0161

National University of Computer and Emerging Sciences

FAST School of Computing Spring-2023 Islamabad Campus

CS-3006: Parallel and Distributed Computing

Serial No:

Final Exam (Part B)
Total Time: 1.5 Hours
Total Marks: 100

Friday, 2nd June, 2023

Course Instructors

Dr. Aleem, Dr. Arshad Islam, Dr. Qaisar Shafi,
Ms. Humera Sabir, Ms. Madiha Umar

Signature of Invigilator

Asadullah Nawaz	20I-0761	C	
Student Name	Roll No.	Course Section	Student Signature

DO NOT OPEN THE QUESTION BOOK OR START UNTIL INSTRUCTED.

Instructions:

1. Attempt on question paper. Attempt all of them. Read the question carefully, understand the question, and then attempt it.
2. No additional sheet will be provided for rough work. Use the back of the last page for rough work.
3. If you need more space write on the back side of the paper and clearly mark question and part number etc.
4. After asked to commence the exam, please verify that you have **14** different printed pages(for Part-B) including this title page.
5. Calculator sharing is strictly prohibited.
6. Use permanent ink pens only. Any part done using soft pencil will not be marked and cannot be claimed for rechecking.

Part-B	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Total
Marks Obtained	9	6	8	07	08	18	13	69
Total Marks	10	10	10	08	10	28	24	100

Question 1 [10 Marks]

Below is the sketch of an OpenCL code that converts even rows of colored pixels into greyscale. T
means we aim to convert row 0,2,4,6,8… of pixels into greyscale pixels. A greyscale image can
obtained by replacing a pixel with the average of RGB of color image. You are also provided
command for enqueuing the kernel at host. You are required to complete the code by filling the blanks

```
size_t globalWorkSize[2] = {width, height};
size_t localWorkSize[2] = {16, 16}; //

clEnqueueNDRangeKernel(commandQueue, kernel, 2, NULL, globalWorkSize,
localWorkSize, 0, NULL, NULL);

__kernel void colorToGrayscale(__global const uchar* inputImage, __global uchar*
outputImage, int width, int height)
{
    int globalIdX = get_global_id(0);

    int globalIdY = get_global_id(1);

    // Check if the current row is an even or odd row
    bool isEvenRow = (globalIdY % 2 == 0);

    // Calculate the index of the current pixel
    int index = globalIdY * height + globalIdX;

    uchar pixel = inputImage[index];

    // Convert color pixel to grayscale
    uchar gray = (pixel.r + pixel.g + pixel.b)/3;

    // Set the grayscale value for alternating rows of pixels
    if ((isEvenRow && isEvenPixel) || (!isEvenRow && !isEvenPixel)) {
        outputImage[index] = gray;
    }
}
```

Question 2 [10 Marks]

a. Parallel and distributed processing can be very helpful for training of large datasets. Complete the code below in which master initiates the dataset for training. The Dataset is distributed to all processes. Each process return its trained part to the main processor. Use exact variables and complete statement in the blank link.

Note: There is only one line missing, use collective communication functions only to complete the blank link.

```
#include <stdio.h>
#include <stdlib.h>
#include <mpi.h>
#define DATASET_SIZE 2000


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

    int rank, size;
    int* dataset = NULL;
    int* local_data = NULL;
    int local_size;


    MPI_Init(&argc, &argv);

    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
```

A. Write Missing function here (if any)

MPI_Comm_size(MPI_COMM_WORLD, &size);

```
    if (rank == 0) {
        dataset = (int*)malloc(sizeof(int) * DATASET_SIZE);
        for (int i = 0; i < DATASET_SIZE; i++) {
            dataset[i] = i;
        }
    }
```

B. Write missing function here (if any)

// No function required here

```
local_size = DATASET_SIZE / size;
```

```
    local_data = (int*)malloc(sizeof(int) * local_size);

    MPI_Scatter(dataset, local_size, MPI_INT, local_data, local_size,
MPI_INT, 0, MPI_COMM_WORLD);
```

