**Name: Varun Menon**

**Reg No: 19BCE1438**

**Course Code: CSE4001**

**Faculty: Dr. Harini S**

**Lab Experiment 9**

1. Circuit Satisfiability

Code:

#include<mpi.h>

#include<stdio.h>

#include<math.h>

void generate\_tt(int);

void check\_ckt(int,int);

int a[16];

int main()

{

    MPI\_Init(NULL,NULL);

    int i, n=4;

    int id;

    int p;

    MPI\_Comm\_rank(MPI\_COMM\_WORLD, &id);

    MPI\_Comm\_size(MPI\_COMM\_WORLD, &p);

    int m = pow(2,n);

    generate\_tt(m);

    for(int i=id;i<m;i+=p)

    {

        check\_ckt(id,i);

    }

    MPI\_Finalize();

    return 0;

}

void generate\_tt(int m)

{

    for(int i=0;i<m;i++)

    {

        if(i%2==0)

        {

            a[i]=1;

        }

        else

        {

            a[i]=0;

        }

    }

}

void check\_ckt(int id,int i)

{

    if(a[i]==1)

    {

    printf("\n[P%d] Row %d : satisfied circuit\n",id,i);

    }

    else

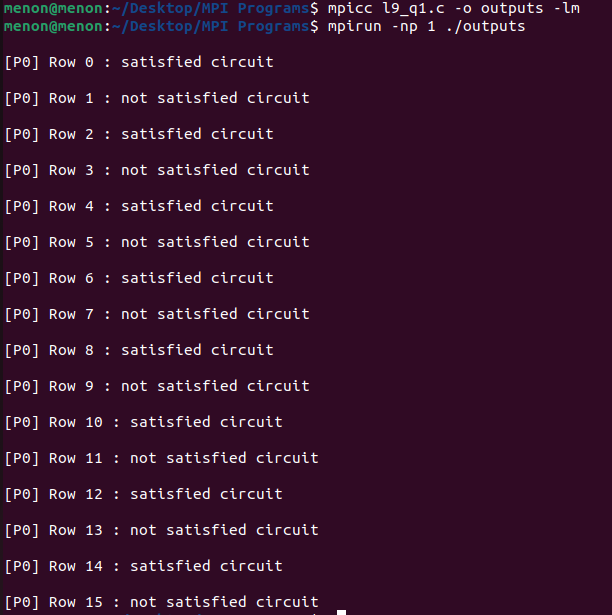
    {

    printf("\n[P%d] Row %d : not satisfied circuit\n",id,i);

    }

}

Output:



1. Number of solutions in circuit satisfiability

Code:

#include<mpi.h>

#include<stdio.h>

#include<math.h>

void generate\_tt(int);

void check\_ckt(int,int);

int a[16], count;

int main()

{

    MPI\_Init(NULL,NULL);

    int i, n=4;

    int id;

    int p;

    MPI\_Comm\_rank(MPI\_COMM\_WORLD, &id);

    MPI\_Comm\_size(MPI\_COMM\_WORLD, &p);

    int m = pow(2,n);

    generate\_tt(m);

    for(int i=id;i<m;i+=p)

    {

        check\_ckt(id,i);

    }

    printf("Number of Solutions: %d\n", count);

    MPI\_Finalize();

    return 0;

}

void generate\_tt(int m)

{

    for(int i=0;i<m;i++)

    {

        if(i%2==0)

        {

            a[i]=1;

        }

        else

        {

            a[i]=0;

        }

    }

}

void check\_ckt(int id,int i)

{

    if(a[i]==1)

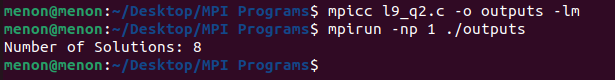
    {

        count++;

    }

}

Output:



1. Adding a count to all values of a matrix with size n\*n

Code:

#include <mpi.h>

#include <stdio.h>

int main(int argc, char\*\* argv) {

    int p, id;

    int n = 3, counter = 1;

    int mat[3][3] = {{9,8,7},

        {6,5,4},

        {3,2,1}};

    MPI\_Init(NULL, NULL);

    MPI\_Comm\_size(MPI\_COMM\_WORLD, &p);

    MPI\_Comm\_rank(MPI\_COMM\_WORLD, &id);

    for(int x = id; x < n\*n; x += p)

    {

        int i = x / n, j = x % n;

        mat[i][j] += counter;

        printf("Process %d : mat[%d][%d] = %d\n", id, i, j, mat[i][j]);

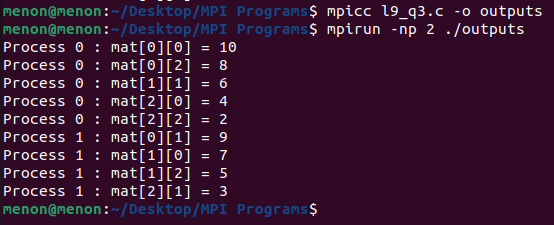
    }

    MPI\_Finalize();

    return 0;

}

Output:



1. Find Max of 'n' no's

Code:

#include <mpi.h>

#include <stdio.h>

#include<math.h>

#include<limits.h>

int main(int argc, char\*\* argv) {

    int p, id, n = 10;

    int arr[10] = {2, 5, 6, 9, 4, 7, 1, 8, 2, 1};

    MPI\_Request request;

    MPI\_Status status;

    MPI\_Init(NULL, NULL);

    MPI\_Comm\_size(MPI\_COMM\_WORLD, &p);

    MPI\_Comm\_rank(MPI\_COMM\_WORLD, &id);

    int m = INT\_MIN;

    for(int i = id; i < n; i += p)

    {

        if(arr[i] > m)

        {

            m = arr[i];

        }

    }

    if(id != 0)

    {

        printf("Process %d; Sending : Max = %d\n", id, m);

        MPI\_Isend(&m, 1, MPI\_INT, 0, 123, MPI\_COMM\_WORLD, &request);

        MPI\_Wait(&request, &status);

    }

    else

    {

        printf("Process 0; Max = %d\n", m);

        for(int j = 1; j < p; j++)

        {

            int x;

            MPI\_Request requestj;

            MPI\_Irecv(&x, 1, MPI\_INT, j, 123, MPI\_COMM\_WORLD, &requestj);

            MPI\_Wait(&requestj, &status);

        printf("Received : Max = %d from Process %d\n", x, j);

            if(x > m) m = x;

        }

        printf("Max of array is : %d\n", m);

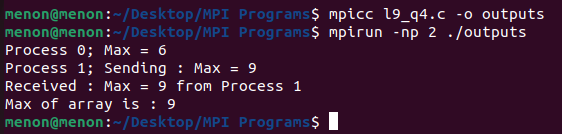
    }

    MPI\_Finalize();

    return 0;

}

Output:



1. Four Queen's Problem

Code:

#include <mpi.h>

#include <stdio.h>

int factorial(int n)

{

    int f = 1;

    for(int i = 2; i < n+1; i++)

    {

        f \*= i;

    }

    return f;

}

int check(int n, int k, int id)

{

    int len = n;

    int perm[n];

    int nums[n];

    for(int i = 0; i < n; i++) nums[i] = i;

    int size = n;

    for(int i = 0; i < n; i++)

    {

        int index = k / factorial(n-i-1);

        perm[i] = nums[index];

        for(int j = index; j < size-1; j++)

        {

            nums[j] = nums[j+1];

            size--;

        }

        k -= index\*factorial(n-i-1);

    }

    for(int i = 0; i < n; i++)

    {

        for(int j = i+1; j < n; j++)

        {

            if ((i-perm[i] == j-perm[j]) || (i+perm[i] == j+perm[j]))

            {

                return 0;

