National University of Computer & Emerging Sciences, Karachi
School of Computing (BSCS)
Midterm # 2 (Fall 2022)

SOLUTION

Course Code: CS3006	**Course Name: Parallel and Distributed Computing**
Instructor Name: Dr. Nausheen Shoaib, Mr. Danish, Dr. Nadeem Kafi	

Instructions:

- Attempt ALL questions. There are **3 questions on 2 pages**. All carry equal marks.
- **Do not write anything on the question paper.** Return by placing it inside your answer sheet.
- In case of any ambiguity, you may make assumptions without contradict any statement in the question paper.

Time: 60 Minutes. **Max Marks**: 15 Points (15 Marks)

Question 1: [CLO # 1] **[5 Points]**

a) How task interaction graph useful when 100s of task are running in a distributed memory system? **Justify your answer with the help of a brief computational scenario.** [1.5]

Task interactions graph (TIG) help identify how task running of different nodes interact with each other. Take example of sparse matrix multiplication with a vector where n is a large values e.g. 1000. If each row of the matrix is at different nodes, then a lot of communication time is saved by identifying matrix values that are needed to be sent or received by each nodes instead of all 1000 values. This same communication cost, thus saving completion time.

b) Suppose you are given a 1920 x 1080 pixel image where each pixel is a 16 bit values. If you are tasked to multiply each pixel value with an integer (e.g. x 2), a serial program will use a loop. We have studied that we can do various types of decompositions to parallelize computation. **Given brief explanation (3-4 sentences) of one decomposition that you think is most suitable and why?** [1.5]

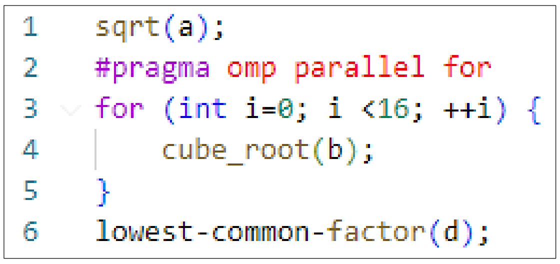
We can divide the data into partitions (input data decomposition) N times if we have N computer nodes, which multiply values in each partition in parallel thus reducing the time by a factor of N. Why?: Easy division of data into chunk and single program can run of different nodes. At node level, it is Single Program Multiple Data (data parallelism) where CPU with multiple cores or GPUs are used. In a distributed setup, large quantities of data (e.g. Terabytes are distributed among different nodes a large (~10K nodes) cluster and processed in parallel

c) Consider the problem of computing the frequency of a set of items (3-6 items), each called an itemsets, in a transaction database. The store (e.g. Imtaiz Store) wants to place them together on a prominent location. Suppose the transaction dataset size is estimated at 2-3 GB per day and not less than 50GB per month. In order to calculate the itemsets frequency, **list at least two type of decomposition and justify correct parallel processing.** [2]

i) Input data decomposition is most suitable (see part b above) as it will reduce overall completion time as it calculates results for each chunk and combine them to produce the overall result.
ii) Output data decomposition is also possible; however, it either needs all data at each nodes responsible to compute each element of the output or additional communication with other nodes in order to fetch data elements stored there

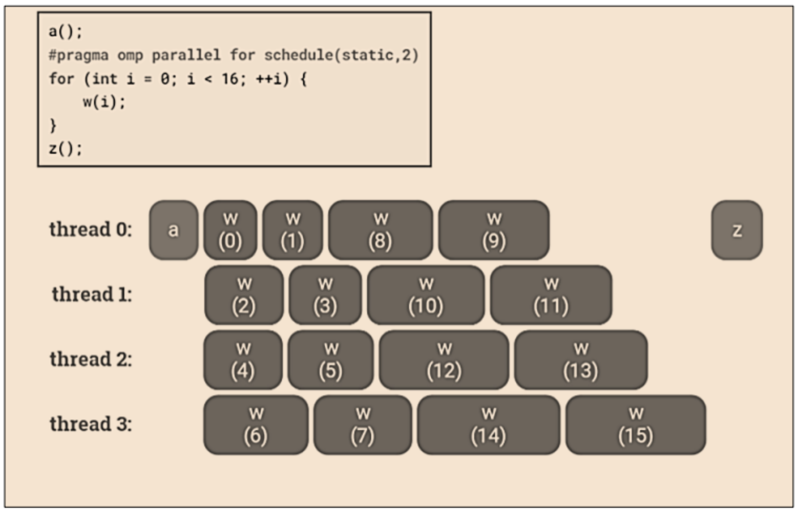
Question 2: [CLO # 1] **[5 Points]**

a) **Add static scheduling** in Figure 1. **Illustrate** how loop iterations are assignments to threads (assume b=i). [0.25 + 0.75]



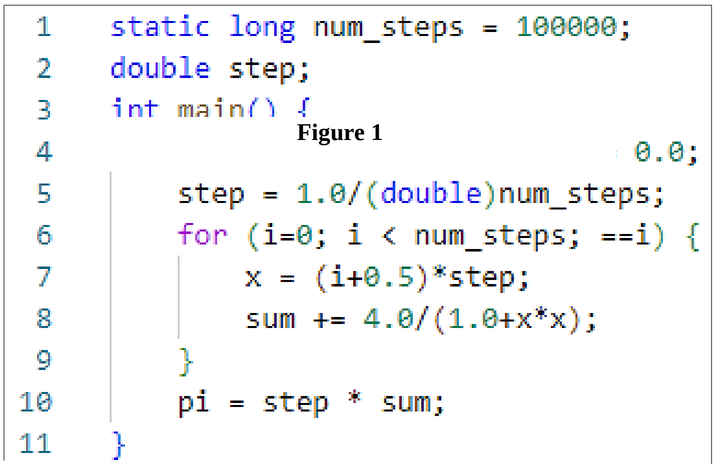
```
sqrt(a);
#pragma omp parallel for
for (int i=0; i <16; ++i) {
    cube_root(b);
}
lowest-common-factor(d);
```

Figure 1



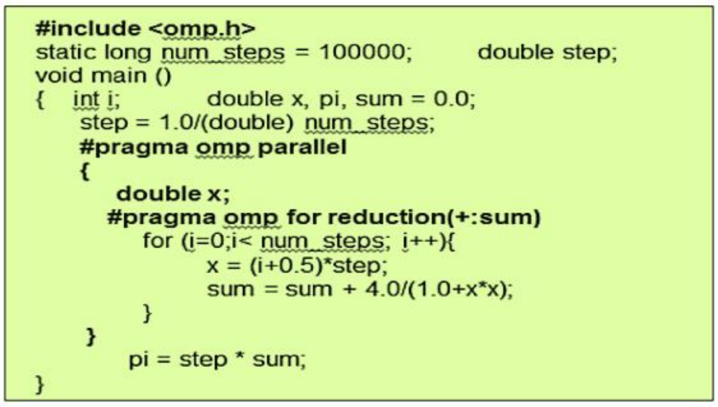
```
a();
#pragma omp parallel for schedule(static,2)
for (int i = 0; i < 16; ++i) {
    w(i);
}
z();
```

b) **Insert OpenMP pragmas and clauses in the serial C code** shown in Figure 2. Write only OpenMP code in your answer book having line numbers in points e.g. if you want to add OpenMP clause between line 2 and 3 use 2.1, 2.2 etc. *Note. Specify all necessary clauses in your pragmas.* **[2]**



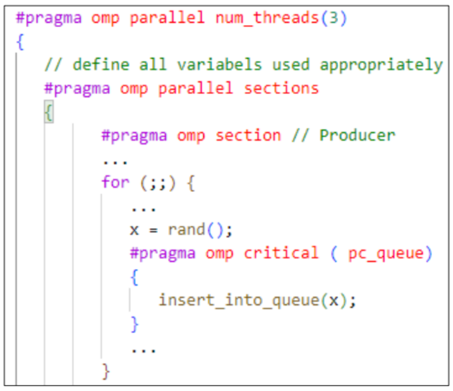
```
1   static long num_steps = 100000;
2   double step;
3   int main() {
4                                  0.0;
5       step = 1.0/(double)num_steps;
6       for (i=0; i < num_steps; ==i) {
7           x = (i+0.5)*step;
8           sum += 4.0/(1.0+x*x);
9       }
10      pi = step * sum;
11  }
```

Figure 2

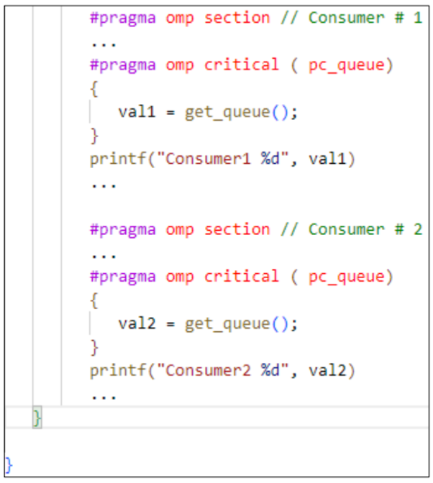


```
#include <omp.h>
static long num_steps = 100000;        double step;
void main ()
{   int i;          double x, pi, sum = 0.0;
    step = 1.0/(double) num_steps;
    #pragma omp parallel
    {
        double x;
        #pragma omp for reduction(+:sum)
          for (i=0;i< num_steps; i++){
                x = (i+0.5)*step;
                sum = sum + 4.0/(1.0+x*x);
          }
    }
        pi = step * sum;
}
```

c) **Write OpenMP snippet which creates three parallel threads.** One thread is assigned computation which continuously generates a double value [use rand()] and inserts it in a queue [use void insert_queue(int a)]. The other two run similar computations such each of them concurrently pulls data from the same queue [int get_queue()] and display it for the user. *Note: Only focus on writing correct OpenMP code and use queue functions, as a block box, to manage the queue.* **[2]**



```
#pragma omp parallel num_threads(3)
{
    // define all variabels used appropriately
    #pragma omp parallel sections
    {
            #pragma omp section // Producer
            ...
            for (;;) {
                ...
                x = rand();
                #pragma omp critical ( pc_queue)
                {
                    insert_into_queue(x);
                }
                ...
            }
```

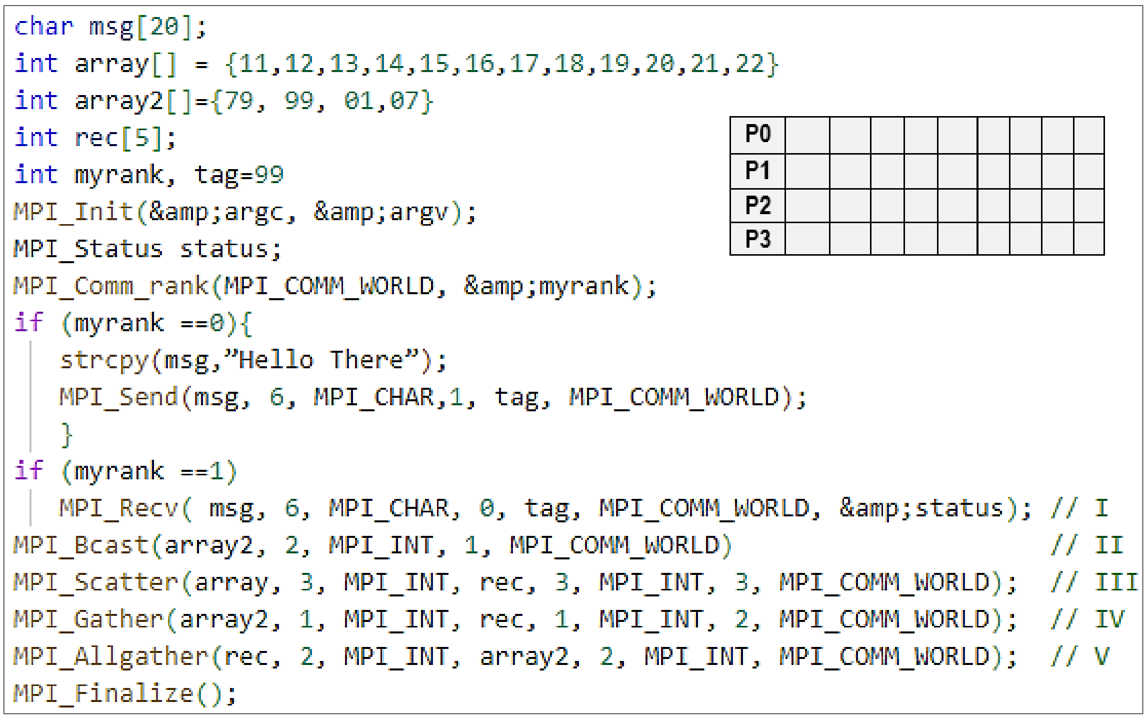


```
            #pragma omp section // Consumer # 1
            ...
            #pragma omp critical ( pc_queue)
            {
                val1 = get_queue();
            }
            printf("Consumer1 %d", val1)
            ...

            #pragma omp section // Consumer # 2
            ...
            #pragma omp critical ( pc_queue)
            {
                val2 = get_queue();
            }
            printf("Consumer2 %d", val2)
            ...
    }
}
```

Question 3: [CLO # 1] **[5 Points]**

a) Suppose the MPI code snippet as shown in Figure 3 is executing using four processes (P0, P1, P2, and P3). **Draw the <u>receive buffer</u> of each process (see table shown as part of Figure) for each numbered call (I, II, III, IV, V).** *Note: Marks will be awarded on correct values received after each call and not drawing a scaled table.***[2.5]**



```
char msg[20];
int array[] = {11,12,13,14,15,16,17,18,19,20,21,22}
int array2[]={79, 99, 01,07}
int rec[5];
int myrank, tag=99
MPI_Init(&amp;argc, &amp;argv);
MPI_Status status;
MPI_Comm_rank(MPI_COMM_WORLD, &amp;myrank);
if (myrank ==0){
   strcpy(msg,"Hello There");
   MPI_Send(msg, 6, MPI_CHAR,1, tag, MPI_COMM_WORLD);
   }
if (myrank ==1)
   MPI_Recv( msg, 6, MPI_CHAR, 0, tag, MPI_COMM_WORLD, &amp;status); // I
MPI_Bcast(array2, 2, MPI_INT, 1, MPI_COMM_WORLD)                  // II
MPI_Scatter(array, 3, MPI_INT, rec, 3, MPI_INT, 3, MPI_COMM_WORLD);  // III
MPI_Gather(array2, 1, MPI_INT, rec, 1, MPI_INT, 2, MPI_COMM_WORLD);  // IV
MPI_Allgather(rec, 2, MPI_INT, array2, 2, MPI_INT, MPI_COMM_WORLD);  // V
MPI_Finalize();
```

