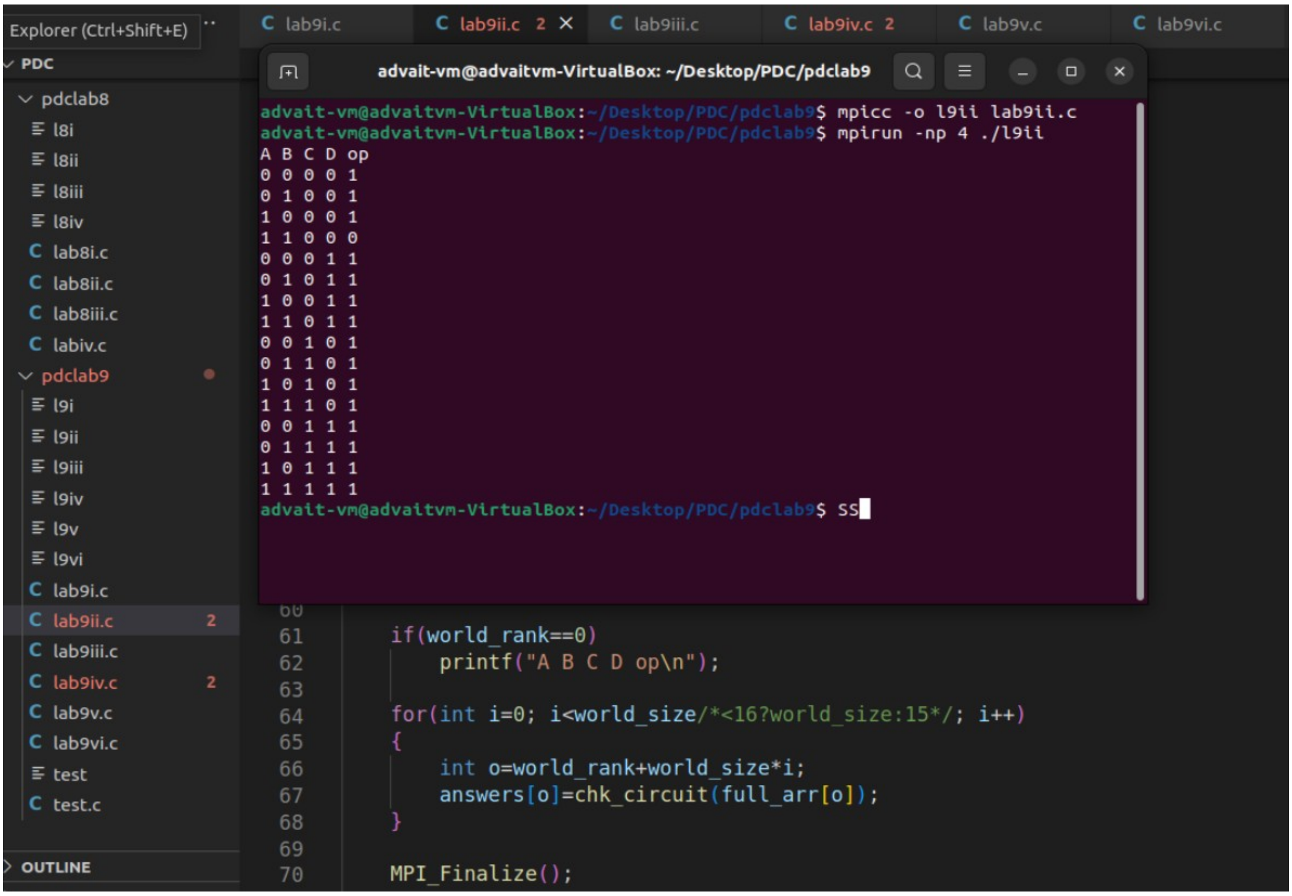
PDC Lab 9 L13+L14

Advait Deochakke

20BCE1143

Q1) Circuit Satisfiability



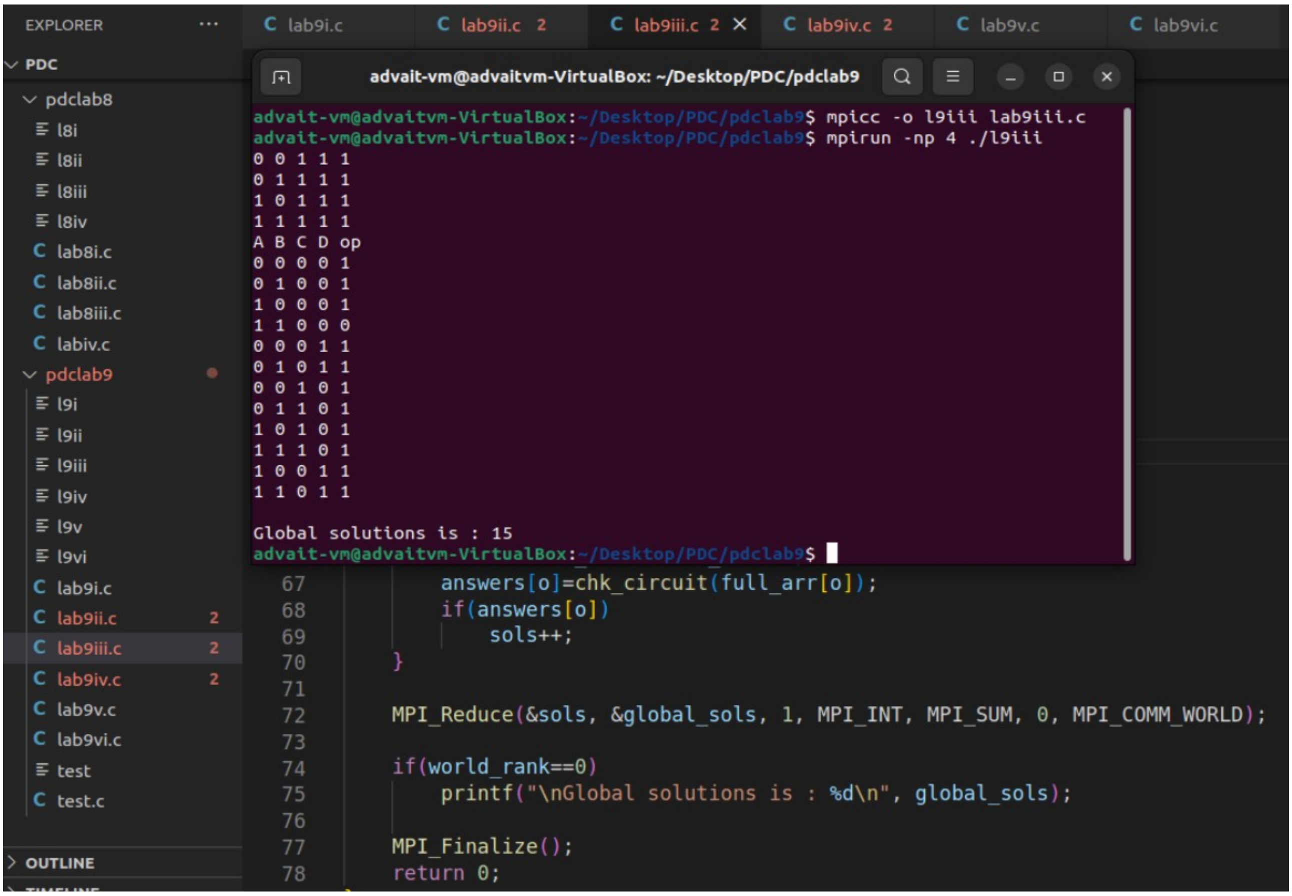
```
advait-vm@advaitvm-VirtualBox:~/Desktop/PDC/pdclab9$ mpicc -o l9ii lab9ii.c
advait-vm@advaitvm-VirtualBox:~/Desktop/PDC/pdclab9$ mpirun -np 4 ./l9ii
A B C D op
0 0 0 0 1
0 1 0 0 1
1 0 0 0 1
1 1 0 0 0
0 0 0 1 1
0 1 0 1 1
1 0 0 1 1
1 1 0 1 1
0 0 1 0 1
0 1 1 0 1
1 0 1 0 1
1 1 1 0 1
0 0 1 1 1
0 1 1 1 1
1 0 1 1 1
1 1 1 1 1
advait-vm@advaitvm-VirtualBox:~/Desktop/PDC/pdclab9$ SS
```

```
    if(world_rank==0)
        printf("A B C D op\n");

    for(int i=0; i<world_size/*<16?world_size:15*/; i++)
    {
        int o=world_rank+world_size*i;
        answers[o]=chk_circuit(full_arr[o]);
    }

    MPI_Finalize();
```

