National University of Computer and Emerging Sciences

Lahore Campus

Parallel and Distributed Computing (CS3006)

Sessional-II Exam

Total Time (Hrs): 1

Total Marks: 100

Total Questions: 5

Date: April 5th 2024

Course Instructor(s)

Dr. Abdul Qadeer

Mr. Danyal Farhat

Ms. Namra Absar

Dr. Rana Asif Rehman

Dr. Zeeshan Ali Khan

_____ _____ _____

Roll No Section Student Signature

Do not write below this line

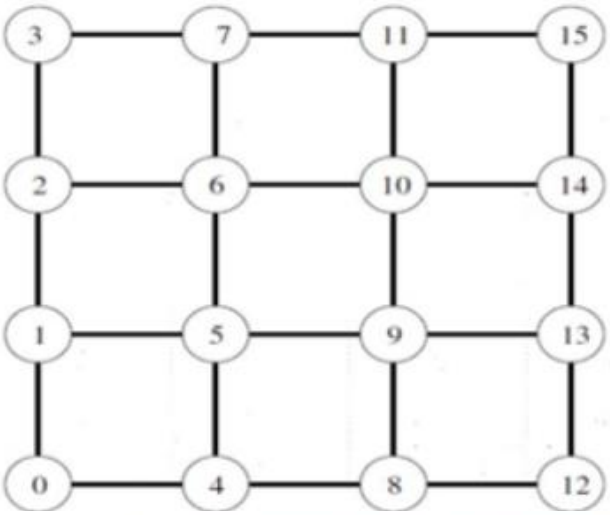
Instruction/Notes

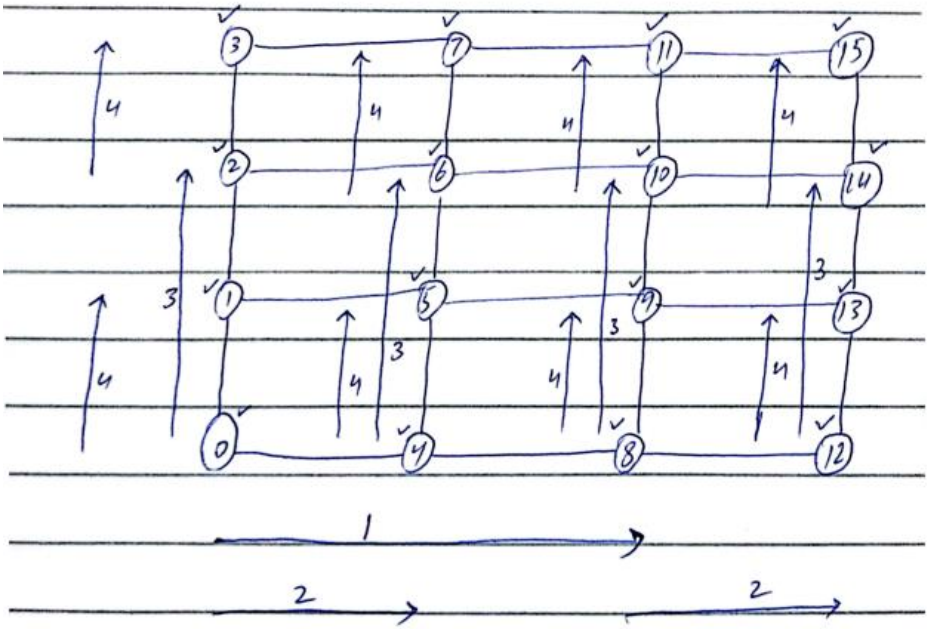
- If you find any ambiguity in a question, you can make your own assumption and answer the question accordingly by stating your assumption.