            }

        }

    }

    printf("Process %d: found solution place queens at: (%d, %d) (%d, %d) (%d, %d) (%d, %d)\n", id, 0, perm[0], 1, perm[1], 2, perm[2], 3, perm[3]);

    return 1;

}

int main(int argc, char\*\* argv) {

    int p, id, n = 4;

    int f = factorial(n);

    MPI\_Init(NULL, NULL);

    MPI\_Comm\_size(MPI\_COMM\_WORLD, &p);

    MPI\_Comm\_rank(MPI\_COMM\_WORLD, &id);

    for(int x = id; x < f; x += p)

    {

        check(n, x, id);

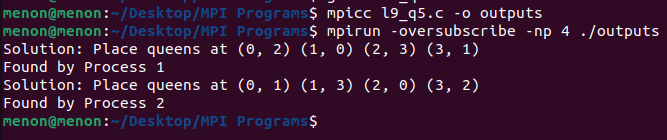
    }

    MPI\_Finalize();

    return 0;

}

Output:



1. Sample isend, ireceive with mpi\_wtime

Code:

#include <mpi.h>

#include <stdio.h>

int main(int argc, char\*\* argv)

{

    int rank;

    int buf;

    int world\_size;

    const int root=0;

    MPI\_Init(&argc, &argv);

    MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);

    MPI\_Comm\_size(MPI\_COMM\_WORLD, &world\_size);

    MPI\_Status status;

    MPI\_Request request = MPI\_REQUEST\_NULL;

    double time = 0;

    if(rank == root)

    {

        buf = 19;

        printf("Rank %d going to MPI\_Isend data '%d' to others\n",rank,buf);

        time = - MPI\_Wtime();

        for(int i=0;i<world\_size;i++)

        {

            MPI\_Isend(&buf, 1, MPI\_INT, i, 0, MPI\_COMM\_WORLD,&request);

        }

        time += MPI\_Wtime();

    }

    else

    {

        printf("[P%d]: Before MPI\_Irecv, data is %d\n", rank, buf);

        MPI\_Irecv(&buf, 1, MPI\_INT, 0, 0, MPI\_COMM\_WORLD,&request);

        MPI\_Wait(&request, &status);

        printf("[P%d]: After MPI\_Irecv, data is %d\n", rank, buf);

    }

    MPI\_Finalize();

    if(rank == root)

    {

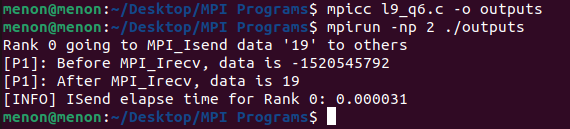
        printf("[INFO] ISend elapse time for Rank %d: %f\n",rank,time);

    }

    return 0;

}

Output:



1. Sample send and receive with mpi\_wtime

Code:

#include <mpi.h>

#include <stdio.h>

int main(int argc, char\*\* argv)

{

    int rank;

    int buf;

    const int root=0;

    double stime = 0;

    MPI\_Init(&argc, &argv);

    MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);

    int world\_size;

    MPI\_Comm\_size(MPI\_COMM\_WORLD,&world\_size);

    if(rank == root)

    {

        buf = 19;

        printf("Rank %d going to MPI\_Send data '%d' to others\n",rank,buf);

        stime = - MPI\_Wtime();

        for(int i=0;i<world\_size;i++){

            MPI\_Send(&buf, 1, MPI\_INT, i, 0, MPI\_COMM\_WORLD);

        }

        stime += MPI\_Wtime();

    }

    else

    {

        printf("[P%d]: Before MPI\_Recv, data is %d\n", rank, buf);

        MPI\_Recv(&buf, 1, MPI\_INT, 0, 0, MPI\_COMM\_WORLD,

        MPI\_STATUS\_IGNORE);

        printf("[P%d]: After MPI\_Recv, data is %d\n", rank, buf);

    }

    MPI\_Finalize();

    if(rank == root)