Code ->

```
#include<mpi.h>
#include<omp.h>
#include<stdio.h>
#include<math.h>
#include<stdbool.h>
//#include<iostream>

//using namespace std;

struct myb{
    bool i;
    bool j;
    bool k;
    bool l;
};

#define btoa(x) ((x)?"1":"0")

bool chk_circuit(struct myb boo)
{
    // let circuit be ab(and) + cd(or) -> (not)ab(or)cd
    bool ans = !(boo.i&&boo.j)||(boo.k||boo.l);
    printf("%s %s %s %s %s\n", btoa(boo.i), btoa(boo.j), btoa(boo.k), 
btoa(boo.l), btoa(ans));
    return ans;
}

int main()
{
    //MPI circuit satisfiability

    struct myb full_arr[16];
    for(int i=0; i<2; i++)
    {
        for(int j=0; j<2; j++)
        {
            for(int k=0; k<2; k++)
            {
                for(int l=0; l<2; l++)
                {
                    struct myb bol;
                    bol.i=(bool)i;
                    bol.j=(bool)j;
                    bol.k=(bool)k;
                    bol.l=(bool)l;
                    int val=i*8+j*4+k*2+l*1;
                    full_arr[val]=bol;
                }
            }
        }
    }
    
    bool answers[16];
    
    MPI_Init(NULL, NULL);
    int world_size;
    MPI_Comm_size(MPI_COMM_WORLD, &world_size);

    int world_rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);
    
    if(world_rank==0)
        printf("A B C D op\n");
    for(int i=0; i<world_size/*<16?world_size:15*/; i++)
    {
        int o=world_rank+world_size*i;
        answers[o]=chk_circuit(full_arr[o]);
    }

    MPI_Finalize();
    return 0;
}
```

Q2) Number of solutions in circuit satisfiability



```
advait-vm@advaitvm-VirtualBox:~/Desktop/PDC/pdclab9$ mpicc -o l9iii lab9iii.c
advait-vm@advaitvm-VirtualBox:~/Desktop/PDC/pdclab9$ mpirun -np 4 ./l9iii
0 0 1 1 1
0 1 1 1 1
1 0 1 1 1
1 1 1 1 1
A B C D op
0 0 0 0 1
0 1 0 0 1
1 0 0 0 1
1 1 0 0 0
0 0 0 1 1
0 1 0 1 1
0 0 1 0 1
0 1 1 0 1
1 0 1 0 1
1 1 1 0 1
1 0 0 1 1
1 1 0 1 1

Global solutions is : 15
advait-vm@advaitvm-VirtualBox:~/Desktop/PDC/pdclab9$ 
```

```
            answers[o]=chk_circuit(full_arr[o]);
            if(answers[o])
                sols++;
        }

        MPI_Reduce(&sols, &global_sols, 1, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);

        if(world_rank==0)
            printf("\nGlobal solutions is : %d\n", global_sols);

        MPI_Finalize();
        return 0;
```

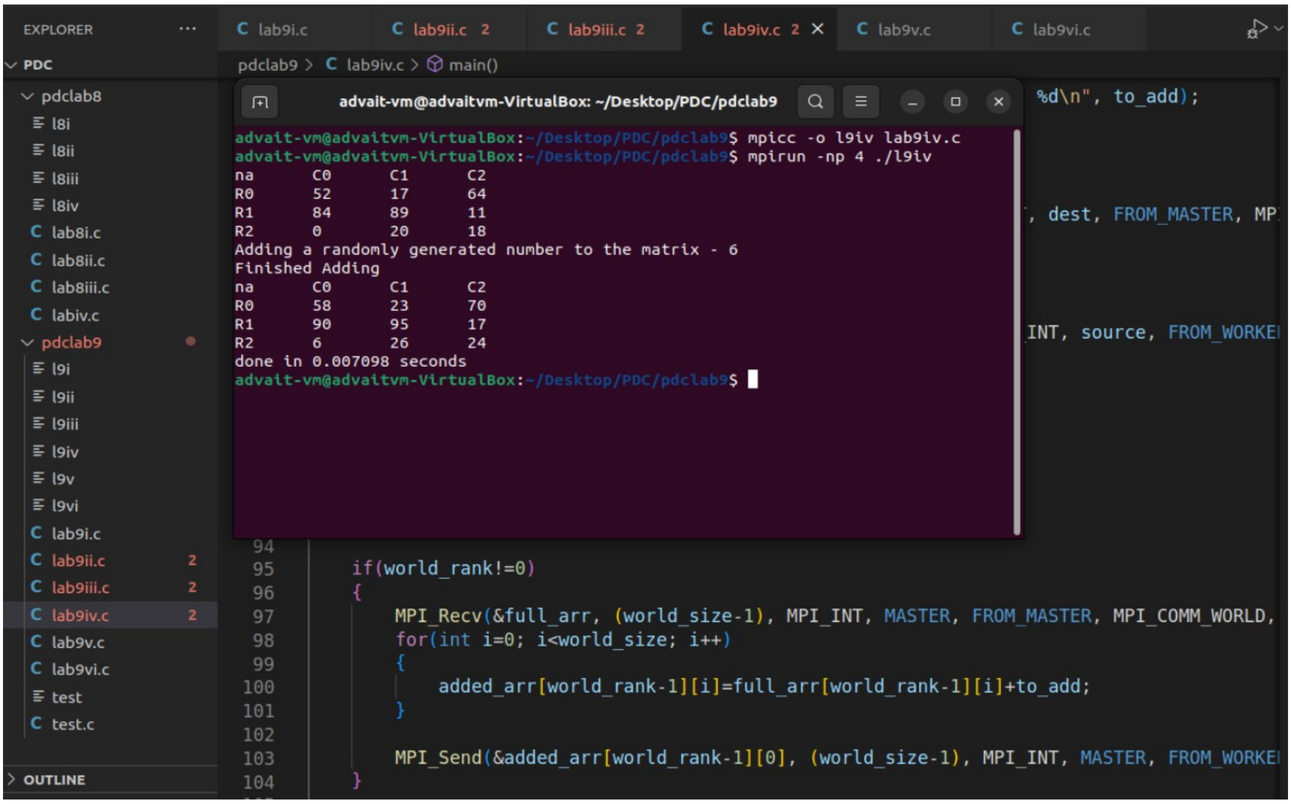
Code->

```
#include<mpi.h>
#include<omp.h>
#include<stdio.h>
#include<math.h>
#include<stdbool.h>
//#include<iostream>
```

```
//using namespace std;

struct myb{
    bool i;
    bool j;
    bool k;
    bool l;
};

#define btoa(x) ((x)?"1":"0")

bool chk_circuit(struct myb boo)
{
    // let circuit be ab(and) + cd(or) -> (not)ab(or)cd
    bool ans = !(boo.i&&boo.j)||(boo.k||boo.l);
    printf("%s %s %s %s %s\n", btoa(boo.i), btoa(boo.j), btoa(boo.k),
btoa(boo.l), btoa(ans));
    return ans;
}

int main()
{
    //MPI circuit satisfiability

    struct myb full_arr[16];
    for(int i=0; i<2; i++)
    {
        for(int j=0; j<2; j++)
        {
            for(int k=0; k<2; k++)
            {
                for(int l=0; l<2; l++)
                {
                    struct myb bol;
                    bol.i=(bool)i;
                    bol.j=(bool)j;
                    bol.k=(bool)k;
                    bol.l=(bool)l;
                    int val=i*8+j*4+k*2+l*1;
                    full_arr[val]=bol;
                }
            }
        }
    }

    bool answers[16];
    int global_sols;
    MPI_Init(NULL, NULL);
    int world_size;
    MPI_Comm_size(MPI_COMM_WORLD, &world_size);
```

```c
  int world_rank;
  MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);

  if(world_rank==0)
    printf("A B C D op\n");
  int sols=0;
  for(int i=0; i<world_size/*<16?world_size:15*/; i++)
  {
    int o=world_rank+world_size*i;
    answers[o]=chk_circuit(full_arr[o]);
    if(answers[o])
      sols++;
  }

  MPI_Reduce(&sols, &global_sols, 1, MPI_INT, MPI_SUM, 0,
MPI_COMM_WORLD);

  if(world_rank==0)
    printf("\nGlobal solutions is : %d\n", global_sols);

  MPI_Finalize();
  return 0;
}
```

Q3) Adding a count to all values of a matrix with size n*n



```
advait-vm@advaitvm-VirtualBox:~/Desktop/PDC/pdclab9$ mpicc -o l9iv lab9iv.c
advait-vm@advaitvm-VirtualBox:~/Desktop/PDC/pdclab9$ mpirun -np 4 ./l9iv
na      C0      C1      C2
R0      52      17      64
R1      84      89      11
R2      0       20      18
Adding a randomly generated number to the matrix - 6
Finished Adding
na      C0      C1      C2
R0      58      23      70
R1      90      95      17
R2      6       26      24
done in 0.007098 seconds
advait-vm@advaitvm-VirtualBox:~/Desktop/PDC/pdclab9$ 
```

```c
%d\n", to_add);
, dest, FROM_MASTER, MP
INT, source, FROM_WORKE

    if(world_rank!=0)
    {
        MPI_Recv(&full_arr, (world_size-1), MPI_INT, MASTER, FROM_MASTER, MPI_COMM_WORLD,
        for(int i=0; i<world_size; i++)
        {
            added_arr[world_rank-1][i]=full_arr[world_rank-1][i]+to_add;
        }

        MPI_Send(&added_arr[world_rank-1][0], (world_size-1), MPI_INT, MASTER, FROM_WORKE
    }
```