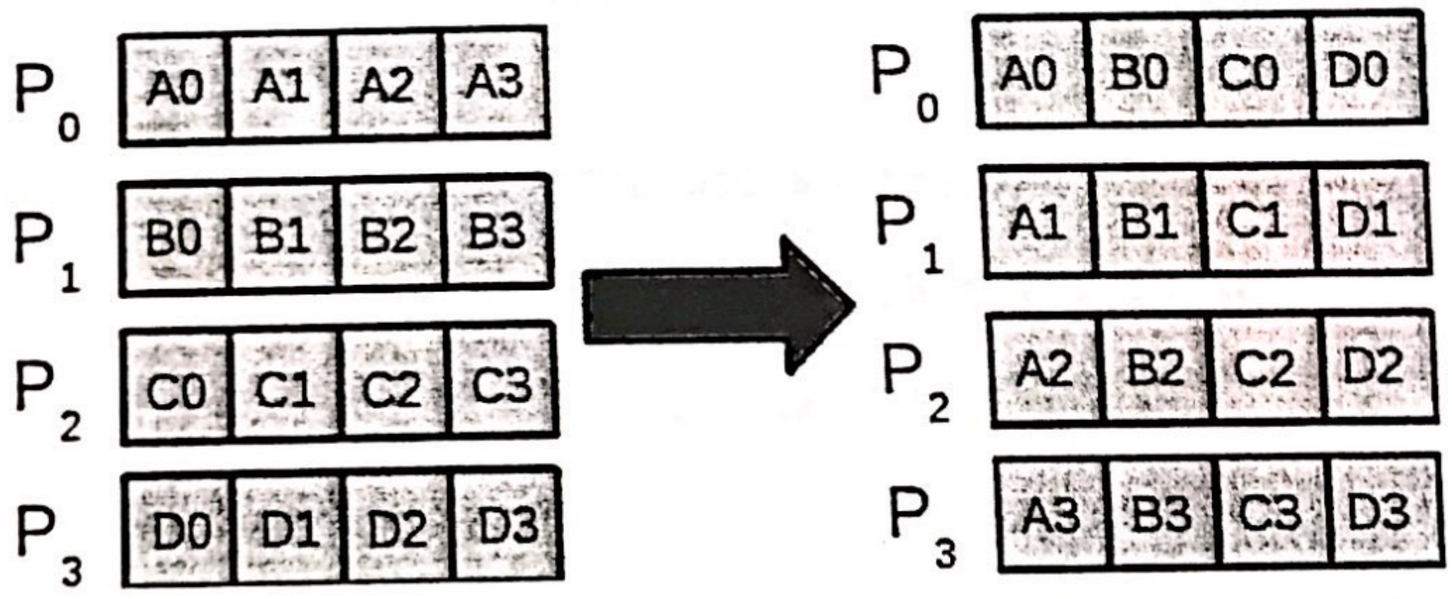
C. Write Missing Variable here (if any)

```
    printf("Process %d received %d elements.\n", rank, local_size );

    if (rank == 0) {
        free(dataset);
    }
    free(local_data);

    MPI_Finalize();
    return 0;
}
```

9

Question 3 [10 Marks]

Write a simple MPI program using **appropriate MPI COLLECTIVE function only** to illustrate the below scenario. Marks will be given for exact variables and complete function arguments.



Note: Please attempt in the space provided on the next page

Code Here:

```
#include <stdio.h>
# include <mpi.h>


int main(){
    int rank, size;
    MPI_Init(NULL, NULL);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    int N = 4;  // Array size

    int** arr = malloc(sizeof(int*)*N);
    for (int i=0; i<N; i++)
        arr[i] = malloc(sizeof(int)*N);
        for (int i=0; i<N; i++){
            arr[i][j] = rand() % 100;
        }
    }

    for (int i=0; i<N; i++)
        arr[i] = rand() % 100;  //random values

    // Using All to all for matrix Transposition
    MPI_Alltoall(arr, N, MPI_INT, arr, N, MPI_INT
                 , MPI_COMM_

    for (int i=0; i<N; i++)
        printf("%d\n", arr[i]); // printing val

    free(arr);
}
```

Question 4 [8 Marks]

Given the code below:

```
MPI_status *s;

if (pid  < 3 ) {
      MPI_Send(b1, 2, MPI_INT, pid+1, 0, MPI_COMM_WORLD); }

else {
      MPI_Send(b1, 2, MPI_INT, 0, 0, MPI_COMM_WORLD); }

if (pid  > 0 ) {
      MPI_Recv(b2, 2, MPI_INT, pid-1, 0, MPI_COMM_WORLD, s); }

else {
      MPI_Recv(b2, 2, MPI_INT, 3, 0, MPI_COMM_WORLD, s); }
```

