National University of Computer and Emerging Sciences

Lahore Campus

Parallel and Distributed Computing (CS3006)

Sessional-II Exam

Total Time (Hrs.): 1

Total Marks: 60

Total Questions: 4

Date: April 09, 2025

Course Instructor(s)

Dr. Abdul Qadeer

Mr. Danyal Farhat

Mr. Raja Muzammil Muneer

Ms. Saba Tariq

Dr. Syed Mohammad Irteza

Dr. Zeeshan Ali Khan

_____ _____ _____

Roll No Section Student Signature

Instructions / notes:

1. Attempt all questions on the answer sheet.
2. If a question is ambiguous, note it, assume clarifying details, and solve it.
3. Show all work with your answer; do not use a separate rough sheet.

CLO 2: Implement different parallel and distributed programming paradigms and algorithms using Message-Passing Interface (MPI) and OpenMP.

Q1a: [11 marks] Show the output of the following program.

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

int main(void) {
    int nums[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
    omp_set_num_threads(3);

    #pragma omp parallel for schedule(static, 3)
    for (int j = 0; j < 10; j++)
    {
        nums[j] *= (j + 3);
        int thread_id = omp_get_thread_num();
        printf("At thread: %d iteration: %d\n", thread_id, j);
    }

    for (int i = 0; i < 10; i++)
    {
        printf("%d ", nums[i]);
    }
    printf("\n");

    return 0;
}
```

Q1b: [1 mark] Assume that above code is compiled and run on a machine where the OpenMP library is installed and configured correctly. The code also makes another implicit assumption. What is that assumption, and on which line does it occur?

Q1c: [3 marks] How would you modify the code to explicitly verify this assumption? Assume you want the program to exit if the assumption is false.

Solution:

Q1a: Following is one run of the above code. The order of printed statements is not important (except the last one). The important thing is that what loop iteration each thread is processing.

```
At thread: 2 iteration: 6
At thread: 2 iteration: 7
At thread: 2 iteration: 8
At thread: 0 iteration: 0
At thread: 0 iteration: 1
At thread: 0 iteration: 2
At thread: 0 iteration: 9
At thread: 1 iteration: 3
At thread: 1 iteration: 4
At thread: 1 iteration: 5
3 8 15 24 35 48 63 80 99 120
```

Q1b: On line 6, omp_set_num_threads(3) is just a request to get three threads from the OpenMP runtime. There is no guarantee that the runtime will always be able to fulfil it. Under some cases, it might be the case that we get just 2 threads.

Q1c: Code will need to change as follows:

```
 7      omp_set_num_threads(3);
 8
 9      #pragma omp parallel
10      #pragma omp single
11      {
12              int actual_threads = omp_get_num_threads();
13              if (actual_threads != 3)
14              {
15                  printf("Error: Expected 3 threads but got %d. Exiting.\n", actual_threads);
16                  exit(1);
17              }
18      }
19      #pragma omp parallel for schedule(static, 3)
20      for (int j = 0; j < 10; j++)
21      {
```

CLO 3: Perform analytical modelling, dependence, and performance analysis of parallel algorithms and programs.

Q2a: [11 marks] Below is the serial code for using binary search to find a target integer in a large, ascending-sorted array. Modify the code to use OpenMP for parallelization so that it is syntactically and logically correct, and achieves a speed-up compared to the serial version.