CLO # 3: Analytical modeling and performance evaluation of parallel programs

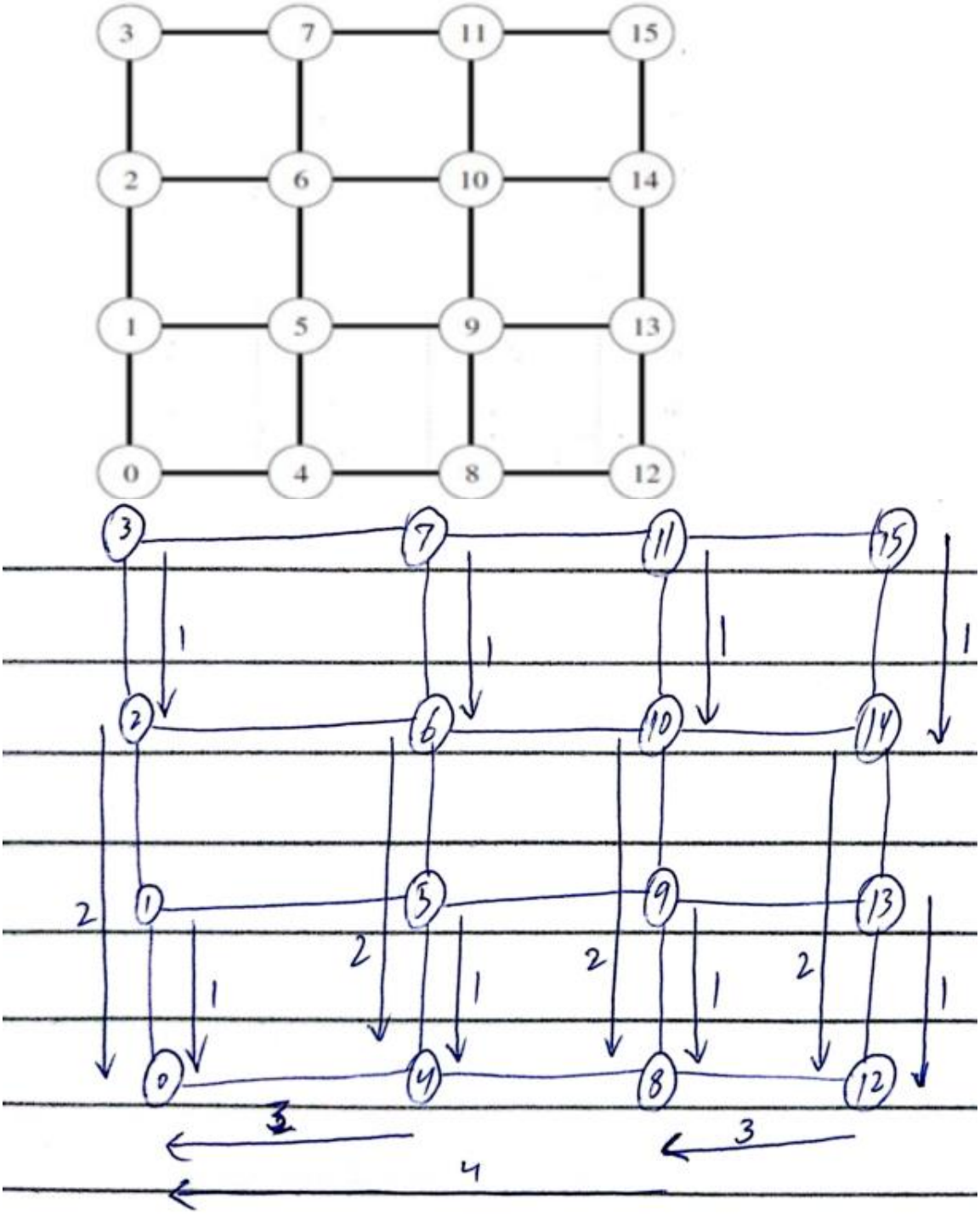
Q 1.

a) [7.5 marks] Perform one-to-all broadcast on a 16-node mesh below. Node 0 is the source of the broadcast. Add message transfer steps on the figure, each of which is shown by a numbered, dotted arrow from the source of the message to its destination. The number on the arrow indicates the time step during which the message is transferred. Use recursive doubling (source sends a message to a selected processor, we now have two independent problems defined over halves of machines). What will be the cost for this broadcast?





b) [7.5 marks] After applying one-to-all broadcast every node now has the same message. Your task is to apply all-to-one reduction on the same 16-node mesh but this time your destination node is 0. You have to label the 16 node mesh diagram again for this reduction and show each step. Use the recursive doubling technique for this problem.



CLO # 3: Analytical modeling and performance evaluation of parallel programs

Q2. [10 marks] Derive the equation for time taken by the All-to-All Broadcast on a 2D mesh having *M* rows and *N* columns.

Solution:

On a mesh, the first phase of all-to-all broadcast happens on each row and it concludes in time $(t_s+t_w m)(N-1)$. Where, m is the message size in words, t_w is the per-word transfer time, and t_s is the startup time. Then this process happens on each columns and number of nodes involved in this step are M, but size of message is now $m * N$. Therefore, this phase takes $(t_s+t_w m N)(M-1)$ time to complete. The time for the entire all-to-all broadcast on the mesh is sum of the times spent in individual phase, which is:

$T = (t_s+t_w m)(N-1) + (t_s+t_w m N)(M-1)$

$=t_s (N+M-2) + t_w m (NM – 1)$

CLO # 2: Code and analyze complex problems of shared memory systems with OpenMP

Q3: A bank with 30 Million savings accounts wants to calculate Zakat on the eve of Ramadan. To speed-up the process bank rents a server named m7g-metal form AWS that has 64 cores and 256 GB RAM. Following is the serial code bank is currently using:

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

#define NUM_ACCOUNTS 32000000
#define ZAKAT_THRESHOLD 135179.0  // Zakat threshold in Pakistani rupees for year 2024
#define ZAKAT_RATE 0.025          // Zakat rate as a decimal

int main() {
  float *accounts = (float *)malloc(NUM_ACCOUNTS * sizeof(float));  // Allocate array from heap

  // Fill the array with random account balances for our purposes. Bank will load it from DB.
  for (int i = 0; i < NUM_ACCOUNTS; i++) {
    accounts[i] = rand() % 500000 + 5000;  // Random balances between 5000 and 550000
  }

  // Calculate zakat ← You ONLY need to parallelize the following code
  for (int i = 0; i < NUM_ACCOUNTS; i++) {
    if (accounts[i] > ZAKAT_THRESHOLD) {
      float zakat = accounts[i] * ZAKAT_RATE;
      accounts[i] -= zakat;
      printf("Zakat deducted for account %d: %.2f rupees\n", i, zakat);
    }
  }

  free(accounts);  // Free the allocated memory
```

```
  return 0;
}
```

(a) [10 Marks] Parallelize the "Calculate zakat" part of the above code using OpenMP work sharing directive only. Your code must be correct and as fast as possible. (Please don't use SIMD instructions or directives for this question.)

Following is one correct code:

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

#define NUM_ACCOUNTS 32000000
#define ZAKAT_THRESHOLD 135179.0  // Zakat threshold in Pakistani rupees for year 2024
#define ZAKAT_RATE 0.025          // Zakat rate as a decimal

int main() {
    float *accounts = (float *)malloc(NUM_ACCOUNTS * sizeof(float));  // Allocate array from heap

    // Fill the array with random account balances (replace with DB loading)
    for (int i = 0; i < NUM_ACCOUNTS; i++) {
        accounts[i] = rand() % 500000 + 5000;  // Random balances between 5000 and 550000
    }

    // Calculate zakat with parallelization and atomic updates
    #pragma omp parallel num_threads(64)  // Use 64 threads for parallelization
    {
        #pragma omp for schedule(static)  // Static scheduling for better load balancing
        // if students don't write schedule(static) that is still okay
        for (int i = 0; i < NUM_ACCOUNTS; i++) {
            if (accounts[i] > ZAKAT_THRESHOLD) {
                float zakat = accounts[i] * ZAKAT_RATE;
                #pragma omp atomic update
                accounts[i] -= zakat;  // Atomic update to avoid race conditions
                printf("Zakat deducted for account %d: %.2f rupees\n", i, zakat);
            }
        }
    }

    free(accounts);  // Free the allocated memory
    return 0;
}
```

(a) [2 Marks] What is expected speedup after correct and efficient OpenMP parallelization?

Speed-up of 64X is expected because all 64 threads will be working without contending with each other.

CLO # 2: Code and analyze complex problems of shared memory systems with OpenMP

Q4: [8 marks] Explain briefly the output of this code?

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

int main() {
    int total_and = 1; // Initialize total with some value
    int total_or = 0;  // Initialize total with some value

    #pragma omp parallel for reduction(&&:total_and) reduction(||:total_or)
    for (int i = 0; i < 8; i++) {
        int thread_value = (i % 2 == 0);
        // Set thread_value to 1 for even threads, 0 for odd threads
        total_and = total_and && thread_value; // Perform logical AND reduction
        total_or = total_or || thread_value;  // Perform logical OR reduction
    }

    printf("Total (AND): %d\n", total_and);
    printf("Total (OR): %d\n", total_or);

    return 0;
}
```

Answer:
Total (AND): 0
Total (OR): 1