    {

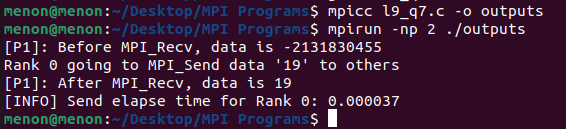
        printf("[INFO] Send elapse time for Rank %d: %f\n",rank,stime);

    }

    return 0;

}

Output:



1. Implementing the broadcast using send and receive

Code:

#include <mpi.h>

#include <stdio.h>

void my\_bcast(void\* data, int count, MPI\_Datatype datatype, int root, MPI\_Comm communicator)

{

    int world\_rank;

    MPI\_Comm\_rank(communicator, &world\_rank);

    int world\_size;

    MPI\_Comm\_size(communicator, &world\_size);

    if (world\_rank == root)

    {

        // If we are the root process, send our data to everyone

        int i;

        for (i = 0; i < world\_size; i++) {

        if (i != world\_rank) {

            MPI\_Send(data, count, datatype, i, 0, communicator);

        }

        }

    }

    else

    {

        // If we are a receiver process, receive the data from the root

        MPI\_Recv(&data, count, datatype, root, 0, communicator,

        MPI\_STATUS\_IGNORE);

        printf("Rank %d received data[%d] by MPI\_Recv\n",world\_rank,data);

    }

}

int main()

{

    MPI\_Init(NULL,NULL);

    double total\_my\_bcast\_time = 0;

    double total\_mpi\_bcast\_time = 0;

    int data=19;

    int num\_elements = 1;

    int world\_rank;

    MPI\_Comm\_rank(MPI\_COMM\_WORLD, &world\_rank);

    if(world\_rank == 0)

    {

        // Time my\_bcast

        total\_my\_bcast\_time -= MPI\_Wtime();

        my\_bcast(&data, num\_elements, MPI\_INT, 0, MPI\_COMM\_WORLD);

        total\_my\_bcast\_time += MPI\_Wtime();

        printf("Comparison between MPI\_Bcast() and my\_bcast()\nBroadcasting Done by Rank 0 process...\n");

        printf("Rank %d broadcasts the data: %d\n",world\_rank,data);

        printf("[INFO] my\_bcast elapsed time: %f\n",total\_my\_bcast\_time);

        // Time MPI\_Bcast

        total\_mpi\_bcast\_time -= MPI\_Wtime();

        MPI\_Bcast(&data, num\_elements, MPI\_INT, 0, MPI\_COMM\_WORLD);

        total\_mpi\_bcast\_time += MPI\_Wtime();

        printf("[INFO] MPI\_Bcast elapsed time: %f\n",total\_my\_bcast\_time);

    }

    else

    {

        my\_bcast(&data, num\_elements, MPI\_INT, 0, MPI\_COMM\_WORLD);

        MPI\_Bcast(&data, num\_elements, MPI\_INT, 0, MPI\_COMM\_WORLD);

        printf("Rank %d received broadcasted data[%d] \n",world\_rank,data);

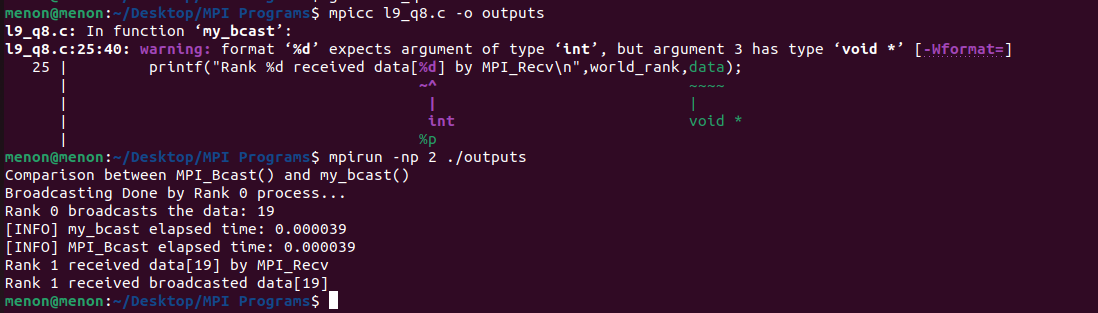
    }

    MPI\_Finalize();

    return 0;

}

Output:



1. Ring communication

Code:

#include <mpi.h>

#include <stdio.h>

int main(int argc, char \*argv[])

{

    int id, p, left, right, n = 10;

    MPI\_Request request, request2;

    MPI\_Status status;

    MPI\_Init(&argc,&argv);

    MPI\_Comm\_size(MPI\_COMM\_WORLD, &p);

    MPI\_Comm\_rank(MPI\_COMM\_WORLD, &id);

    right = (id + 1) % p;

    left = id - 1;

    if(left < 0) left = p - 1;

    MPI\_Isend(&n, 1, MPI\_INT, right, 123, MPI\_COMM\_WORLD, &request2);

    MPI\_Wait(&request2, &status);

    MPI\_Irecv(&n, 1, MPI\_INT, left, 123, MPI\_COMM\_WORLD, &request);

    MPI\_Wait(&request, &status);

    printf("Receiving Process %d received n = %d from sending process %d\n", id, n, left);

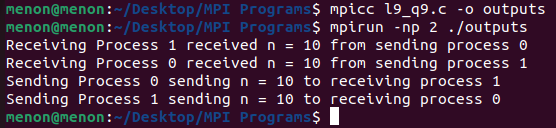
    printf("Sending Process %d sending n = %d to receiving process %d\n", id, n, right);

    MPI\_Finalize();

    return 0;

}

Output:



1. rank0 - sends randnum, rank1 - Add const , rank2 - sub const, rank3 - mul const

Code:

#include <mpi.h>

#include <stdio.h>

int main(int argc, char\*\* argv) {

    int p, id, randnum, constant = 2;

    MPI\_Status status;

    MPI\_Init(NULL, NULL);

    MPI\_Comm\_size(MPI\_COMM\_WORLD, &p);

    MPI\_Comm\_rank(MPI\_COMM\_WORLD, &id);

    if(id == 0)

    {

        randnum = 10;

        printf("Broadcasting randnum = %d\n", randnum);

        MPI\_Bcast(&randnum, 1, MPI\_INT, 0, MPI\_COMM\_WORLD);

        int x;

        MPI\_Request request1, request2, request3;

        MPI\_Irecv(&x, 1, MPI\_INT, 1, 111, MPI\_COMM\_WORLD, &request1);

        MPI\_Wait(&request1, &status);

    printf("Received %d from process 1\n", x);

    MPI\_Irecv(&x, 1, MPI\_INT, 2, 112, MPI\_COMM\_WORLD, &request2);

        MPI\_Wait(&request2, &status);

    printf("Received %d from process 2\n", x);

    MPI\_Irecv(&x, 1, MPI\_INT, 3, 113, MPI\_COMM\_WORLD, &request3);

        MPI\_Wait(&request3, &status);

    printf("Received %d from process 3\n", x);

    }

    else if(id == 1)

    {

        MPI\_Bcast(&randnum, 1, MPI\_INT, 0, MPI\_COMM\_WORLD);

        MPI\_Request request;

        int x = randnum + constant;

        MPI\_Isend(&x, 1, MPI\_INT, 0, 111, MPI\_COMM\_WORLD, &request);

        MPI\_Wait(&request, &status);

    }

    else if(id == 2)

    {

        MPI\_Bcast(&randnum, 1, MPI\_INT, 0, MPI\_COMM\_WORLD);

        MPI\_Request request;

        int x = randnum - constant;

        MPI\_Isend(&x, 1, MPI\_INT, 0, 112, MPI\_COMM\_WORLD, &request);