```
1   int binary_search(const int arr[], int size, int target) {
2       int low = 0, high = size - 1;
3       while (low <= high) {
4           int mid = low + (high - low) / 2;
5           if (arr[mid] == target)
6               return mid;
7           else if (arr[mid] < target)
8               low = mid + 1;
9           else
10              high = mid - 1;
11      }
12      return -1;
13  }
```

Q2b: [2 marks] What is the time complexity of your parallel code if there were ***n*** elements in the array and your code used ***p*** number of processors? Assume that n >>> p (i.e. number of elements in the array are way too many as compared to available number of processors).

Q2c: [2 marks] What speed-up is expected from your code as compared to the serial code? Are you satisfied with the speed-up?

Solution:

Q2a: The nature of binary search code is inherently serial --- on the base of the middle element we decide to go right or left. So what we can do is divide the large array into n/p pieces where p are the available processors and let each processor do the binary search in its part. Last processor might have less than n/p elements and students need to tackle that case.

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

int parallel_binary_search(const int arr[], int size, int target) {
    int result = -1;
    #pragma omp parallel
    {
        int tid = omp_get_thread_num();
        int num_threads = omp_get_num_threads();
        int chunk_size = size / num_threads;
        int start = tid * chunk_size;
        int end = (tid == num_threads - 1) ? size - 1 : start + chunk_size - 1;
        int low = start, high = end, local_result = -1;
        while (low <= high && local_result == -1) {
            int mid = low + (high - low) / 2;
            if (arr[mid] == target) {
                local_result = mid;
            } else if (arr[mid] < target) {
                low = mid + 1;
            } else {
                high = mid - 1;
            }
        }
        if (local_result != -1) {
            #pragma omp critical
            {
                result = local_result;
            }
        }
    }
    return result;
}
```

Q2b: Time complexity = O(log(n/p))

Q2c: Speed-up = O(log(n)) / O(log(n/p)) ~= log(n) / log(n/p) = log(n) / [log(n) – log(p)] = 1 / [1 – log(p)/log(n)]

As n approaches infinity, speed-up approaches 1
That means for n >>> p, our parallelization has very limited impact on speed-up, and we could have used the serial code instead. And we have ignored any parallelization overhead as well.

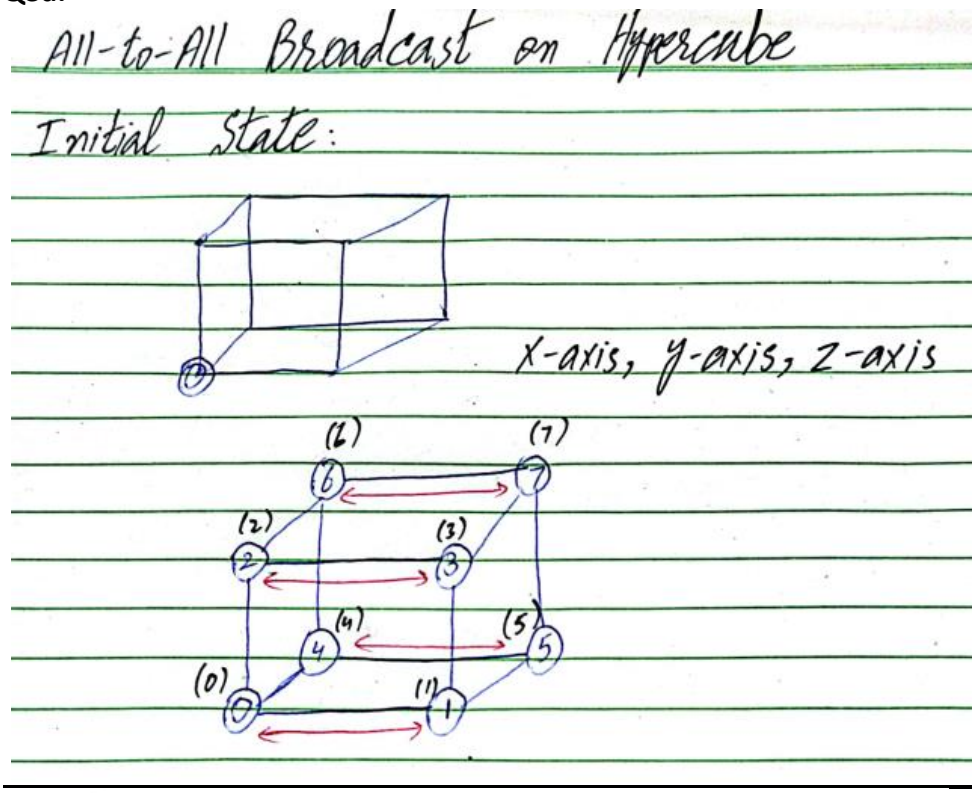
CLO 3: Perform analytical modelling, dependence, and performance analysis of parallel algorithms and programs.

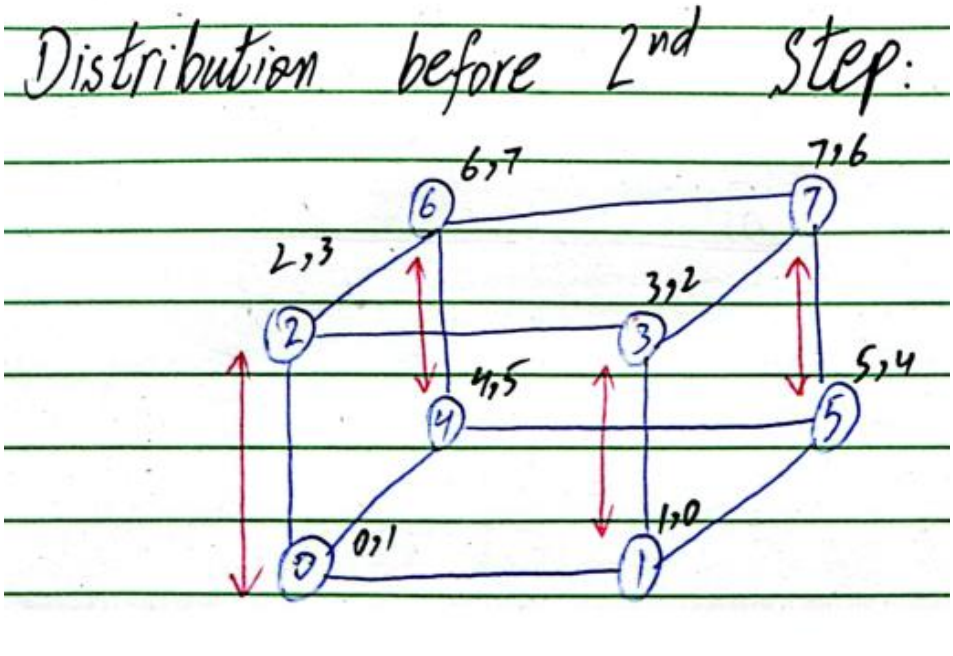
Q3a: [10 marks] Perform *all-to-all broadcast* on a 3-D hypercube structure. Assume each node initially has a message same as its ID (for example, node 0 has message 0, node 1 has message 1 and so on). Write all the steps involved and final state.

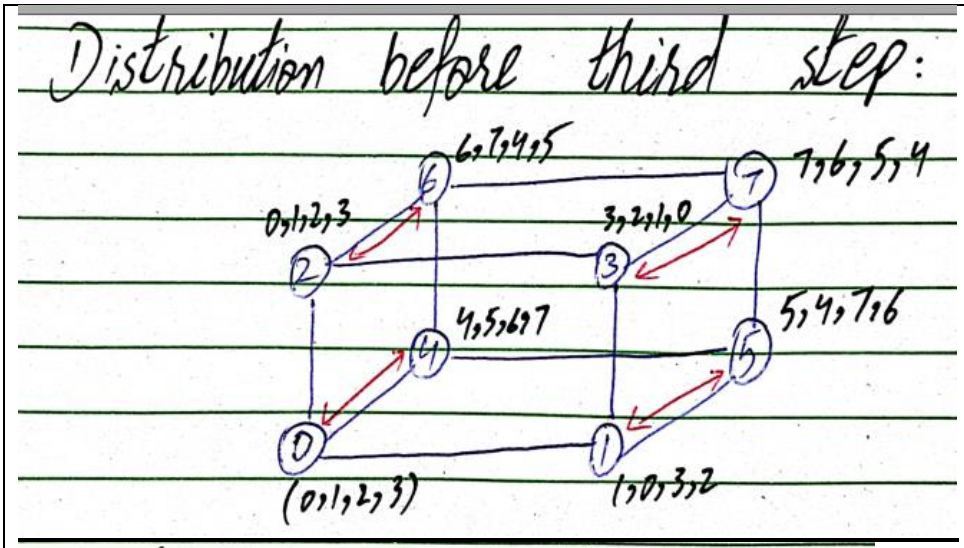
Q3b: [5 marks] If the size of the messages each node started with was of ***4 words,*** what will be the communication cost of your algorithm above to perform all-to-all broadcast?

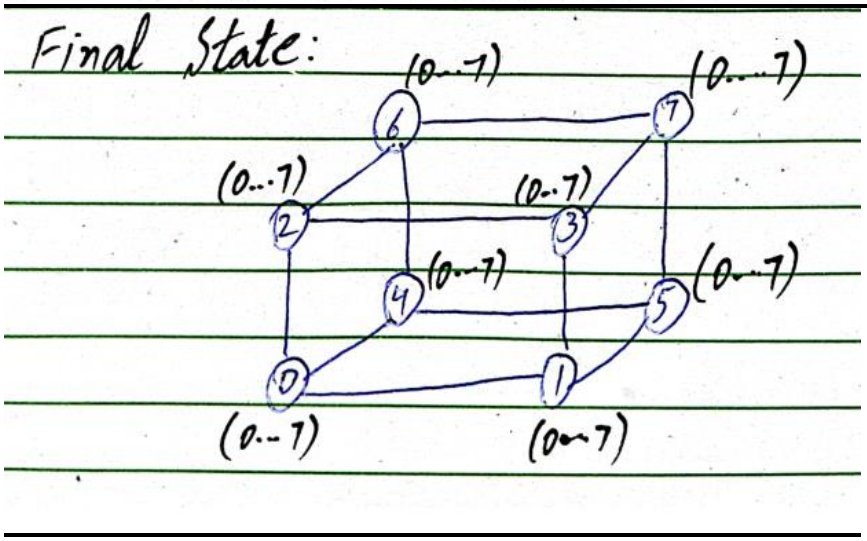
Solution:

Q3a:









Q3b: We need three steps for the algorithm to converge. Overall message sizes will increase from 4 words (in step 1) to 8 words (in step 2) to 16 words (in step 3).

Communication cost = $(3*t_s + 4*7*t_w)$ [Correct]

Communication cost = $(3*t_s + 4*28*t_w)$ [wrong]
Where t_s is the start-up cost
t_w is the cost of transmitting w words

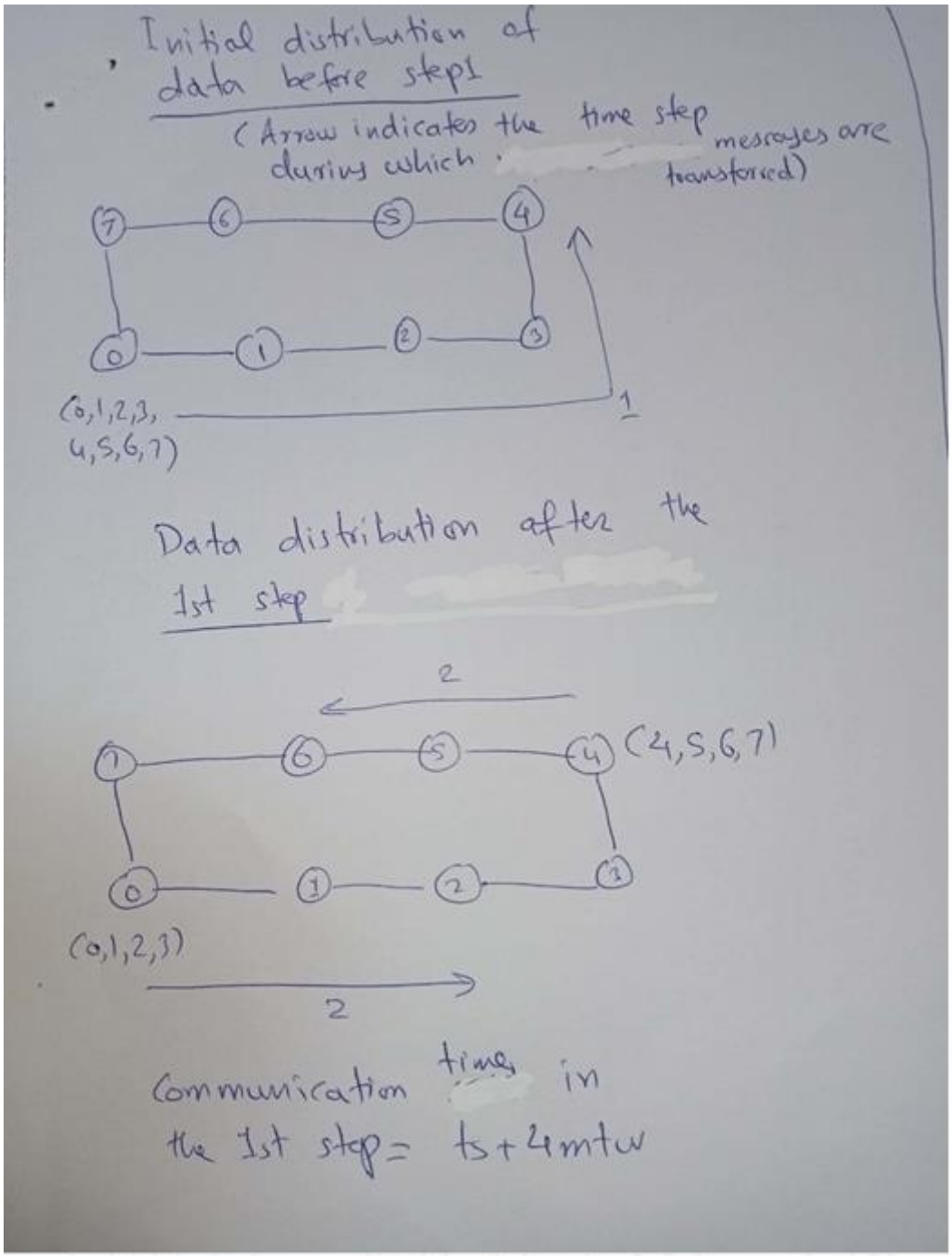
CLO 3: Perform analytical modelling, dependence, and performance analysis of parallel algorithms and programs.