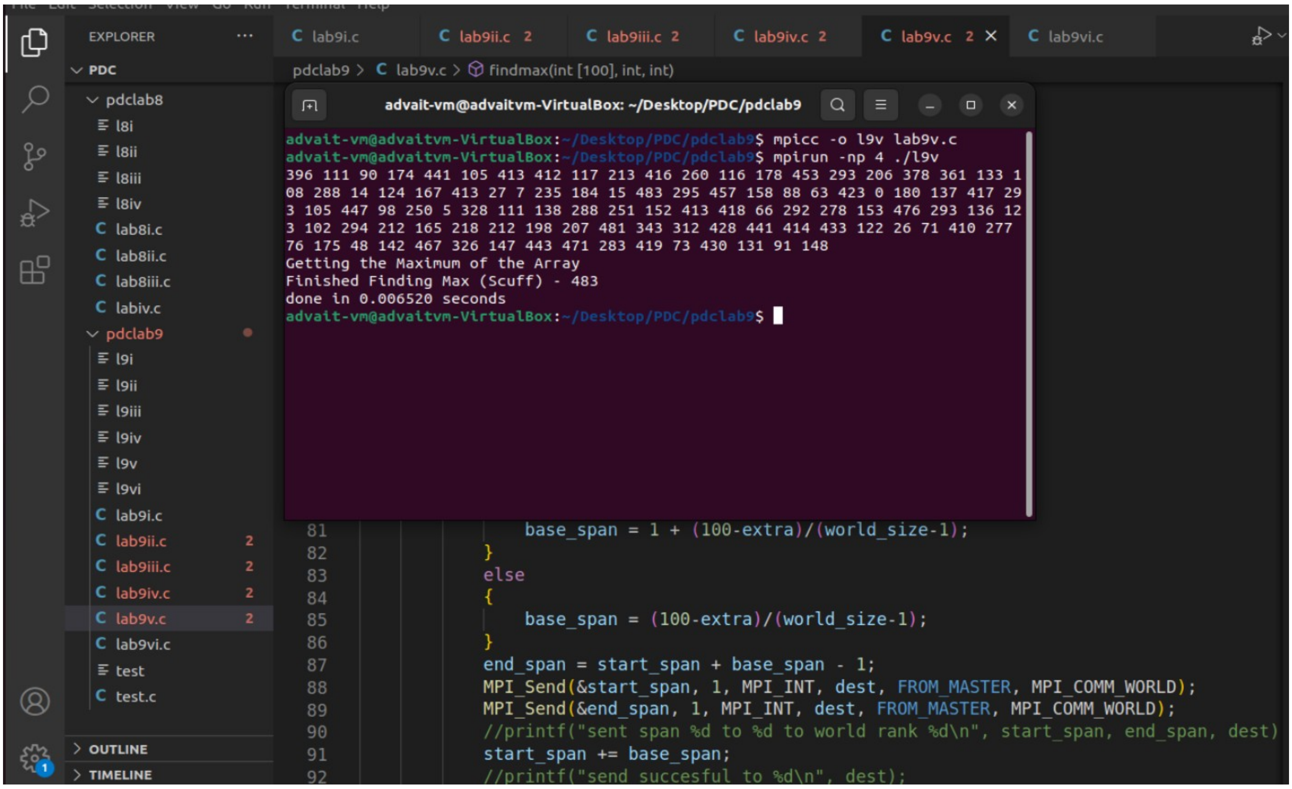
Code ->

```
#include<mpi.h>
#include<omp.h>
#include<stdio.h>
#include<stdlib.h>
#include<time.h>
#include<math.h>
#include<stdbool.h>
//#include<iostream>

//using namespace std;

#define MASTER 0
#define FROM_MASTER 1
#define FROM_WORKER 2

void printmat(int arr[100][100], int world_size)
{
  printf("na");
  for(int i=0; i<world_size; i++)
    printf("\tC%d", i);
  printf("\n");
  for(int i=0; i<world_size; i++)
  {
    printf("R%d", i);
    for(int j=0; j<world_size; j++)
    {
      printf("\t%d", arr[i][j]);
    }
    printf("\n");
  }
}

void addtomat(int arr[100][100], int world_size, int world_rank, int 
to_add)
{ 
  printf("executing from inside world - %d\n", world_rank);
  for(int j=0; j<world_size; j++)
  {
    arr[world_rank][j]+=to_add;
  }
}

int main()
{
  //MPI add number to matrix

  MPI_Status status;
```

```c
    MPI_Init(NULL, NULL);

    int world_size;
    MPI_Comm_size(MPI_COMM_WORLD, &world_size);

    int world_rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);

    int full_arr[100][100];
    int added_arr[100][100];
    int to_add;

    srand(time(NULL));
    for(int i=0; i<world_size-1; i++)
    {
        for(int j=0; j<world_size-1; j++)
        {
            int x=rand()%100;
            full_arr[i][j]=x;
            added_arr[i][j]=0;
        }
    }

    to_add=rand()%10+1;
    if(world_rank==0)
    {
        printmat(full_arr, world_size-1);
        printf("Adding a randomly generated number to the matrix - %d\n", to_add);
        double start = MPI_Wtime();

        for(int dest=1; dest<=world_size-1; dest++)
        {
            MPI_Send(&full_arr[dest-1][0], (world_size-1), MPI_INT, dest, FROM_MASTER, MPI_COMM_WORLD);
        }

        for(int source=1; source<=world_size-1; source++)
        {
            MPI_Recv(&added_arr[source-1][0], (world_size-1), MPI_INT, source, FROM_WORKER, MPI_COMM_WORLD, &status);
        }

        printf("Finished Adding\n");
        printmat(added_arr, world_size-1);
        double finish = MPI_Wtime();
        printf("done in %f seconds\n", finish-start);
    }

    if(world_rank!=0)
    {
```

```
    MPI_Recv(&full_arr, (world_size-1), MPI_INT, MASTER, FROM_MASTER, 
MPI_COMM_WORLD, &status);
    for(int i=0; i<world_size; i++)
    {
      added_arr[world_rank-1][i]=full_arr[world_rank-1][i]+to_add;
    }
    MPI_Send(&added_arr[world_rank-1][0], (world_size-1), MPI_INT, 
MASTER, FROM_WORKER, MPI_COMM_WORLD);
  }

  MPI_Finalize();
  return 0;
}
```