P0									
P1									
P2									
P3									

Figure 3

MPI_Recv(msg, 6, MPI_CHAR, 0, tag, MPI_COMM_WORLD, &status); // I

P0									
P1	H	e	l	l	o				
P2									
P3									

MPI_Bcast(array2, 2, MPI_INT, 1, MPI_COMM_WORLD); // II

P0	79	99							
P1	79	99							
P2	79	99							
P3	79	99							

MPI_Scatter(array, 3, MPI_INT, rec, 3, MPI_INT, 3, MPI_COMM_WORLD); // III

P0	11	12	13						
P1	14	15	16						
P2	17	18	19						
P3	20	21	22						

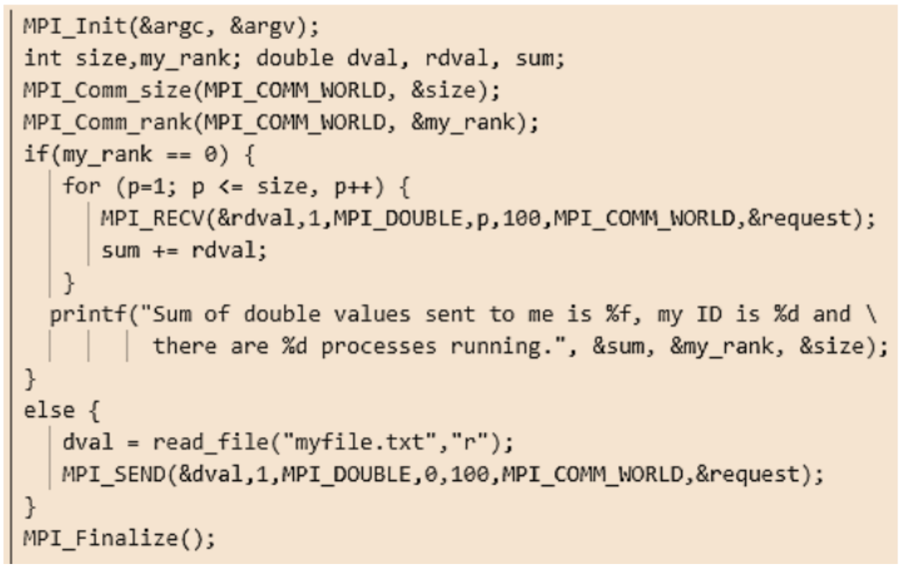
MPI_Gather(array2, 1, MPI_INT, rec, 1, MPI_INT, 2, MPI_COMM_WORLD); // IV

P0									
P1									
P2	79	79	79	79					
P3									

MPI_Allgather(rec, 2, MPI_INT, array2, 2, MPI_INT, MPI_COMM_WORLD); // V

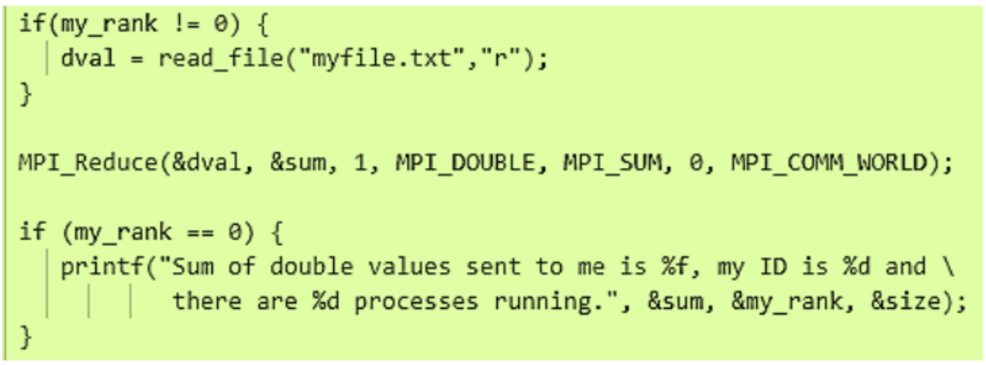
P0	11	12	14	15	79	79	20	21	
P1	11	12	14	15	79	79	20	21	
P2	11	12	14	15	79	79	20	21	
P3	11	12	14	15	79	79	20	21	

b) Write MPI snippet where each slaves process reads one double value from a local text file [use double read_file(myfile), which returns -1 for end-of-file.] and sent it to a master process. The master prints the message → `Sum of double values sent to me is 38746528.09, my ID is 100 and there are 3419 processes running`. *Note: You are allowed to assume values, which are not given in the question*. **[2.5]**



```
MPI_Init(&argc, &argv);
int size,my_rank; double dval, rdval, sum;
MPI_Comm_size(MPI_COMM_WORLD, &size);
MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
if(my_rank == 0) {
    for (p=1; p <= size, p++) {
        MPI_RECV(&rdval,1,MPI_DOUBLE,p,100,MPI_COMM_WORLD,&request);
        sum += rdval;
    }
   printf("Sum of double values sent to me is %f, my ID is %d and \
            there are %d processes running.", &sum, &my_rank, &size);
}
else {
    dval = read_file("myfile.txt","r");
    MPI_SEND(&dval,1,MPI_DOUBLE,0,100,MPI_COMM_WORLD,&request);
}
MPI_Finalize();
```

OR



```
if(my_rank != 0) {
    dval = read_file("myfile.txt","r");
}

MPI_Reduce(&dval, &sum, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);

if (my_rank == 0) {
    printf("Sum of double values sent to me is %f, my ID is %d and \
            there are %d processes running.", &sum, &my_rank, &size);
}
```

----------(X)----------