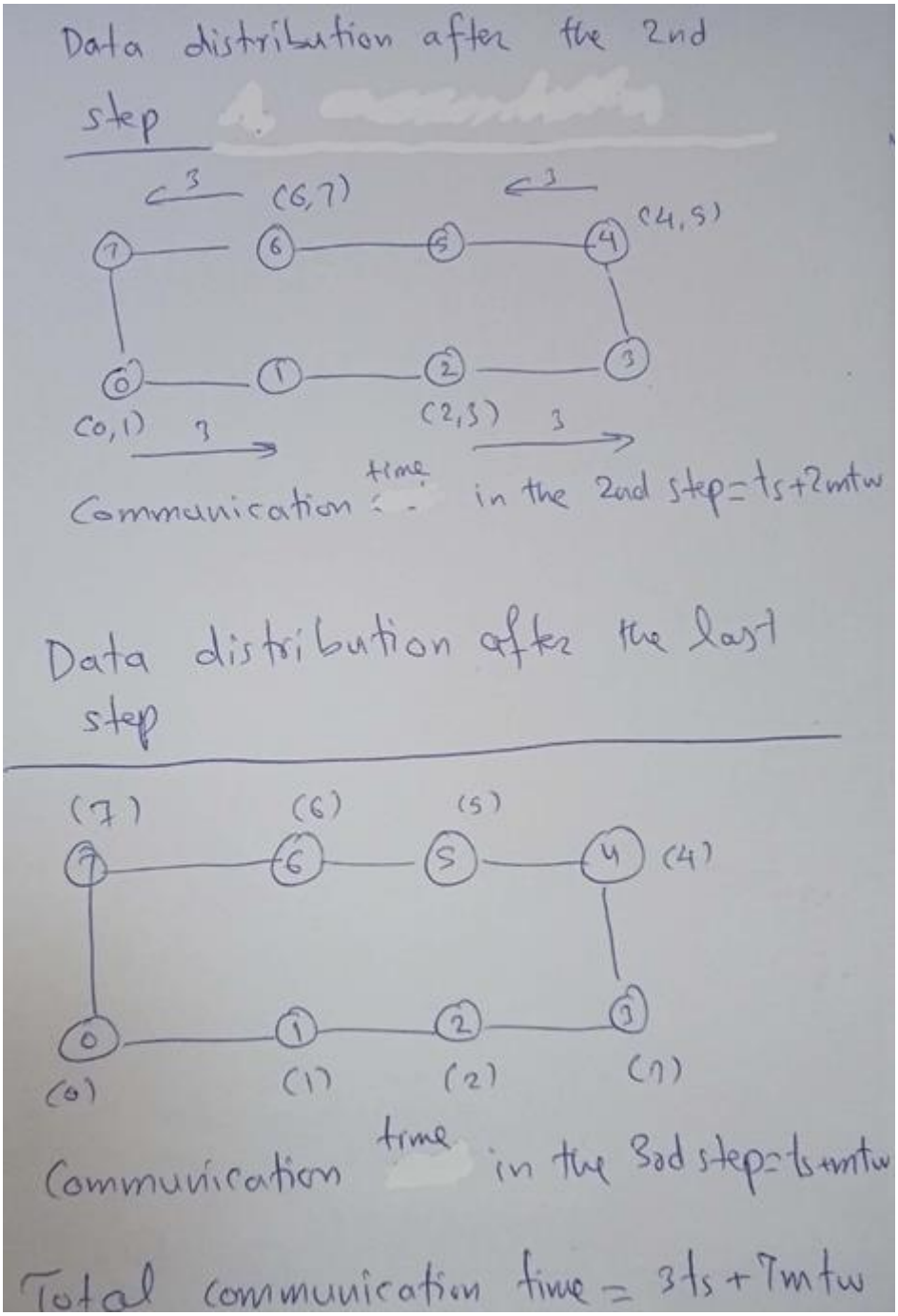
Q4a: [10 marks] Give the procedure for *one-to-all personalized communication (Scatter operation)* on a ring with 8 nodes. Let's assume node with ID 0 initiates this operation and have messages numbers 0, 1, 2, …,7 such that message with a specific number is intended for the node with that ID (for example message 6 is for the node with ID 6).

Q4b: [5 marks] What will be the communication cost of your algorithm above to perform one-to-all personalized communication? Assume that each message has size ***m-words*** when we started the algorithm.

Solution:

Q4a and Q4b:




Data distribution after the 2nd step
3
(6,7)
3
(4,5)
7
6
5
4
0
1
2
3
(0,1)
3
(2,3)
3
Communication time in the 2nd step = ts + 2mtw
Data distribution after the last step
(7)
(6)
(5)
(4)
7
6
5
4
0
1
2
3
(0)
(1)
(2)
(7)
Communication time in the 3rd step = ts + mtw
Total communication time = 3ts + 7mtw