Q4) Find Max of 'n' no's



```
advait-vm@advaitvm-VirtualBox:~/Desktop/PDC/pdclab9$ mpicc -o l9v lab9v.c
advait-vm@advaitvm-VirtualBox:~/Desktop/PDC/pdclab9$ mpirun -np 4 ./l9v
396 111 90 174 441 105 413 412 117 213 416 260 116 178 453 293 206 378 361 133 1
08 288 14 124 167 413 27 7 235 184 15 483 295 457 158 88 63 423 0 180 137 417 29
3 105 447 98 250 5 328 111 138 288 251 152 413 418 66 292 278 153 476 293 136 12
3 102 294 212 165 218 212 198 207 481 343 312 428 441 414 433 122 26 71 410 277
76 175 48 142 467 326 147 443 471 283 419 73 430 131 91 148
Getting the Maximum of the Array
Finished Finding Max (Scuff) - 483
done in 0.006520 seconds
advait-vm@advaitvm-VirtualBox:~/Desktop/PDC/pdclab9$ 
```

```
            base_span = 1 + (100-extra)/(world_size-1);
        }
        else
        {
            base_span = (100-extra)/(world_size-1);
        }
        end_span = start_span + base_span - 1;
        MPI_Send(&start_span, 1, MPI_INT, dest, FROM_MASTER, MPI_COMM_WORLD);
        MPI_Send(&end_span, 1, MPI_INT, dest, FROM_MASTER, MPI_COMM_WORLD);
        //printf("sent span %d to %d to world rank %d\n", start_span, end_span, dest)
        start_span += base_span;
        //printf("send succesful to %d\n", dest);
```

Code ->

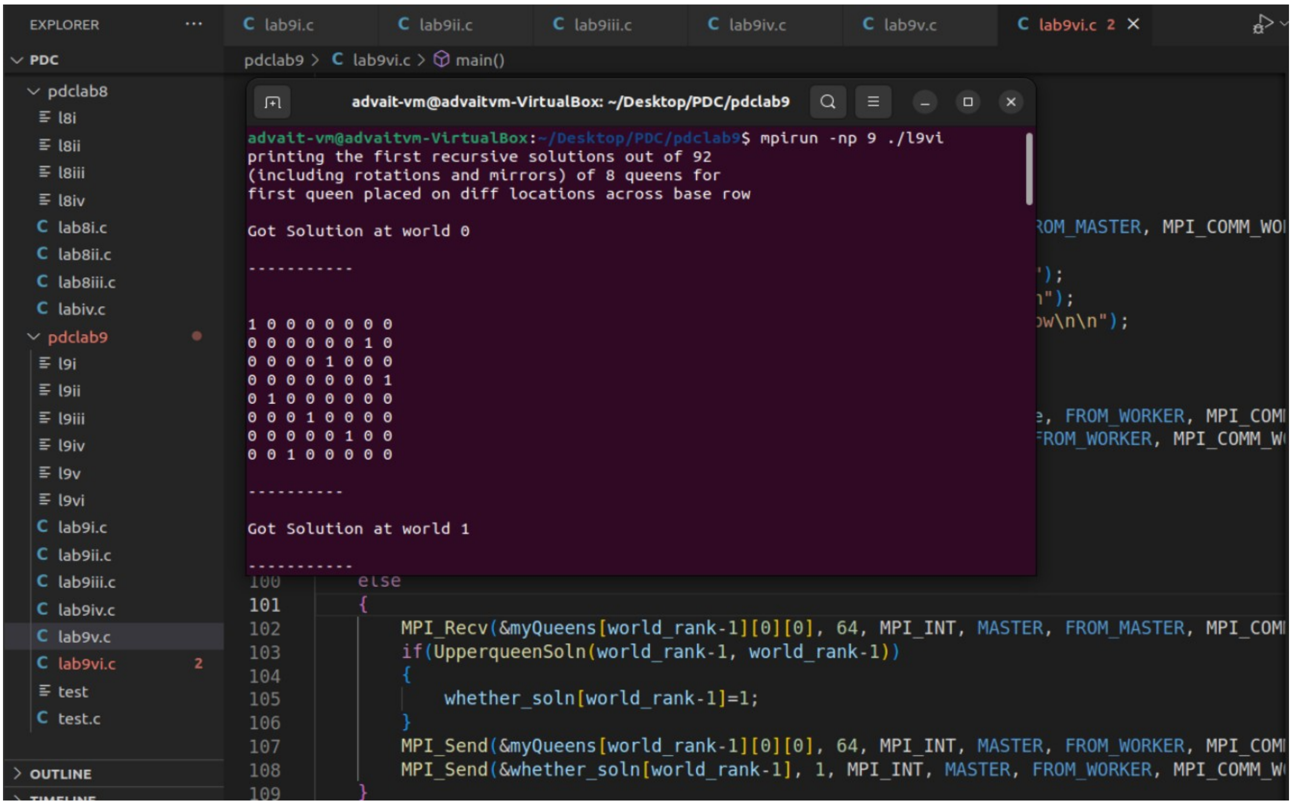
```
#include<mpi.h>
#include<omp.h>
#include<stdio.h>
#include<stdlib.h>
#include<time.h>
#include<math.h>
```

```
#include<stdbool.h>
//#include<iostream>

//using namespace std;

#define MASTER 0
#define FROM_MASTER 1
#define FROM_WORKER 2

void printarr(int arr[100])
{
  for(int i=0; i<100; i++)
    printf("%d ", arr[i]);
  printf("\n");
}

int findmax(int arr[100], int start_span, int end_span)
{
  int loc_mx=0;
  for(int span=start_span; span<end_span; span++)
  {
    //printf("currently checking %d\n", arr[span]);
    loc_mx=arr[span]>loc_mx?arr[span]:loc_mx;
  }
  //printf("findmax ended successfully\n");
  return loc_mx;
}

int main()
{
  //MPI add number to matrix

  MPI_Status status;

  MPI_Init(NULL, NULL);

  int world_size;
  MPI_Comm_size(MPI_COMM_WORLD, &world_size);

  int world_rank;
  MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);

  //char processor_name[MPI_MAX_PROCESSOR_NAME];
  //int name_len;
  //MPI_Get_processor_name(processor_name, &name_len);
  int full_arr[100];
  int span_chart[100]={0};
  int global_max_scuff=0;
  //int global_max_reduce = 0;
  int local_max=0;
```

```
    srand(time(NULL));
    for(int i=0; i<100; i++)
    {
        int x=rand()%500;
        full_arr[i]=x;
    }

    int extra=100%(world_size-1);
    int base_span;
    int start_span=0;
    int end_span;
    //int dummy_sync = 0;
    if(world_rank==0)
    {
        printarr(full_arr);
        printf("Getting the Maximum of the Array\n");
        double start = MPI_Wtime();

        //sending
        for(int dest=1; dest<=world_size-1; dest++)
        {
            if(dest<=extra)
            {
                base_span = 1 + (100-extra)/(world_size-1);
            }
            else
            {
                base_span = (100-extra)/(world_size-1);
            }
            end_span = start_span + base_span - 1;
            MPI_Send(&start_span, 1, MPI_INT, dest, FROM_MASTER, 
MPI_COMM_WORLD);
            MPI_Send(&end_span, 1, MPI_INT, dest, FROM_MASTER, MPI_COMM_WORLD);
            //printf("sent span %d to %d to world rank %d\n", start_span, 
end_span, dest);
            start_span += base_span;
            //printf("send succesful to %d\n", dest);
        }

        //receiving
        for(int source=1; source<=world_size-1; source++)
        {
            MPI_Recv(&local_max, 1, MPI_INT, source, FROM_WORKER, 
MPI_COMM_WORLD, &status);
            //printf("received successfully from %d\n", source);
            if(local_max>global_max_scuff)
                global_max_scuff=local_max;
        }

        //MPI_Reduce(&local_max, &global_max_reduce, 1, MPI_INT, MPI_MAX, 
MASTER, MPI_COMM_WORLD);
```

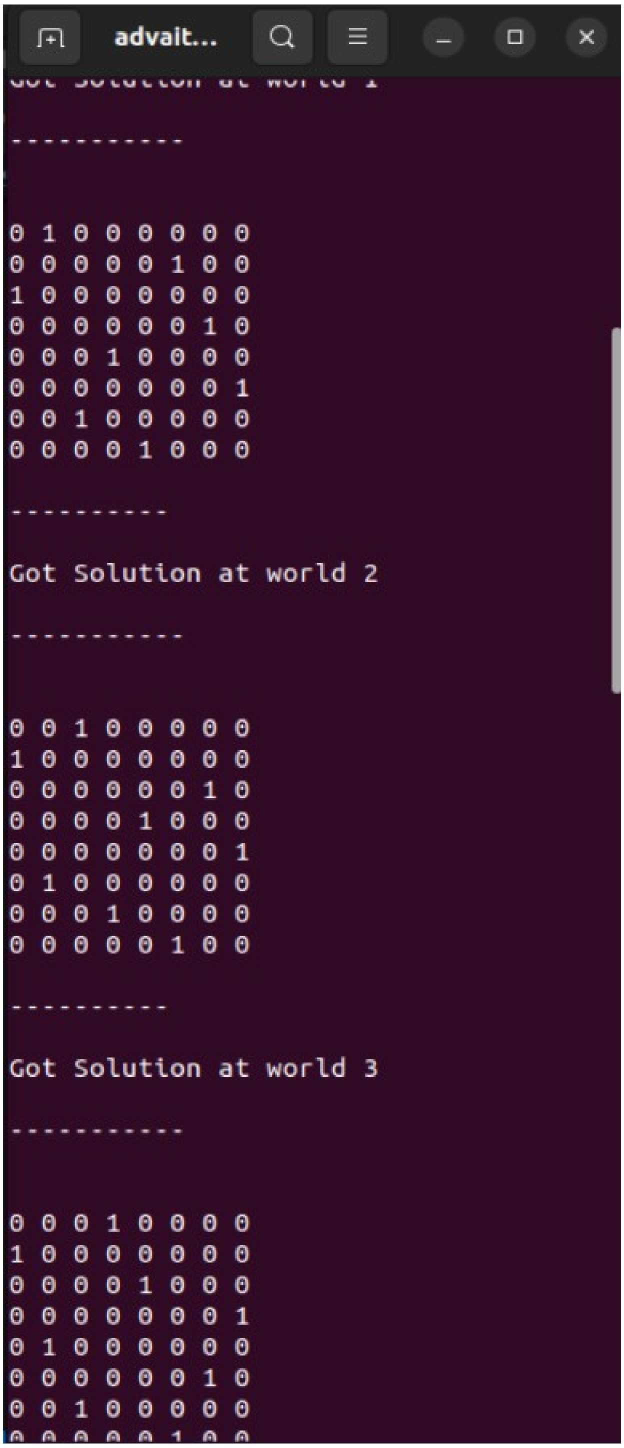
```
    printf("Finished Finding Max (Scuff) - %d\n", global_max_scuff);
    //printf("Finished Finding Max (Reduce) - %d\n", global_max_reduce);
    double finish = MPI_Wtime();
    printf("done in %f seconds\n", finish-start);

  }

  if(world_rank!=0)
  {
    MPI_Recv(&span_chart[2*world_rank], 1, MPI_INT, MASTER, FROM_MASTER, 
MPI_COMM_WORLD, &status);
    MPI_Recv(&span_chart[2*world_rank+1], 1, MPI_INT, MASTER, 
FROM_MASTER, MPI_COMM_WORLD, &status);
    local_max = findmax(full_arr, span_chart[2*world_rank], 
span_chart[2*world_rank+1]);
    //printf("local max from world %d is %d\n", world_rank, local_max);
    MPI_Send(&local_max, 1, MPI_INT, MASTER, FROM_WORKER, 
MPI_COMM_WORLD);
    //printf("sucessfull send from world %d\n", world_rank);

  }

  MPI_Finalize();
  return 0;
}
```

Q5) Four Queen's Problem

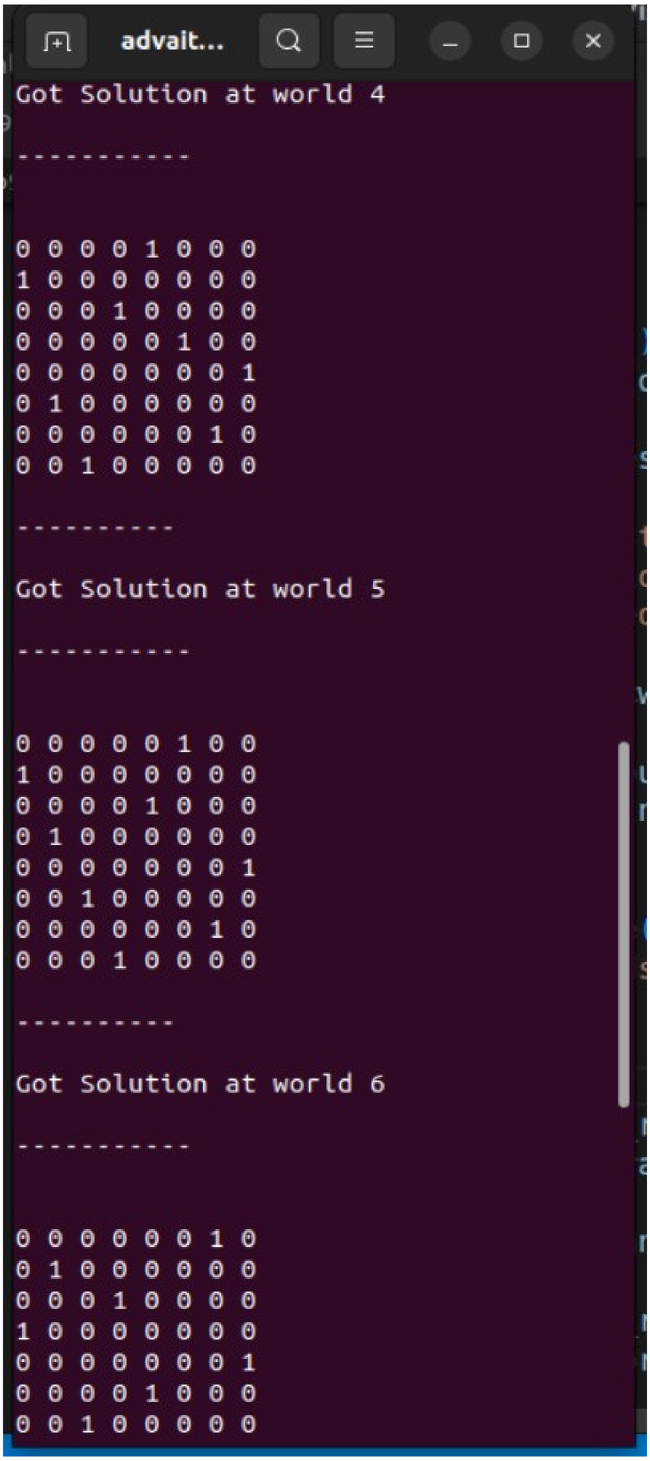


```
advait-vm@advaitvm-VirtualBox:~/Desktop/PDC/pdclab9$ mpirun -np 9 ./l9vi
printing the first recursive solutions out of 92
(including rotations and mirrors) of 8 queens for
first queen placed on diff locations across base row

Got Solution at world 0

-----------


1 0 0 0 0 0 0 0
0 0 0 0 0 0 1 0
0 0 0 0 1 0 0 0
0 0 0 0 0 0 0 1
0 1 0 0 0 0 0 0
0 0 0 1 0 0 0 0
0 0 0 0 0 1 0 0
0 0 1 0 0 0 0 0

----------

Got Solution at world 1

-----------
```

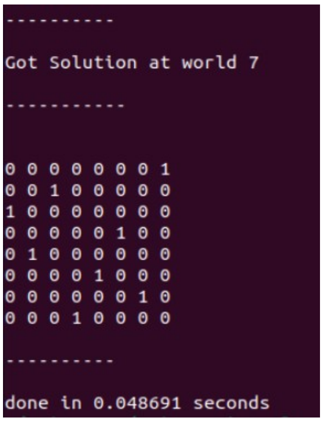
```
    else
    {
        MPI_Recv(&myQueens[world_rank-1][0][0], 64, MPI_INT, MASTER, FROM_MASTER, MPI_COM
        if(UpperqueenSoln(world_rank-1, world_rank-1))
        {
            whether_soln[world_rank-1]=1;
        }
        MPI_Send(&myQueens[world_rank-1][0][0], 64, MPI_INT, MASTER, FROM_WORKER, MPI_COM
        MPI_Send(&whether_soln[world_rank-1], 1, MPI_INT, MASTER, FROM_WORKER, MPI_COMM_W
    }
```



```
Got Solution at world 1

-----------


0 1 0 0 0 0 0 0
0 0 0 0 0 1 0 0
1 0 0 0 0 0 0 0
0 0 0 0 0 0 1 0
0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 1
0 0 1 0 0 0 0 0
0 0 0 0 1 0 0 0

----------

Got Solution at world 2

-----------


0 0 1 0 0 0 0 0
1 0 0 0 0 0 0 0
0 0 0 0 0 0 1 0
0 0 0 0 1 0 0 0
0 0 0 0 0 0 0 1
0 1 0 0 0 0 0 0
0 0 0 1 0 0 0 0
0 0 0 0 0 1 0 0

----------

Got Solution at world 3

-----------


0 0 0 1 0 0 0 0
1 0 0 0 0 0 0 0
0 0 0 0 1 0 0 0
0 0 0 0 0 0 0 1
0 1 0 0 0 0 0 0
0 0 0 0 0 0 1 0
0 0 1 0 0 0 0 0
0 0 0 0 0 1 0 0
```



```
Got Solution at world 4

-----------


0 0 0 0 1 0 0 0
1 0 0 0 0 0 0 0
0 0 0 1 0 0 0 0
0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 1
0 1 0 0 0 0 0 0
0 0 0 0 0 0 1 0
0 0 1 0 0 0 0 0

----------

Got Solution at world 5

-----------


0 0 0 0 0 1 0 0
1 0 0 0 0 0 0 0
0 0 0 0 1 0 0 0
0 1 0 0 0 0 0 0
0 0 0 0 0 0 0 1
0 0 1 0 0 0 0 0
0 0 0 0 0 0 1 0
0 0 0 1 0 0 0 0

----------

Got Solution at world 6

-----------


0 0 0 0 0 0 1 0
0 1 0 0 0 0 0 0
0 0 0 1 0 0 0 0
1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1
0 0 0 0 1 0 0 0
0 0 1 0 0 0 0 0
```



```
----------

Got Solution at world 7

-----------


0 0 0 0 0 0 0 1
0 0 1 0 0 0 0 0
1 0 0 0 0 0 0 0
0 0 0 0 0 1 0 0
0 1 0 0 0 0 0 0
0 0 0 0 1 0 0 0
0 0 0 0 0 0 1 0
0 0 0 1 0 0 0 0

----------

done in 0.048691 seconds
```