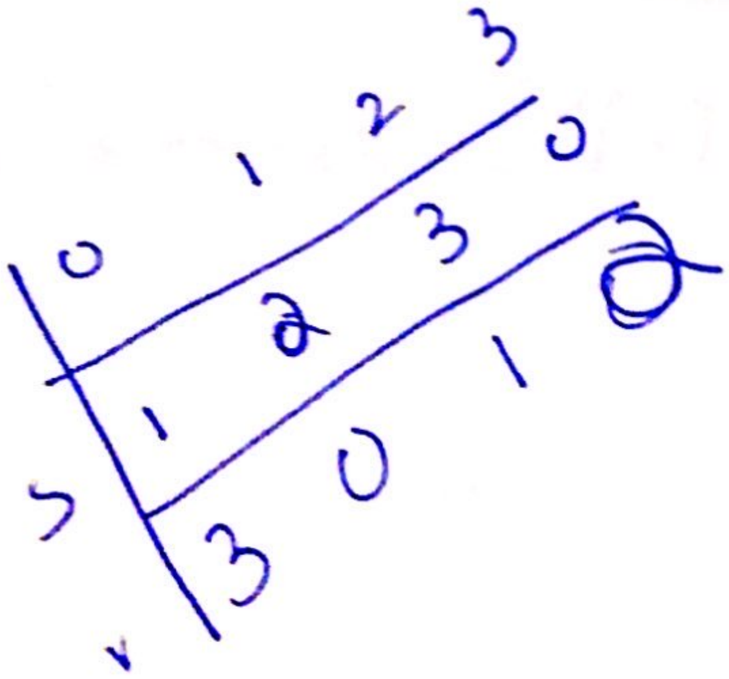
what will be the values of each process? Fill the boxes given below:

P0
b1 | 0 | 6 |
b2 | 5 | 7 |

P1
b1 | 2 | 4 |
b2 | 0 | 6 |

P2
b1 | 9 | 3 |
b2 | 2 | 4 |

P3
b1 | 5 | 7 |
b2 | 9 | 3 |



Question 5 [10 Marks]

What will be the output of the following code?

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <mpi.h>
int main(int argc, char* argv[])
{
    MPI_Init(&argc, &argv);

// Get number of processes and check that 3 processes are used
int size,i;
MPI_Comm_size(MPI_COMM_WORLD, &size);
if(size != 4)
{
    printf("This application is meant to be run with 4 processes.\n");
    MPI_Abort(MPI_COMM_WORLD, EXIT_FAILURE);
}

// Determine root's rank
int root_rank = 0;

// Get my rank
int my_rank;
MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);

switch(my_rank)
{
    case 0:
    {
        // Define my value
        int my_value;

        // Declare the buffer
        int buffer[9] = {100, 101, 102, 103,104,105,106,107,108};

        // Declare the counts
        int counts[4] = {1, 2, 3,1};

        // Declare the displacements
        int displacements[4] = {0, 2, 6,8};


        printf("\n");
        MPI_Scatterv(buffer, counts, displacements, MPI_INT, &my_value, 1,
MPI_INT,    root_rank, MPI_COMM_WORLD);

        break;
    }
    case 1:
```

```
        case 2:
        {
            // Declare my values
            int my_values[4];

            MPI_Scatterv(NULL, NULL, NULL, MPI_INT, my_values, 4, MPI_INT,
root_rank, MPI_COMM_WORLD);
            printf("Process : %d", my_rank);
            for(int j=0;j<4;j++)

            printf(" %d", my_values[j]);
            printf("\n");
            break;
    }
```

Output:

Process : 1 102 103 g g

Process : 2 106 107 108 g

// g is garbage value

There are multiple issues in above code,

(1) Size of disp, send counts is one greater than number of process.

(2) case 1: code is either missing

(3) If it is not missing the P1 and P2 will execute same code which will lead to send/recv count issue for P1 as it should recv 3 elements but receiving 4.

I am ignoring all above issues

Question 6 [8+8+12 Marks]

In this part You only must provide the Input and Output <key, value> for each function. Do not write pseudocode/code for the map reduce functions.

a. DISTINCT operator using Map-Reduce

The DISTINCT(X) operator is used to return only distinct (unique) values for datatype (or column) X in the entire dataset. As an example, for the following table A:

A.ID	A.ZIPCODE	A.AGE
1	12345	30
2	12345	40
3	78910	10
4	78910	10
5	78910	20

