INDIAN INSTITUTE OF TECHNOLOGY KHARAGPUR
Mid-Autumn Semester Examination 2025-26

Date of Examination: 19 Sep 2025 **Session: AN** **Duration: 2 hrs.**
Full Marks: 30 **Subject No.: CD61004**
Subject: High-performance Computing and Its Applications to Complex and Physical Systems
Department/Center/School: Centre for Computational and Data Sciences
Special Instructions:

Go inside the directory `CD61004-midsem-autumn202526` *on the computer before you start answering.*
Make sure that you create all your files inside this directory.

Section A

Answer ALL questions in this section.
For MCQ questions, more than one option may be correct. Write all the options that you believe to be correct for each question.

1. Which of the following statements about OpenMP are correct? (1 mark)

   (A) OpenMP supports intranode (shared memory) parallelism by creating threads which typically match the number of processor cores.

   (B) OpenMP requires explicit parallelism via programmer-inserted directives or subroutines.

   (C) OpenMP is exclusively a compiler without any language or pragma extension.

   (D) OpenMP can be used as a directive-based extension in both C and Fortran programming languages.

2. Consider the following serial code fragment (given in C and Fortran), performing matrix-vector multiplication: (2 marks)

   C:

```c
for (int i = 0; i < N; i++) {
    y[i] = 0.0;
    for (int j = 0; j < N; j++) {
        y[i] += A[i][j] * x[j];
    }
}
```

   Fortran:

```fortran
do i = 1, N
    y(i) = 0.0
    do j = 1, N
        y(i) = y(i) + A(i, j) * x(j)
    end do
end do
```

(A) Rewrite this code using OpenMP directives to parallelize it appropriately with maximum thread efficiency.

(B) Discuss which data should be shared and which should be private in this scenario.

3. Consider the following statements about parallel computing speedup and efficiency: (1 mark)

   (A) Efficiency is defined as speedup divided by the number of processors.

   (B) Amdahl's Law states that the maximum speedup of a program is limited by the fraction of code that cannot be parallelized.

   (C) Strong scaling refers to increasing the number of processors while decreasing the problem size.

   (D) Weak scaling keeps the problem size fixed while increasing the number of processors.

   Which of these statements are true?

   (A) Only A and B

   (B) Only A and D

   (C) Only B and C

   (D) All are true

4. Given a CPU with the following attributes: 8 cores, 3.0 GHz clock speed, and each core can perform 4 double-precision floating-point operations per clock cycle. Which of the following correctly describe the peak FLOPS? (1 mark)

   (A) Peak FLOPS = number of cores $\times$ clock speed $\times$ FLOPs per cycle

   (B) Peak FLOPS for single precision is higher than double precision on the same CPU.

   (C) Peak FLOPS is expressed in GFLOPs or TFLOPs depending on magnitude.

   Choose the correct combination:

   (A) A and B only

   (B) A and C only

   (C) B and C only

   (D) All are correct

5. A program consists of 90% parallelizable code and 10% sequential code. Using Amdahl's Law: (2 marks)

   (A) Calculate the theoretical maximum speedup using 8 processors.

   (B) If the program runs 50 seconds sequentially, estimate the parallel runtime on 8 processors assuming ideal conditions.

6. Explain the differences between strong scaling and weak scaling in parallel computing (1 mark)

7. In the Fortran program below, calculate the total memory used by the program. How much total memory (in bytes) is consumed by this program? Show your calculations. Assume that 1 KB = 1000 bytes for ease of calculation. (1 mark)

```
integer :: i, j, k
real :: A(1000), B(1000), C(200), x, y, z
real(kind=8) :: start_time,end_time
```

8. Which of the following are key considerations for efficient MPI programming? (Select all that apply) (1 mark)

   (A) Minimizing synchronizations

   (B) Reducing contentions

   (C) Maximizing the size of messages

   (D) Optimizing reduction operations

9. Define a race condition in the context of parallel programming. Provide an example of a race condition that could occur in an OpenMP program. How would you resolve this race condition using OpenMP constructs? (1 mark)

10. Why is it important to call `MPI_Finalize` in every process of an MPI program? (1 mark)

Section B

Answer ALL questions in this section. Serial Fortran/C program for Question 12. is included in the folder sectionB.

11. Write an OpenMP Fortran/C program to print output in sequential order, starting from master thread. You can write any simple program for demonstration where each thread prints a message, and thread 0 should print its message first, followed by thread 1, thread 2, and so on. (4 marks)

12. Parallelize the program, `simpsons_integration.f90/simpsons_integration.c`, using:

    A) MPI-based model

    B) OpenMP-based model

    to compute the integration of a function using Simpson's formula. (10 marks)

13. Write a MPI program in which two processes (say rank 0 and rank 1) repeatedly pass a message back and forth (for N number of times, take N>10000). You should write your program so that it operates correctly even when run on more than two processes, i. e. processes with rank greater than one should simply do nothing. For simplicity, use a message that is an array of integers. How the time taken varies with the size of the message (write your observation in the answer sheet). (4 marks)

A Additional Information: MPI Subroutines

Here is a list of some MPI subroutines:

- `MPI_Init(err)`
- `MPI_Finalize(err)`
- `MPI_Send(data, count, MPI_Datatype datatype, destination, tag, MPI_Comm communicator, err)`
- `MPI_Recv(data, count, MPI_Datatype datatype, source, tag, MPI_Comm communicator, MPI_Status* status, err)`

- MPI_Bcast(data, count, MPI_Datatype datatype, root, MPI_Comm communicator, err)
- MPI_Reduce(send_data, recv_data, count, MPI_Datatype datatype, MPI_Op op, root, MPI_Comm communicator, err)
- MPI_Comm_Rank(Comm, Rank, err)
- MPI_Comm_Size(Comm, Size, err)
- MPI_Barrier(Comm, err)
- MPI_Datatype: MPI_Integer, MPI_Real, MPI_Double_Precision, MPI_Complex, MPI_Logical, MPI_Character
- MPI_Op: MPI_Max, MPI_Min, MPI_Sum, MPI_Prod
- MPI constants: MPI_Any_Tag, MPI_Status_Ignore, MPI_Comm_World
- MPI_Wtime()
- !$OMP PARALLEL
- !$OMP DO
- omp_get_thread_num()
- omp_get_num_threads()
- !$OMP BARRIER
- !$OMP CRITICAL [(name)]
- !$OMP REDUCTION(operator:var)
- omp_get_wtime()
- !$OMP SINGLE
- !$OMP MASTER