Code -> *Long*

```
//recursion pseudo
//place first queen at [rank][0]
//for X in 0:8
    //for Y in 0:8
        //if X,Y == 0
            //place queen
            //invalidate(x, y) - add 1 to all squares
            //recurse
            //remove queen
            //validate(x, y) - sub 1 from all squares
//if placed==8
    //print matrix
/*************************************
 * after more consideration, above method is kinda unfeasible, because of
 * 0s (blank possible spaces for queen) appearing at the end and
 * messing with stuff later
 * Going back to normal n-Queens
 * ***********************************/

/*****************
 * Normal n-Queens algo -
 * isSafe(x, y, board) -> check diagonals and row/cols of (x, y) of board
 * Recursive bool nQueen(board, current col)
 * -if(col>=boardLength) -> return True (board is full with N Queens)
 * -for(row space in the given column)
 * -if isSafe(row space, col, board)
 * -{
 * -    board(row space, col) = 1
 * -    if(recursive nQueens(board, next col))
 * -        return true
 * -    board(row space, col) = 0 -> backtrack
 * -}
 *
 *
 * main()
 * {
 *  if(recursive nQeeun(board, world_rank))
 *      print(solution)
 *  else
 *      print("no sol found")
 * }
 * ***************/
#include<mpi.h>
#include<stdlib.h>
#include<stdio.h>
#include<stdbool.h>
```

```
#define MASTER 0
#define FROM_MASTER 1
#define FROM_WORKER 2


int myQueens[8][8][8] = {0};
//ideally only need one 8x8 matrix, not 8 sets. but im too lazy to recode
int whether_soln[8]={0};
int synchronizer=0;

bool UpperqueenSoln(int boardNum, int col);
bool queenSoln(int boardNum, int col);
bool checkCols(int boardNum, int row, int col);
bool checkSafe(int boardNum, int row, int col);
void printsols();
void printarray(int boardNum);


int main()
{


    MPI_Status status;

    MPI_Init(NULL, NULL);

    int world_size;
    MPI_Comm_size(MPI_COMM_WORLD, &world_size);

    int world_rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);

    if(world_rank==MASTER)
    {
        double start = MPI_Wtime();
        for(int dest=1; dest<world_size; dest++)
        {
            MPI_Send(&myQueens[dest-1][0][0], 64, MPI_INT, dest, FROM_MASTER, 
MPI_COMM_WORLD);//synchronize and signal start of the program
        }
        printf("printing the first recursive solutions out of 92\n");
        printf("(including rotations and mirrors) of 8 queens for\n");
        printf("first queen placed on diff locations across base row\n\n");

        for(int source=1; source<world_size; source++)
        {
            MPI_Recv(&myQueens[source-1][0][0], 64, MPI_INT, source, 
FROM_WORKER, MPI_COMM_WORLD, &status);
            MPI_Recv(&whether_soln[source-1], 1, MPI_INT, source, FROM_WORKER, 
MPI_COMM_WORLD, &status);//synchronize receive and whether a solution has
```

```
been found or not
        }
        printsols();
        double finish = MPI_Wtime();
        printf("done in %f seconds\n", finish-start);
    }
    else
    {
        MPI_Recv(&myQueens[world_rank-1][0][0], 64, MPI_INT, MASTER, 
FROM_MASTER, MPI_COMM_WORLD, &status);//sync start
        if(UpperqueenSoln(world_rank-1, world_rank-1))
        {
            whether_soln[world_rank-1]=1;
        }
        MPI_Send(&myQueens[world_rank-1][0][0], 64, MPI_INT, MASTER, 
FROM_WORKER, MPI_COMM_WORLD);
        MPI_Send(&whether_soln[world_rank-1], 1, MPI_INT, MASTER, 
FROM_WORKER, MPI_COMM_WORLD);//sync
    }

    MPI_Finalize();

    return 0;
}

bool UpperqueenSoln(int boardNum, int col)
{
    myQueens[boardNum][0][col]=1;
    return(queenSoln(boardNum, 0));
}

bool queenSoln(int boardNum, int col)
{
    if(col >= 8)
        return true;
    for(int i=0; i<8; i++)
    {
        if(checkSafe(boardNum, i, col))
        {
            if(checkCols(boardNum, i, col))
            {
                if(queenSoln(boardNum, col+1))
                    return true;
            }
            else{
                myQueens[boardNum][i][col] = 1;
                if(queenSoln(boardNum, col+1))
                    return true;
                /*if(boardNum==7)
                    printarray(boardNum);
                */myQueens[boardNum][i][col] = 0;
```

```
      }
    }
  }
  return false;
}

bool checkCols(int boardNum, int row, int col)
{
  //across the col set by og queen call
  for(int j=0; j<8; j++)
    if(myQueens[boardNum][j][col])
      return true;

  return false;
}

bool checkSafe(int boardNum, int row, int col)
{
  //edge case check
  if(boardNum==7 && col==7)
    return true;

  //across the row
  for(int i=0; i<8; i++)
    if(myQueens[boardNum][row][i])
      return false;

  //upper left diag
  for(int i=row, j=col; i>=0 && j>=0; i--, j--)
    if(myQueens[boardNum][i][j])
      return false;

  //lower right diag
  for(int i=row, j=col; i<=7 && j<=7; i++, j++)
    if(myQueens[boardNum][i][j])
      return false;

  //lower left diag
  for(int i=row, j=col; i < 8 && j>=0; i++, j--)
    if(myQueens[boardNum][i][j])
      return false;

  //upper right diag
  for(int i=row, j=col; i>=0 && j<8; i--, j++)
    if(myQueens[boardNum][i][j])
      return false;

  return true;
}

void printsols()
```

```
{
  for(int i=0; i<8; i++)
  {
    if(whether_soln[i])
      {
        printf("Got Solution at world %d\n\n-----------\n", i);
        printarray(i);
        printf("\n----------\n\n");
      }
    else
      printf("no solution from world %d\n\n----------\n\n", i);
  }
}

void printarray(int boardNum)
{
  printf("\n");
  for(int i=0; i<8; i++)
  {
    printf("\n");
    for(int j=0; j<8; j++)
    {
      printf("%d ", myQueens[boardNum][i][j]);
    }
  }
  printf("\n");
}
```