CS416 Parallel and Distributed Computing

Monday, June 22, 2020

Course Instructor

Mr. Muhammad Husnain

Serial No:

Final Term Exam
Spring Semester 2020
Max Time: 3 Hour
Max Marks: 100
Exam Weight (Out of 100). 50

______________________ Roll No	________________ Section

General Guidelines:

1. Strictly follow the submission guidelines.
2. Questions reading and understanding are also part of the exam, only answer what is asked.
3. Read the questions carefully for clarity of context and understanding of meaning and make assumptions wherever required, for neither the invigilator will address your queries, nor the teacher/examiner will respond to any of your queries.
4. There are total ten questions in the paper. Q0 is an extra-requirement that belongs to the submission's guidelines.
5. While answering a question on answer-sheet, you need not to write question statement, just write question number instead.

Guidelines for Submission:

1. You must solve the exam on the blank sheets in your own handwriting, take visible snaps of the sheets, and then combine them into a single PDF/DOCX file in the right order**.**
    a. If your roll number is '16F-0123' then, you should name the PDF/DOCX file as '16F-0123.pdf' OR '16F-0123.docx'
    b. Following the right submission format carries: 1 Point

2. To avoid ordering issues, write your roll number and page number on each of the solution sheets. Following this convention carries: **1 Point**
3. Your snaps should be fairly visible, otherwise the paper shall not be graded.
4. You must submit your solution before due time via **Google Classroom**. Submissions submitted after the due time shall not be considered.
5. Also email me your solution at 'm.husnain@nu.edu.pk' within the due time.
6. If you do not finish every part of a question, do not worry! You can still submit what you have done to get marks based on your efforts.
7. In case of copied or plagiarized solutions in exam OR If a student provided help to another student during exam both will be awarded "F" grade and it will affect the student CGPA.
8. Viva of any student can be conducted by the instructor after conducting an online exam in case of any doubt.

	Q0	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Total
Marks Distribution	2	15	5	10	10	8	8	6	8	8	20	100

Attempt All the Questions

Question No. 1: [Write this question on blank white paper] **[15 Points] [25 mins]**

(a) Consider a hyper-cube of 8 processing nodes with one process on each of the nodes. Further consider that process P_3 has 8 integer values: [0, 1, 2, 3, 4, 5, 6, 7, (last four digits of your roll number)] to be scattered to P_0, P_1, P_2, P_3, P_4, P_5, P_6, P_7 respectively as per the definition of scatter operation. Perform and draw step-by-step Scatter operation on an 8-node hyper-cube inter-connection with P_3 as source of the scatter.

Question No. 2: [Write this question on blank white paper] **[3 + 2 Points] [7 mins]**

Assume that you have a sequential program that runs in [64 + last two digits of your roll number] seconds (e.g., if your roll number is 16F-1234 then 64+34= 98 sec). You write a parallel version which you run on different core configurations and get the following execution times:

Number of cores	1	2	4	8
Execution time in seconds	70	40	30	28

What is the speedup for each of the configuration? Discuss your observation about the relationship between number of cores and the speedup?

Question No. 3: [Also attach snap of your rough work] **[10 Points] [15 mins]**

Assume that *P* is total number of MPI processes generated and is equal to 5. Further assume that '*my_rank*' is the process's ID. Write output of the following piece of code assuming that MPI environment is initialized properly and there are no errors:

```
int a = my_rank; int b = 0;

if (my_rank % 2 == 1) {
    MPI_Send(&a, 1, MPI_INT, (my_rank + 1) % P, 1, MPI_COMM_WORLD);
    MPI_Recv(&b, 1, MPI_INT, (my_rank - 1 + P) % P, 1, MPI_COMM_WORLD, &status);
    printf("Process# %d, Received b= %d \n  ", my_rank, b);
}
else {
    MPI_Recv(&b, 1, MPI_INT, (my_rank - 1 + P) % P, 1, MPI_COMM_WORLD, &status);
    printf("Process# %d, Received b= %d \n  ", my_rank, b);
    MPI_Send(&a, 1, MPI_INT, (my_rank + 1) % P, 1, MPI_COMM_WORLD);
}
```

Question No. 4: [Also attach snap of your rough work] **[10 Points] [10 mins]**

Suppose a process P_0 has an integer array of 15 elements as:-

```
 int sendbuff[15] =[12,13,14,15,16,17,18,19,20,1,2,3,4,5,6];
```

Now, P_0 wants to scatter that in such a way that:
"P_0 shall keep first element in its own receiveBuffer, next 2 elements shall be given to P_1, P_2 will be provided by next 3 elements, P_3 shall receive next 2, P_4 will receive next 1 element, P_5 shall receive next 2 elements, and P_6 will receive the remaining 4 elements."

For MPI primitive scatter operation (vector variant) following is the routine header:-

```
int MPI_Scatterv(void *sendbuf, int *sendcounts, int *displs,
      MPI_Datatype senddatatype, void *recvbuf, int recvcount,
      MPI_Datatype recvdatatype, int source, MPI_Comm comm)
```

Your task is to calculate the '*sendcounts*' and the '*displs*' arrays for the scatter operation. You are not required to write the code, instead you only need to provide resultant '*sendcounts*' and '*displs*' arrays.

Question No. 5: [Also attach snap of your rough work] **[8 Points] [10 mins]**

Write **output and the output-reason** for the following piece of code. If last digit of your Roll number is an even number, then you will perform part (a), else you will go for part (b).

[Assume that there are no errors in the code]

(a) For even Roll numbers

```
int i;
omp_set_dynamic(0);
omp_set_num_threads(4);
int counter = 0;
printf("total threads are:%d \n", omp_get_num_threads());

#pragma omp parallel for schedule(static) firstprivate(counter) lastprivate(counter)
for (i = 1; i <= 40; i++) {
     counter = counter + i;
}
printf("counter after the join is:%d \n", counter);
```

(b) For odd Roll numbers

```
int k;
omp_set_dynamic(0);
omp_set_num_threads(4);
int counter = 0;
printf("total threads are:%d \n", omp_get_num_threads());

#pragma omp parallel
{
        #pragma omp master
        printf("total threads are:%d \n", omp_get_num_threads());

        #pragma omp for schedule(static) firstprivate(counter) lastprivate(counter)
        for (k = 1; k <= 40; k++) {
                        counter = counter + k;
        }
}

printf("counter after the join is:%d \n", counter);
```

Question No. 6: [Write this question on blank white paper] **[8 Points] [15 mins]**

Parallelize following piece of code using OpenMP with *static* scheduling type. Use proper chunk size to minimize the work-imbalance between the threads. Your parallel implementation must produce same results as the sequential does.
Also defend your implementation (e.g., why the used chunk-size reduces the load imbalance, why you parallelized outer-loop only, the inner-loop only, OR why parallelized the both?). Assume that you can only generate maximum four openMP threads.

```
int row, col;
int arr2d[40000][40000] = { 0 }; //initializes whole matrix to zeros
for(row = 0; row < 40000; row++) {
    for (col = row; col < 40000; col++) {
        arr2d[row][ col] = row + col;
    }
}
```

Question No. 7: [Write this question on blank white paper] **[6 Points] [10 mins]**

What is the CUDA program life cycle, write down different steps in the life cycle.

Question No. 8: [Write this question on blank white paper] **[8 Points] [8 mins]**

For each of the following loops, discuss whether it is in canonical form or not? Defend your answers with proper arguments.

(a)	**(b)**

`int i, sum = 0;`
`for (i = 1; i < 100; i = i+2) {`
`    if (i % 15 == 0) break;`
`    sum += i;`
`}`
	`int i, sum = 0, n = 10, m = 4;`
	`for (i = n * m; i >= 0; i = i - 3) {`
	`    if (i % 2 == 0) continue;`
	`    sum += i;`
	`}`

(c)	**(d)**

`int i, sum = 13, g = 450;`
`for (i = 13; i <= g; i++) {`
`    if (i % 2 == 0) return;`
`    sum += i;`
`}`
	`int i, sum = 10;`
	`for (i = 0; i < 400; i++) {`
	`    if (i % 4 == 0) continue;`
	`    sum += i;`
	`}`

Question No. 9: [Write this question on blank white paper] **[8 Points] [15 mins]**

Parallelize the following code blocks using openMP pragmas. Be sure to explicitly specify the "schedule" options that should be used, even if you want to use the default options. Assume variable N is very large. Explicitly list all the variables within the range of a parallel pragma that are private using the private() directive. **[Be careful about the dependencies.]**

Assume that *matrix1* and *matrix2* are 2d-arrays of size *N x N* and *C* is 1d-array of size *N*.

```
C[0] = 1;

for (j = 1; j < N; j++) {

    C[j] = C[j - 1];

    for (k = 0; k < N; k++) {

        C[j] += matrix1[j][k] + matrix2[j][k];

 }
```

Question No. 10: **[20 Points] [30 mins]**
[You can also submit typed version for this question, but remember not to plagiarize]

Write a hybrid parallel program using MPI and openMP that finds global average of distributed mega-arrays using *P* MPI-processes and 4 openMP threads per MPI-process.

1- At the start of the program each MPI-Process creates an integer array of 10 million elements.

2- Then each process uses four OpenMP-threads to initialize the array using the following formula: -

    Value at ith index = ((i * (my_rank + 1) % P) + my_rank)

   Here ***my_rank*** is the process number; ***P*** is total number of MPI-processes.

3- Now, each MPI-process (using the four threads) calculates the local averages on the locally created arrays and displays those averages as [Assuming that there are four mpi-processes]: -

```
process#0-->Average=1.5

process#2-->Average=3.5

process#3-->Average=3.0

process#1-->Average=2.0
```

4- Now, each mpi-process takes part in the reduction communication to help MPI-process with rank 0 to calculate and show the globally correct average.

   “From process#0--> Global average is: 2.5”

[Your code should work for any number of MPI-processes with 4-threads each. Write code from scratch and remember that writing proper program structure also carries points]

---Good Luck 😊---