/ National University of Computer and Emerging Sciences 0361

Lahore Campus

Parallel and Distributed Computing (CS3006)

Final Exam

Date: May 25th 2024

Course Instructor(s)

Dr. Abdul Qadeer

Mr. Danyal Farhat

Ms. Namra Absar

Dr. Rana Asif Rehman

Dr. Zeeshan Ali Khan

Total Time (Hrs):	3
Total Marks:	72
Total Questions:	8

[redacted]

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 # 1: Learn about parallel and distributed computers

Q1.

(a) [2.5 marks]: Write the Amdahl's law. Clearly mention what are each of the symbols in the formula, and what are the units of those symbols (on both sides of the equality). Make sure overall units match on both sides of the equality.

(b) [2.5 marks]: If a program needs 20 hours to complete using a single processor, but a one-hour portion of the program cannot be parallelized, therefore only the remaining 19 hours' execution time can be parallelized. What is the minimum possible execution time? If you need to take any assumptions, write them clearly.2

(c) [5 marks]: Assume that we are given a serial program which is split into four consecutive parts, whose percentages of execution time are $p1 = 0.11$, $p2 = 0.18$, $p3 = 0.23$, and $p4 = 0.48$ respectively. Then we are told that the 1st part is not sped up, while the 2nd part is sped up 5 times, the 3rd part is sped up 20 times, and the 4th part is sped up 1.6 times. What is the overall speedup?

$0.11 + 0.036 + 0.115 + 0.3$

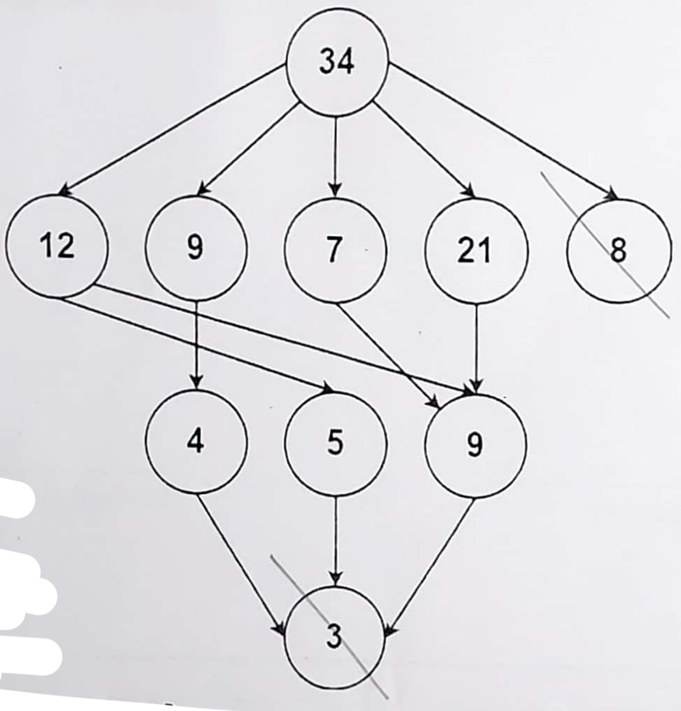
2.186

National University of Computer and Emerging Sciences
Lahore Campus

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

Q2 [2+2+2+2 marks]: For the task dependency graph given below, determine the critical path length, maximum degree of concurrency and average degree of concurrency. Furthermore, map the task graph to three processes to minimize the idling time of the application.

54 ~~55~~



[redacted]

CLO # 4: Write portable programs for distributed architectures using Message-Passing interface (MPI) library
CLO # 2: Code and analyze complex problems of shared memory systems with OpenMP.

Q3: Let's assume FBR wants that any digital and physical transaction in the country is only allowed if a person is an active taxpayer. That implies thousands or millions of requests might be coming in at the same time to FBR to ask if some person is an active taxpayer or not. FBR wants to answer each such query in no more than a few milliseconds. One way to answer such queries is to forward each query to a database, though doing so might be slow and will generate substantial load on the DB layer. FBR tasks you to solve this problem. You decide to instead keep required data in memory for quick response and use either OpenMP or MPI to quickly answer queries. Your task is to parallelize such that your program could answer ***yes*** or ***no*** provided the national id card number in active taxpayer list. Let's assume the number of active taxpayers is ***n*** (where n is in millions and increasing). FBR has active taxpayer list in a huge text file where each record is just the national ID card number of active taxpayer. Let's assume they refresh that list once a month, and your system will reply to trillions of queries before respawning to use the new list. Let's assume you have ***p*** processors available at your disposal for processing. You might take further assumptions that does not conflict with above information.

(a) **[3 marks]:** Will you use an ***OpenMP*** based solution or ***MPI*** based solution? Provide clear and specific reasons for your choice.

(b) **[10 marks]:** How will you solve this problem ***efficiently***? How you plan to solve the problem? Provide either a pseudo code in English or a flowchart to explain your idea. Be as short as possible and clearly mention steps.

(c) **[2 marks]:** What is the time complexity of your solution? Please tell complexity involving parameters ***n***, ***p*** and any other parameter you might need.

CLO # 4: Write portable programs for distributed architectures using Message-Passing interface (MPI) library

Q4 [1.5+1.5+1.5+1.5 marks]: Different MPI collective communication operators are applied on the processes in an MPI program that have data as given in Fig. 1 below. After the completion of a particular operation, each process gets data as shown in part (a), part (b), part (c) and part(d). Give the name of the collective communication operation that leads to the data being communicated for parts (a) through (d). You do not need to give the full function call or arguments.

P0	1	2	3	4	5
P1	6	7	8	9	10
P2	11	12	13	14	15
P3	16	17	18	19	20
P4	21	22	23	24	25

Fig. 1

P0	55	60	65	70	75
P1	55	60	65	70	75
P2	55	60	65	70	75
P3	55	60	65	70	75
P4	55	60	65	70	75

part (a)
All reduce

P0	1	6	11	16	21
P1	2	7	12	17	22
P2	3	8	13	18	23
P3	4	9	14	19	24
P4	5	10	15	20	25

part (b)
All to all

P0	1	2	3	4	5
P1	1	2	3	4	5
P2	1	2	3	4	5
P3	1	2	3	4	5
P4	1	2	3	4	5

part (c)
bcast

P0	1	2	3	4	5
P1	7	9	11	13	15
P2	18	21	24	27	30
P3	34	38	42	46	50
P4	55	60	65	70	75

part (d)
Scan

CLO # 4: Write portable programs for distributed architectures using Message-Passing interface (MPI) library

Q5 [3+5 marks]: Consider the following MPI code. Explain the working of the code and write its output.

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

int main(int argc, char** argv) {
  const int PING_PONG_LIMIT = 5;

  // Initialize the MPI environment
  MPI_Init(NULL, NULL);
  // Find out rank, size
  int world_rank;
  MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);
  int world_size;
  MPI_Comm_size(MPI_COMM_WORLD, &world_size);

  // We are assuming 2 processes for this task
  if (world_size != 2) {
    fprintf(stderr, "World size must be two for %s\n", argv[0]);
    MPI_Abort(MPI_COMM_WORLD, 1);
  }

  int ping_pong_count = 0;
  int partner_rank = (world_rank + 1) % 2;
  while (ping_pong_count < PING_PONG_LIMIT) {
    if (world_rank == ping_pong_count % 2) {
      ping_pong_count++;
      MPI_Send(&ping_pong_count, 1, MPI_INT, partner_rank, 0, MPI_COMM_WORLD);
      printf("%d sent and incremented ping_pong_count %d to %d\n",
           world_rank, ping_pong_count, partner_rank);
    } else {
      MPI_Recv(&ping_pong_count, 1, MPI_INT, partner_rank, 0, MPI_COMM_WORLD,
             MPI_STATUS_IGNORE);
      printf("%d received ping_pong_count %d from %d\n",
           world_rank, ping_pong_count, partner_rank);
    }
  }
  MPI_Finalize();
}
```

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

Q6. Parallelize following pieces of code using OpenMP pragmas. Be careful that some problems may require re-arranging the code to break the dependencies. Assume that there is no syntax error, and all the required libraries are pre-included. For all the subparts, explicitly mention private and shared variables, reduction, and scheduling type through proper clauses. Do not take things implicitly.

(a) **[4 marks]:** Consider the following code. Parallelize the code using OpenMP pragmas. Your parallel team must contain exactly four threads. There is no restriction on pragmas.

```
int sum=0, n=1000;
for (int i = 1; i <= n; ++i)
{
    sum += i;
}
printf ("sum is: %d\n", sum);
```

(b) [4 marks]: Parallelize following piece of code using OpenMP with a team of 8 threads.

```
int brr[1000];
int i;
brr[0]=5;
for (i = 1; i < 1000; ++i)
{
    brr[i]=brr[i-1] + 1;
}
printf ("Sum at last index=%d\n", brr[999]);
```

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

Q7 [6 marks]: Show the output of the following OpenMP program.

```
#include <iostream>
#include <omp.h>
using namespace std;
int main() {
   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 x = omp_get_thread_num();
      cout << "At thread: " << x << " iteration: ";
      cout << j << endl;
   }
   for (int i = 0; i < 10; i++) {
      cout << nums[i] << " ";
   }
   cout << endl;
   return 0;
}
```

0 0
0 1
0 2
1 3
1 4
1 5
2 6
2 7
2 8
0 9

3 8 15 24 35 48 63
80 99 120

0361

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

Q8.

(a) **[5 marks]:** Drive the cost of All-to-All Broadcast operation on Mesh topology consisting of nine nodes.

(b) **[6 marks]:** How do you compute prefix sum on the 8-node hypercube? Show the proper working.