```
DISTINCT(A.ID) = (1, 2, 3, 4, 5)
DISTINCT(A.ZIPCODE) = (12345, 78910)
DISTINCT(A.AGE) = (30, 40, 10, 20)
```

Implement the DISTINCT(X) operator using Map-Reduce. Provide the algorithm pseudocode. You should use only one Map-Reduce stage, i.e. the algorithm should make only one pass over the data.

Suppose we take zipcode as input

Input (Key, Value)	Function	Output (Key, Value)
(Byte offset, zipcode) e.g (0, 12345) (5, 12345) (10, 78910) (15, 78910) (20, 78910)	Map	(zipcode, 1) e.g (12345, 1) (12345, 1) (78910, 1) (78910, 1) (78910, 1)
Not needed	Combine (if any)	Not needed
(zipcode, Grouped cnt) e.g (12345, [1,1]) (78910, [1,1,1])	Reduce	(zipcode, Empty) e.g (12345,) (78910,)

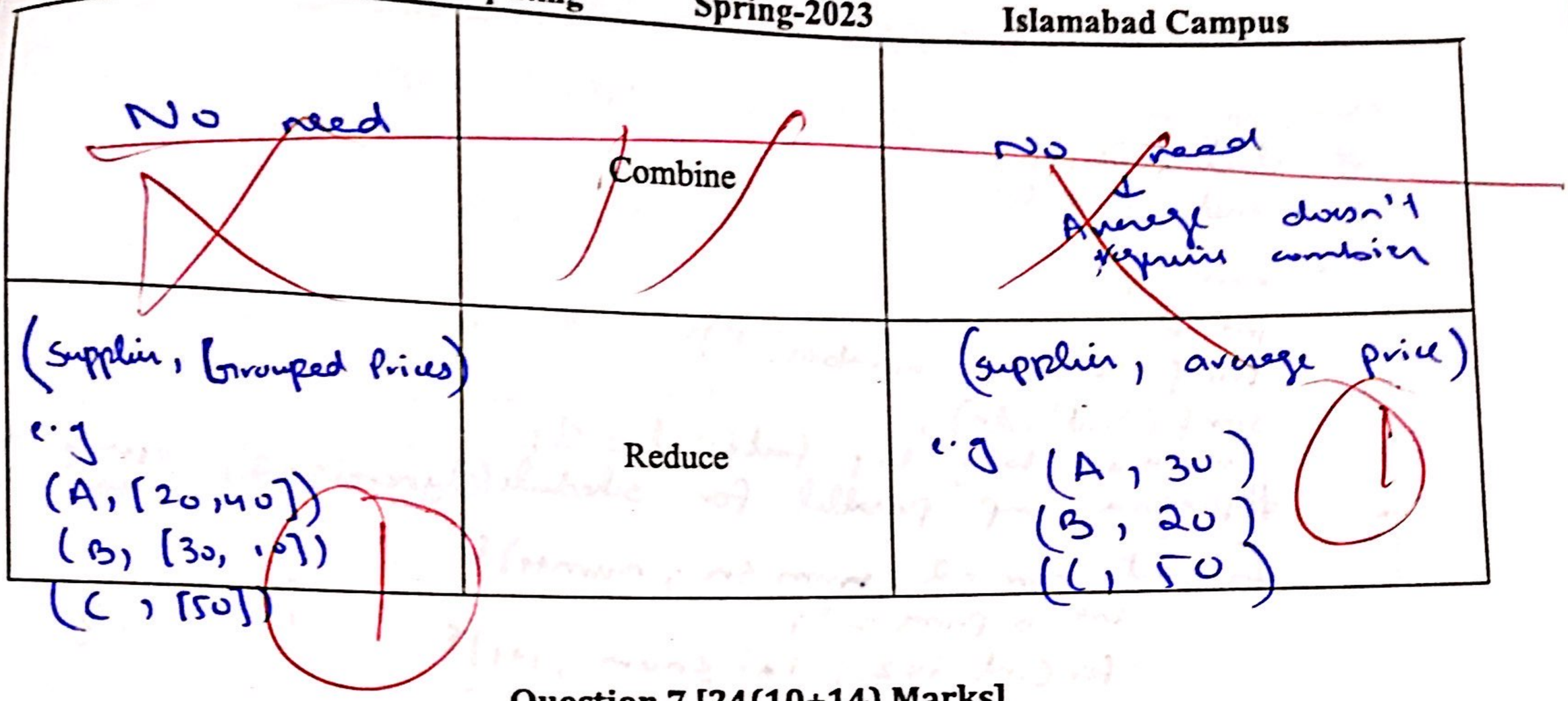
b. The SHUFFLE operator takes a dataset as input and randomly re-orders it. Assume that we have a function rand(m) that is capable of outputting a random integer between [1, m]. Implement the SHUFFLE operator using Map-Reduce.

Input (Key, Value)	Function	Output (Key, Value)
(index, record) e.g (0, 'A,B,C') (1, 'D,E,F') (2, 'G,H,I')	Map	(record, old Index) e.g ('A,B,C', 0) ('D,E,F', 1) ('G,H,I', 2)
Not needed	Combine (if any)	Not needed
(record, grouped Indexes) e.g ('A,B,C', [0]) ('D,E,F', [1]) ('G,H,I', [2])	Reduce	(record, new random Index) i.e (record, rand(size-of-dataset)) e.g ('A,B,C', 5) ('D,E,F', 0) ('G,H,I', 3)

c. Consider the checkout counter at a large supermarket chain. For each item sold, it generates a record of the form [ProductId, Supplier, Price]. Here, ProductId is the unique identifier of a product, Supplier is the supplier name of the product and Price is the sales price for the item. Assume that the supermarket chain has accumulated many terabytes of data over a period of several months. The CEO wants a list of suppliers, listing for each supplier the average sales price of items provided by the supplier. How would you organize the computation using the Map-Reduce computation model? You must optimize the solution using combiner function for this part.

Provide the Key value input and output of the following Functions.

Input (Key, Value)	Function	Output (Key, Value)
(Byte offset, 1 Record) e.g 0, '[1, A, 20]' 8, '[2, B, 30]' 17, '[3, A, 140]'	Map	(Supplier, Price) e.g (A, 20) (B, 30) (A, 140) (C, 50) (B, 10)



Question 7 [24(10+14) Marks]

a. Given below is a serial program to calculate the factorial of a number ***n*** using only the prime factors. The program decompose ***n*** into its prime factors and multiply them together to get the factorial.

```
#include <stdio.h>

int main() {
    int n;
    printf("Enter a number: ");
    scanf("%d", &n);


    unsigned long long factorial = 1;

    for (int num = 2; num <= n; num++) {
        int is_prime = 1;
        for (int i = 2; i * i <= num; i++) {
            if (num % i == 0) {
                is_prime = 0;
                break;
            }
        }
        if (is_prime) {
            factorial *= num;
        }
    }

    printf("Prime Factorial of %d is %llu\n", n, factorial);

    return 0;
}
```

Change the Program to do this in parallel using OPENMP with 5 threads.

```
#include <stdio.h>
# include <omp.h>

int main(){
    int n;
    printf("Enter a number: ");
    scanf("%d",&n);
    unsigned long long factorial = 1;
#pragma omp parallel for schedule(dynamic,2) //using dynamic for load balancing
    for(int num=2; num<=n; num++){
        int is_prime = 1;
        for(int i=2; i*i<=num; i++){
            if (num%i==0){
                is_prime = 0
                break;
            }
        }
#if(is_prime){
        factorial *= num;
    }
}
printf("Prime Factorial of %d is %llu\n",n,factorial);
return 0;
}
```

b. Given the code below, predict the output and fill up the provided table

```
#include <stdio.h>
#include <omp.h>

#define NUM_OF_COLUMNS 4
#define NUM_OF_ROWS(3 * (NUM_OF_COLUMNS - 1))

int whichThread[NUM_OF_ROWS][NUM_OF_COLUMNS];

void fillColumn(int j) {
  int i;
  #pragma omp for
  for (i = 0; i < NUM_OF_ROWS; i++)
    whichThread[i][j] = omp_get_thread_num();
}

int main( )
  {
    int i, j;

    for (i = 0; i < NUM_OF_ROWS; i++) // initialize the array
      for (j = 0; j < NUM_OF_COLUMNS; j++)
        whichThread[i][j] = -1;
```

```
#pragma omp parallel num_threads(NUM_OF_COLUMNS - 1)
fillColumn(0);

#pragma omp parallel
for num_threads(NUM_OF_COLUMNS - 1)
for (j = 1; j < NUM_OF_COLUMNS; j++)
  fillColumn(j);

 for (i = 0; i < NUM_OF_ROWS; i++) // print out the results
 {
   for (j = 0; j < NUM_OF_COLUMNS; j++)
     printf(" %2d ", whichThread[i][j]);
   printf("\n");
 }

  return 0;
}
```

Output: