**PARALLEL AND DISTRIBUTED COMPUTING LAB CSE 4001 : LAB 9**

**Q1:Circuit Satisfiability CODE:**

#include<stdio.h> #include<mpi.h> #include<stdlib.h> #include<time.h> int ckt(int i)

{

if(i==0)

{

return 0;

}

return 1;

}

void main()

{ MPI\_Init(NULL,NULL);

int arr1[]={0,0,0,0,1,1,1,1};

int arr2[]={0,0,1,1,0,0,1,1};

int arr3[]={0,1,0,1,0,1,0,1};

int n=8; int id; int i; int p;

MPI\_Comm\_rank(MPI\_COMM\_WORLD,&id); MPI\_Comm\_size(MPI\_COMM\_WORLD,&p); int check[n];

time\_t t;

srand((unsigned) time(&t)); for(int i=id;i<n;i=i+p)

{

check[i]= rand()%2; if(ckt(check[i]))

{

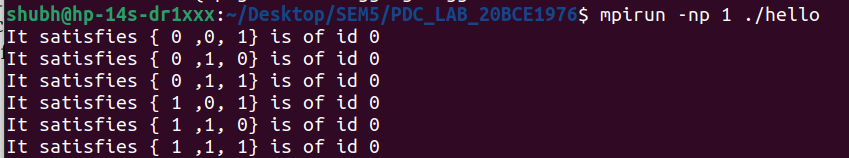
printf("It satisﬁes { %d ,%d, %d} is of id %d \n",arr1[i],arr2[i],arr3[i],id);

}

}

MPI\_Finalize();

}

**OUTPUT:**

# Q2:Number of solutions in circuit satisfiability CODE:

#include<stdio.h> #include<mpi.h> #include<stdlib.h> #include<time.h> int ckt(int i)

{

if(i==0)

{

return 0;

}

return 1;

}

void main()

{ MPI\_Init(NULL,NULL);

int arr1[]={0,0,0,0,1,1,1,1};

int arr2[]={0,0,1,1,0,0,1,1};

int arr3[]={0,1,0,1,0,1,0,1};

int n=8; int id; int i; int p;

MPI\_Comm\_rank(MPI\_COMM\_WORLD,&id); MPI\_Comm\_size(MPI\_COMM\_WORLD,&p); int check[n];

int solutions=0;

int globalsolutions=0; time\_t t;

srand((unsigned) time(&t)); for(int i=id;i<n;i=i+p)

{

check[i]= rand()%2; solutions=solutions+ ckt(check[i]);

}

MPI\_Reduce(&solutions,&globalsolutions,1,MPI\_INT,MPI\_SUM,0,MPI\_COMM

\_WORLD);

if(id==0)

{

printf("The total solution are %d \n",globalsolutions);

}

MPI\_Finalize();

}

**OUTPUT:**

# Q3:Adding a count to all values of a matrix with size n\*n CODE:

#include <mpi.h> #include <stdio.h> #include <stdlib.h> #include <time.h> int b[3][3];

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

{

int a[3][3] = {{32, 43, 72}, {45, 93, 31}, {59, 40, 23}}; MPI\_Init(NULL, NULL);

int c = 0, id, p; MPI\_Comm\_size(MPI\_COMM\_WORLD, &p);

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &id); if (id == 0)

{

printf("Row of matrix: %d\n", id + 1); for (int i = 0; i < p; i++)

{

printf("%d ", a[0][i]);

}

printf("\n"); printf("Updated:\n"); for (int i = 0; i < p; i++)

{

a[0][i] = a[0][i] + 3;

printf("%d ", a[0][i]);

}

printf("\n");

}

if (id == 1)

{

printf("Row of matrix: %d\n", id + 1); for (int i = 0; i < p; i++)

{

printf("%d ", a[1][i]);

}

printf("\n"); printf("Updated:\n"); for (int i = 0; i < p; i++)

{

a[1][i] = a[1][i] + 3;

printf("%d ", a[1][i]);

}

printf("\n");

}

if (id == 2)

{

printf("Row of matrix: %d\n", id + 1); for (int i = 0; i < p; i++)

{

printf("%d ", a[2][i]);

}

printf("\n"); printf("Updated:\n"); for (int i = 0; i < p; i++)

{

a[2][i] = a[2][i] + 3;

printf("%d ", a[2][i]);

}

printf("\n");

}

MPI\_Finalize();

}

# OUTPUT:

**4. Find Max of 'n' no's CODE:**

#include<stdio.h> #include<mpi.h> #include<stdlib.h> int main()

{ MPI\_Init(NULL,NULL);

int id; int p;

MPI\_Comm\_rank(MPI\_COMM\_WORLD,&id); MPI\_Comm\_size(MPI\_COMM\_WORLD,&p); int n;

printf("Enter the size of array: ");

scanf("%d",&n);

int answer; int arr[n];

printf("Enter the array elements : "); for(int i=0;i<n;i++)

{

scanf("%d",&arr[i]);

}

int mmax=arr[id];

for(int i=id+p;i<n;i=i+p){ if(arr[i]>mmax){ mmax=arr[i];

}

}

MPI\_Reduce(&mmax,&answer,1,MPI\_INT,MPI\_MAX,0,MPI\_COMM\_WORLD);

if(id==0)

{

printf("The Maximum Number in the array is %d\n",answer);

}

MPI\_Finalize(); return 1;

}

# OUTPUT:

**Q5:Four Queen's Problem CODE:**

#include <string.h> #include <stdio.h> #include <stdlib.h> #include <mpi.h>

typedef unsigned long long ull; int nqueens(int proc, ull i, ull n); ull factorial(ull n);

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

int rnk, sze;

ull i = 0 , n = 0, total = 0, subtot = 0; double elapsed\_time;

MPI\_Init(&argc, &argv);

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rnk); MPI\_Comm\_size(MPI\_COMM\_WORLD, &sze);

MPI\_Barrier(MPI\_COMM\_WORLD); elapsed\_time = -MPI\_Wtime();

if(argc != 2) { if(rnk == 0)

printf("Invalid number of command line arguments.\nFormat should be

./nqueens <n>\n\n");

} else {

n = atoi(argv[1]);

ull max = factorial(n);

for( i = rnk; i < max; i+=sze ) { subtot += nqueens(rnk, i, n);

}

MPI\_Reduce(&subtot, &total, 1, MPI\_INT, MPI\_SUM, 0, MPI\_COMM\_WORLD);

}

MPI\_Barrier(MPI\_COMM\_WORLD); elapsed\_time += MPI\_Wtime();

if(0 == rnk) {

printf("Program executed in %8.3f ms\n", 1000\*elapsed\_time); printf("Total number of solutions found: %llu\n\n", total);

½ush(stdout);

}

MPI\_Finalize(); return 0;

}

int nqueens(int proc, ull i, ull n) {

int a, b = 0;

int \*fact = (int \*)calloc(n, sizeof(int)); int \*perm = (int \*)calloc(n, sizeof(int));

fact[b] = 1; while(++b < (int)n) { fact[b] = fact[b-1]\*b;

}

for(b = 0; b < (int)n; ++b) {

perm[b] = i / fact[n - 1 - b];

i = i % fact[n - 1 - b];

}

for(b = n - 1; b > 0; --b) {

for(a = b - 1; a >= 0; --a) {

if(perm[a] <= perm[b]) { perm[b]++;

}

}

}

free(fact);

for(ull j = 0; j < n; j++) { int val = perm[j];

for(int k = j+1, dist = 1; k < (int)n; k++, dist++) {

if(val - dist == perm[k] || val + dist == perm[k]) { free(perm);

return 0;

}

}

for(int k = j-1, dist = 1; k >= 0; k--, dist++) {

if(val - dist == perm[k] || val + dist == perm[k]) { free(perm);

return 0;

}

}

}

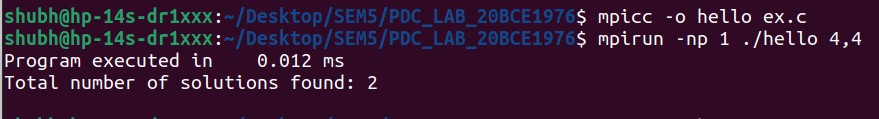
free(perm); return 1;

}

ull factorial(ull n) {

return (n == 1 || n == 0) ? 1 : factorial( n - 1 ) \* n;

}

**OUTPUT:**

# Q6:Sample isend and ireceive with mpi\_wtime CODE:

#include <mpi.h> #include <stdio.h> int main(){

int rank, size;

int tag, destination, count; int buffer;

double starttime, endtime;

tag = 1234;

destination = 1; //destination process count = 1; //number of elements in buffer

MPI\_Status status;

MPI\_Request request = MPI\_REQUEST\_NULL; MPI\_Init(NULL,NULL);

MPI\_Comm\_size(MPI\_COMM\_WORLD, &size); //number of processes MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank); //rank of current process

starttime = MPI\_Wtime(); if (rank == 0) {

buffer=10;

MPI\_Isend(&buffer, count, MPI\_INT, destination, tag, MPI\_COMM\_WORLD, &request); //non blocking send to destination process

}

if (rank == destination) {

MPI\_Irecv(&buffer, count, MPI\_INT, 0, tag, MPI\_COMM\_WORLD, &request); //destination process receives

}

MPI\_Wait(&request, &status); //bloks and waits for destination process to receive data

endtime=MPI\_Wtime(); if (rank == 0) {

printf("processor %d sent %d\n", rank, buffer);

printf("id: %d took %f seconds\n",rank,endtime-starttime);

}

if (rank == destination) {

printf("processor %d got %d\n", rank, buffer);

printf("id: %d took %f seconds\n",rank,endtime-starttime);

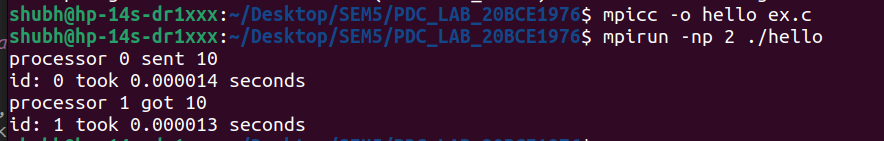
}

MPI\_Finalize();

return 0;

}

# OUTPUT:



**Q7:Sample send and receive with mpi\_wtime CODE:**

#include <mpi.h> #include <stdio.h> int main(){

MPI\_Init(NULL,NULL);

double starttime, endtime; int myid, numprocs; starttime = MPI\_Wtime();

MPI\_Comm\_size(MPI\_COMM\_WORLD, &numprocs); MPI\_Comm\_rank(MPI\_COMM\_WORLD, &myid);

int number;

if (myid == 0) { number = 20;

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

printf("id: %d sent number %d to id:1\n",myid,number);} else if (myid == 1) {

MPI\_Recv(&number, 1, MPI\_INT, 0, 0, MPI\_COMM\_WORLD,MPI\_STATUS\_IGNORE);

printf("Recieved %d number by id:%d\n",number,myid);} endtime=MPI\_Wtime();

MPI\_Finalize();

printf("id: %d took %f seconds\n",myid,endtime-starttime); return 0;

}

# OUTPUT:

**Q8:Implementing the broadcast using send and receive CODE:**

#include <mpi.h> #include <stdio.h> int main(){

MPI\_Init(NULL,NULL);

double starttime, endtime; int myid, numprocs; starttime = MPI\_Wtime();

MPI\_Comm\_size(MPI\_COMM\_WORLD, &numprocs); MPI\_Comm\_rank(MPI\_COMM\_WORLD, &myid);

int number;

if (myid == 0) { number = 20;

for(int i=1;i<numprocs;i++){

printf("id: %d sent number %d to id: %d\n",myid,number,i); MPI\_Send(&number, 1, MPI\_INT, i, 0, MPI\_COMM\_WORLD);

}

}

else {

printf("Recieved %d number by id:%d\n",number,myid); MPI\_Recv(&number, 1, MPI\_INT, 0, 0,

MPI\_COMM\_WORLD,MPI\_STATUS\_IGNORE);

}

endtime=MPI\_Wtime(); MPI\_Finalize();

printf("id: %d took %f seconds\n",myid,endtime-starttime); return 0;

}

# OUTPUT:

**Q9:RING COMMUNICATION:**

# CODE:

#include "mpi.h" #include <stdio.h> int main(){

int myid, numprocs, left, right; int buffer=10, buffer2; MPI\_Request request, request2; MPI\_Status status; MPI\_Init(NULL,NULL);

MPI\_Comm\_size(MPI\_COMM\_WORLD, &numprocs); MPI\_Comm\_rank(MPI\_COMM\_WORLD, &myid); right = (myid+ 1) % numprocs;

left = myid-1; if (left < 0)

left = numprocs-1;

printf("id: %d to id: %d sent number: %d\n",myid,right,buffer); MPI\_Isend(&buffer, 1, MPI\_INT, right, 123, MPI\_COMM\_WORLD,

&request2);

MPI\_Irecv(&buffer2, 1, MPI\_INT, left, 123, MPI\_COMM\_WORLD, &request);

MPI\_Wait(&request, &status); MPI\_Wait(&request2, &status);

printf("id: %d from id: %d, received number: %d\n",myid,left,buffer2); MPI\_Finalize();

return 0;

}

# OUTPUT:

**Q10:rank0 - sends randnum, rank1 - Add const , rank2 - sub const, rank3 - mul const**

# CODE:

#include <mpi.h> #include <stdio.h> #include<time.h> #include<stdlib.h> int main(){

time\_t t; MPI\_Init(NULL,NULL);

double starttime, endtime; int myid, numprocs; starttime = MPI\_Wtime(); srand((unsigned) time(&t));

MPI\_Comm\_size(MPI\_COMM\_WORLD, &numprocs); MPI\_Comm\_rank(MPI\_COMM\_WORLD, &myid);

int number;

int constant=5; if (myid == 0) {

for(int i=1;i<numprocs;i++){ number = rand() % 50;

printf("id: %d sent number %d to id: %d\n",myid,number,i); MPI\_Send(&number, 1, MPI\_INT, i, 0, MPI\_COMM\_WORLD);}

}

else if(myid==1){

MPI\_Recv(&number, 1, MPI\_INT, 0, 0, MPI\_COMM\_WORLD,MPI\_STATUS\_IGNORE);

printf("Recieved %d number by id:%d, After adding 10, number:

%d\n",number,myid,number+constant);

}

else if(myid==2){

MPI\_Recv(&number, 1, MPI\_INT, 0, 0,

MPI\_COMM\_WORLD,MPI\_STATUS\_IGNORE);

printf("Recieved %d number by id:%d, After subtracting 10, number:

%d\n",number,myid,number-constant);}

else if(myid==3){

MPI\_Recv(&number, 1, MPI\_INT, 0, 0, MPI\_COMM\_WORLD,MPI\_STATUS\_IGNORE);

printf("Recieved %d number by id:%d, After multiplying 10, number:

%d\n",number,myid,number\*constant);} endtime=MPI\_Wtime(); MPI\_Finalize();

printf("id: %d took %f seconds\n",myid,endtime-starttime); return 0;

}

# OUTPUT: