
        cout <<
"\n^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^";

        // SOLUTION DISPLAY
        int R01[24][4], R02[24][4], R03[24][4], R04[24][4];
        for (i = 0; i<24; i++)
            for (j = 0; j<4; j++)
            {
                R01[i][j] = B[(i + 0) * 4 + j];
                R02[i][j] = B[(i + 24) * 4 + j];
                R03[i][j] = B[(i + 48) * 4 + j];
                R04[i][j] = B[(i + 72) * 4 + j];
            }
        cout << "\n\n\t TOTAL POSSIBLE COMBINATIONS : "<<posn;
        for (i = 0; i < posn; i++)
        {
            if (RES[i] == 99)
            {
                cout << "\n\n\t\t -- SUDOKU INPUT -- \n\n\t\t";
                for (j = 0; j < 4; j++)
                {
                    for (k = 0; k < 4; k++)
                        cout << A[j * 4 + k]<<" ";
                    cout << "\n\t\t";
                }
                cout << "\n\t\t -- SUDOKU SOLUTION --\n\n\t  KERNEL ID : " << i << " |
COMBINATION : ";
                cout << " "<<combination[i][0] << " " << combination[i][1] << " " <<
combination[i][2] << " " << combination[i][3] << " |\n\n\t\t";
                for (j = 0; j < 4; j++)
                    cout << R01[combination[i][0] - 1][j] << " "; cout << "\n\t\t";
                for (j = 0; j < 4; j++)
                    cout << R02[combination[i][1] - 1][j] << " "; cout << "\n\t\t";
                for (j = 0; j < 4; j++)
                    cout << R03[combination[i][2] - 1][j] << " "; cout << "\n\t\t";
                for (j = 0; j < 4; j++)
                    cout << R04[combination[i][3] - 1][j] << " "; cout << "\n\t\t";
                cout <<
"\n\n^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^";
            }
        }




    getchar();
    getchar();
    getchar();
    /////////////////

    ret = clFlush(command_queue);
    ret = clReleaseKernel(kernel_1);
    ret = clReleaseProgram(program_1);
    ret = clReleaseKernel(kernel_2);
    ret = clReleaseProgram(program_2);
    ret = clReleaseMemObject(a_mem_obj);
    ret = clReleaseMemObject(b_mem_obj);
    ret = clReleaseMemObject(r_mem_obj);
    ret = clReleaseMemObject(combo_mem_obj);
    ret = clReleaseMemObject(res_mem_obj);
    ret = clReleaseCommandQueue(command_queue);
    ret = clReleaseContext(context);

    return 0;

}
```

SUDOKU_PARALLEL.cl

```
__kernel void LOADCHECK(__global int *B,__global int *CMB,__global int *RES,int posn)
{
    int id = get_global_id(0); //get the index of current thread

    int R1[24][4], R2[24][4], R3[24][4], R4[24][4], combo[5200][4], S[4][4] ;
    int maxm,minm,sum,pro,f,i,j,k;

    for(i=0;i<24;i++)
    for(j=0;j<4;j++)
    {
        R1[i][j]=B[(i+0) *4 + j];
        R2[i][j]=B[(i+24)*4 + j];
        R3[i][j]=B[(i+48)*4 + j];
        R4[i][j]=B[(i+72)*4 + j];
    }

    for(i=0;i<posn;i++)
    for(j=0;j<4;j++)
    combo[i][j]=CMB[i*4+j];

    i=id;

for(j=0;j<4;j++)     //LOAD SUDOKU INTO ARRAY
        S[0][j]=R1[combo[i][0]-1] [j];
        for(j=0;j<4;j++)
        S[1][j]=R2[combo[i][1]-1] [j];
        for(j=0;j<4;j++)
        S[2][j]=R3[combo[i][2]-1] [j];
        for(j=0;j<4;j++)
        S[3][j]=R4[combo[i][3]-1] [j];

        ///// CHECK SUDOKU MATRIX /////
```

```
                //CHECK COLUMN
                for(k=0;k<4;k++)
                {
                        f=1;sum=0;pro=1;maxm=0;minm=5;
                        for(j=0;j<4;j++)
                        {
                                sum+=S[j][k];
                                pro*=S[j][k];
                                if(maxm<S[j][k]) maxm=S[j][k];
                                if(minm>S[j][k]) minm=S[j][k];
                        }
                        if(maxm!=4 || minm!=1 ||sum!=10 || pro!=24 )
                        {
                                f=0;break;
                        }
          }
          if(f==1) //CHECK BOX
          {

                sum=0;pro=1;maxm=0;minm=5;
                        sum=S[0][0]+S[0][1]+S[1][0]+S[1][1];
                pro=S[0][0]*S[0][1]*S[1][0]*S[1][1];

        minm=(S[0][0]<S[0][1])?(S[0][0]<S[1][0])?(S[0][0]<S[1][1])?S[0][0]:S[1][1]:(S[1][0]<S[1][1])?S[1][0]:S[1][1]:(S[0][1]<S[1][0])?(S[0][1]<S[1][1])?S[0][1]:S[1][1]:(S[1][0]<S[1][1])?S[1][0]:S[1][1];

        maxm=(S[0][0]>S[0][1])?(S[0][0]>S[1][0])?(S[0][0]>S[1][1])?S[0][0]:S[1][1]:(S[1][0]>S[1][1])?S[1][0]:S[1][1]:(S[0][1]>S[1][0])?(S[0][1]>S[1][1])?S[0][1]:S[1][1]:(S[1][0]>S[1][1])?S[1][0]:S[1][1];

                if(maxm==4 && minm==1 &&sum==10 && pro==24 )
                        {
                                sum=0;pro=1;maxm=0;minm=5;
                                sum=S[2][0]+S[2][1]+S[3][0]+S[3][1];
                        pro=S[2][0]*S[2][1]*S[3][0]*S[3][1];

        minm=(S[2][0]<S[2][1])?(S[2][0]<S[3][0])?(S[2][0]<S[3][1])?S[2][0]:S[3][1]:(S[3][0]<S[3][1])?S[3][0]:S[3][1]:(S[2][1]<S[3][0])?(S[2][1]<S[3][1])?S[2][1]:S[3][1]:(S[3][0]<S[3][1])?S[3][0]:S[3][1];

        maxm=(S[2][0]>S[2][1])?(S[2][0]>S[3][0])?(S[2][0]>S[3][1])?S[2][0]:S[3][1]:(S[3][0]>S[3][1])?S[3][0]:S[3][1]:(S[2][1]>S[3][0])?(S[2][1]>S[3][1])?S[2][1]:S[3][1]:(S[3][0]>S[3][1])?S[3][0]:S[3][1];

               if(maxm==4 && minm==1 &&sum==10 && pro==24 )
                                {
                                        sum=0;pro=1;maxm=0;minm=5;
                                        sum=S[2][2]+S[2][3]+S[3][2]+S[3][3];
                                pro=S[2][2]*S[2][3]*S[3][2]*S[3][3];
        minm=(S[2][2]<S[2][3])?(S[2][2]<S[3][2])?(S[2][2]<S[3][3])?S[2][2]:S[3][3]:(S[3][2]<S[3][3])?S[3][2]:S[3][
3]:(S[2][3]<S[3][2])?(S[2][3]<S[3][3])?S[2][3]:S[3][3]:(S[3][2]<S[3][3])?S[3][2]:S[3][3];

        maxm=(S[2][2]>S[2][3])?(S[2][2]>S[3][2])?(S[2][2]>S[3][3])?S[2][2]:S[3][3]:(S[3][2]>S[3][3])?S[3][2]:S[3]
[3]:(S[2][3]>S[3][2])?(S[2][3]>S[3][3])?S[2][3]:S[3][3]:(S[3][2]>S[3][3])?S[3][2]:S[3][3];

                  if(maxm==4 && minm==1 &&sum==10 && pro==24 )
                                          {
                                                  sum=0;pro=1;maxm=0;minm=5;
                                                  sum=S[0][2]+S[0][3]+S[1][2]+S[1][3];
                                          pro=S[0][2]*S[0][3]*S[1][2]*S[1][3];

        minm=(S[0][2]<S[0][3])?(S[0][2]<S[1][2])?(S[0][2]<S[1][3])?S[0][2]:S[1][3]:(S[1][2]<S[1][3])?S[1][2]:S[1][
3]:(S[0][3]<S[1][2])?(S[0][3]<S[1][3])?S[0][3]:S[1][3]:(S[1][2]<S[1][3])?S[1][2]:S[1][3];

        maxm=(S[0][2]>S[0][3])?(S[0][2]>S[1][2])?(S[0][2]>S[1][3])?S[0][2]:S[1][3]:(S[1][2]>S[1][3])?S[1][2]:S[1]
[3]:(S[0][3]>S[1][2])?(S[0][3]>S[1][3])?S[0][3]:S[1][3]:(S[1][2]>S[1][3])?S[1][2]:S[1][3];

                    if(maxm==4 && minm==1 &&sum==10 && pro==24 )
                                                  {
                                                          RES[id]=99;
                                                          printf("\n\n\t** Kernel ID : %d -> SUCCESS **",id);
                                                  }
                                  }
                          }
          }
        }

///// DISPLAY /////
printf("\n\t ID : %d | COMBINATION : %d %d %d
%d\n",id,combo[id][0],combo[id][1],combo[id][2],combo[id][3]);
printf("\n\t     --- SUDOKU Matrix ---\n\t\t");

for(i=0;i<4;i++)
{
        for(j=0;j<4;j++)
        printf("%d ",S[i][j]);
        printf("\n\t\t");
}
        for(i=0;i<24;i++) //Fake printing R1 R2 R3 R4 - Some deep mythological or spiritual error
{
  if(R1[i][0]==0) break;
        for(j=0;j<4;j++)
        printf(" ",R1[i][j]);

}
        for(i=0;i<24;i++)
```

```
{
   if(R2[i][0]==0) break;
        for(j=0;j<4;j++)
        printf(" ",R2[i][j]);

}
        for(i=0;i<24;i++)
{
   if(R3[i][0]==0) break;
        for(j=0;j<4;j++)
        printf(" ",R3[i][j]);
}
        for(i=0;i<24;i++)
{
   if(R4[i][0]==0) break;
        for(j=0;j<4;j++)
        printf(" ",R4[i][j]);
}




 }
```

SUDOKU_PARALLEL_OPENCL.cl

```
__kernel void SUDOKU(__global int *A,__global int *B,__global int *R,int n)
{
        int id = get_global_id(0); //get the row no of SUDOKU I/P
        int temp[4] = {1,2,3,4}, l1,i,j,k,r1[4],flag;

        for( i=id*4 , j=0 ; j<4 ; i++,j++)
        r1[j]=A[i]; //copy row[id] into r1

        l1=id*24; //row index to start writing in B

   for(i=0;i<24;i++) //possibilities for row 1
   {
        for(j=0;j<4;j++)
        {
                flag=1;
                for(k=0;k<4;k++)
                {
                        if(r1[k]!=0 && r1[k]!=R[i*4+k])
                        flag=0;
                        }
                        if(flag==0) break;
```

```
            B[l1*n + j]=R[i*4+j];
        }
        if(flag==1)
        l1++;
    }
            for(i=l1;i<id*24+24;i++)
            for(j=0;j<4;j++)
            B[i*n + j]=0;

}
```