        MPI\_Wait(&request, &status);

    }

    else if(id == 3)

    {

        MPI\_Bcast(&randnum, 1, MPI\_INT, 0, MPI\_COMM\_WORLD);

        MPI\_Request request;

        int x = randnum \* constant;

        MPI\_Isend(&x, 1, MPI\_INT, 0, 113, MPI\_COMM\_WORLD, &request);

        MPI\_Wait(&request, &status);

    }

    if(id != 0)

    {

        printf("Process %d has received from broadcast randnum = %d\n", id, randnum);

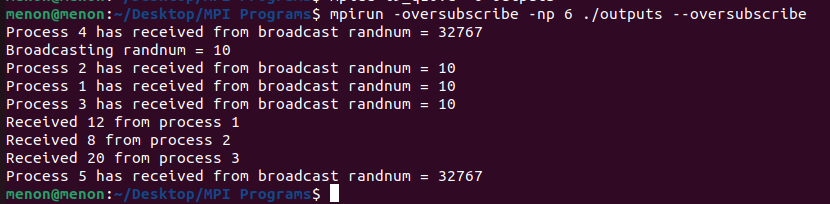
    }

    MPI\_Finalize();

    return 0;

}

Output:



1. Simulate a chat window - server answers query to client

Code:

#include <mpi.h>

#include <stdio.h>

int main(int argc, char\*\* argv) {

    int p, id, randnum, constant = 2;

    MPI\_Status status;

    MPI\_Init(NULL, NULL);

    MPI\_Comm\_size(MPI\_COMM\_WORLD, &p);

    MPI\_Comm\_rank(MPI\_COMM\_WORLD, &id);

    if(id == 0)

    {

        randnum = 10;

        printf("Broadcasting randnum = %d\n", randnum);

        MPI\_Bcast(&randnum, 1, MPI\_INT, 0, MPI\_COMM\_WORLD);

        int x;

        MPI\_Request request1, request2, request3;

        MPI\_Irecv(&x, 1, MPI\_INT, 1, 111, MPI\_COMM\_WORLD, &request1);

        MPI\_Wait(&request1, &status);

    printf("Received %d from process 1\n", x);

    MPI\_Irecv(&x, 1, MPI\_INT, 2, 112, MPI\_COMM\_WORLD, &request2);

        MPI\_Wait(&request2, &status);

    printf("Received %d from process 2\n", x);

    MPI\_Irecv(&x, 1, MPI\_INT, 3, 113, MPI\_COMM\_WORLD, &request3);

        MPI\_Wait(&request3, &status);

    printf("Received %d from process 3\n", x);

    }

    else if(id == 1)

    {

        MPI\_Bcast(&randnum, 1, MPI\_INT, 0, MPI\_COMM\_WORLD);

        MPI\_Request request;

        int x = randnum + constant;

        MPI\_Isend(&x, 1, MPI\_INT, 0, 111, MPI\_COMM\_WORLD, &request);

        MPI\_Wait(&request, &status);

    }

    else if(id == 2)

    {

        MPI\_Bcast(&randnum, 1, MPI\_INT, 0, MPI\_COMM\_WORLD);

        MPI\_Request request;

        int x = randnum - constant;

        MPI\_Isend(&x, 1, MPI\_INT, 0, 112, MPI\_COMM\_WORLD, &request);

        MPI\_Wait(&request, &status);

    }

    else if(id == 3)

    {

        MPI\_Bcast(&randnum, 1, MPI\_INT, 0, MPI\_COMM\_WORLD);

        MPI\_Request request;

        int x = randnum \* constant;

        MPI\_Isend(&x, 1, MPI\_INT, 0, 113, MPI\_COMM\_WORLD, &request);

        MPI\_Wait(&request, &status);

    }

    if(id != 0)

    {

        printf("Process %d has received from broadcast randnum = %d\n", id, randnum);

    }

    MPI\_Finalize();

    return 0;

}

Output:

