UNIVERSITY OF RUHUNA

Faculty of Engineering

End-Semester 8 Examination in Engineering: July 2022

Module Number: EE8207 **Module Name: High Performance Computing**

[Three Hours]

[Answer **all questions**, each question carries 10 marks.]

Q1. a) Briefly explain the Deadlock-Free and Starvation-Free concepts in multi-threaded environment.

[2 marks]

b) Write a program using Peterson's algorithm which does the following.

i) The program should decrement a variable using two threads.
ii) The initial value of the variable is 1000 and should be decremented by two using each thread.
iii) In each execution (in loop), program should print variable value and thread id.
iv) The program terminates when the variable value is 0.

Note: State the programming language you have used to write the program.

[4 marks]

c) Explain the "Arbitrator solution" in "Dining philosopher" problem.

[2 marks]

d) Describe how the solution proposed by K. Mani Chandy and J. Misra to the Dining philosopher problem solves starvation.

[2 marks]

Q2. a) Describe the barrier synchronization method used in OpenMP programming model.

[2 marks]

b) Write a program using OpenMP to calculate the addition of two matrices A and B in parallel. Each matrix has M rows and N columns, where M = 5 and N is a positive integer value greater than 10.

[2 marks]

c) What is a loop work sharing construct in OpenMP?

[1 mark]

d) Describe what is a recursive mutex in POSIX threads.

[2 marks]

e) Write all possible outputs of the program shown in Figure Q2.e) written using POSIX threads.

[3 marks]

Q3. a) Write all possible outputs of the MPI program shown in Figure Q3.a), if the number of tasks is 2.

[2 marks]

b) Write a program using MPI which does the following. Assume there are even number of tasks running and each task having an even number as the rank (n) will send its rank to the task having the next higher rank (n + 1). Tasks having an odd number rank will print the values received from the other process.

[3 marks]

c) What is the difference between blocking communication vs non-blocking communication in MPI?

[2 marks]

d) Describe the use of MPI_REDUCE function in the context of MPI.

[1 mark]

e) Explain MPI_BCAST and MPI_SCATTER functions in the context of MPI. (Use diagrams if required)

[2 marks]

Q4. a) Briefly explain the architectural difference between CPU and GPU.

[1 mark]

b) Write short notes on three memory types used in CUDA platform.

[3 marks]

c) Briefly explain Barriers and importance of implementing a barrier in CUDA programming model.

[2 marks]

d) Write a program using CUDA to convert an array containing temperature readings in Fahrenheit to Celsius. Consider the size of the array as N. Use the processing power of GPU for the conversion. Store the converted temperature values in a new array and print it at the end of the program. The formula to convert a temperature reading from Fahrenheit to Celsius in shown below.

cels = (fahr - 32.0) * 5.0/9.0.

[4 marks]

Q5. a) Use the code in Figure Q5.a) to answer the following questions.

i) Write a possible output of the code component.

[2 marks]

ii) Rewrite the code with following changes. At the end of the process, the process having rank 0 should notify the process having rank 1 by sending a message. Once the notification is received by rank 1 process, it should print "completed" and exit the execution.

[3 marks]

b) Explain the advantage of using multiple low speed (clock cycles) processors parallelly to solve a CPU intensive task instead using a high speed (clock rate) single processor.

[1 mark]

c) Explain the difference between a thread and a process.

[2 marks]

d) Suppose you need to simulate a cyclone (weather condition). For this you require massive computational power. Which multiprocessing architecture would you select out of shared memory, distributed memory and hybrid memory? Justify your answer.

[2 marks]

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

void *print_message_function( void *ptr );

int main() {
    pthread_t thread1, thread2;
    int message1 = 123;
    int message2 = 456;
    pthread_attr_t attr;
    pthread_attr_init(&attr);
    pthread_create( &thread1, &attr, print_message_function, (void *)&message1);
    pthread_create( &thread2, &attr, print_message_function, (void *)&message2);

    pthread_join( thread1, NULL);
    pthread_join( thread2, NULL);

    printf("done");
    return 0;
}

void *print_message_function( void * ptr ) {
    int *message;
    int index = 0;
    message = (int* ) ptr;
    for( index = 0 ; index < 2; index++) {
        printf("%d \n", *message + index);
    }
}
```

Figure Q2.e): POSIX threads code

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

int main (int argc, char *argv[]) {
   int numtasks, rank, dest, source, rc, count, tag=1;
   char inmsg, outmsg='x';
   MPI_Status Stat; MPI_Init(&argc,&argv);

   MPI_Comm_size(MPI_COMM_WORLD, &numtasks);
   MPI_Comm_rank(MPI_COMM_WORLD, &rank);

   if (rank == 0) {
         dest = 1; source = 1; outmsg='y';
         rc = MPI_Recv(&inmsg, 1, MPI_CHAR, source, tag, MPI_COMM_WORLD, &Stat);
          rc = MPI_Send(&outmsg, 1, MPI_CHAR, dest, tag, MPI_COMM_WORLD);
          printf("The character %c sent to rank 1 \n",outmsg);
   } else if (rank == 1) {
         dest = 0; source = 0;
         rc = MPI_Send(&outmsg, 1, MPI_CHAR, dest, tag, MPI_COMM_WORLD);
         printf("The character %c sent to rank 0 \n", outmsg)

         rc = MPI_Recv(&inmsg, 1, MPI_CHAR, source, tag, MPI_COMM_WORLD, &Stat);
         printf("The character %c sent to rank 0. \n",outmsg);
   }
   MPI_Finalize();
}
```

Figure Q3.a): MPI program

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

```
int main(int argc, char *argv[]) {
  pthread_t thread1;
  int numtasks, rank, dest, source, rc, count, tag = 1;
  int inmsg;
  MPI_Status Stat;
  MPI_Init(&argc, &argv);
  MPI_Comm_size(MPI_COMM_WORLD, &numtasks);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  omp_set_num_threads(4);

  if (rank == 1) {
    int payload = 4;
    dest = 0;
    source = 0;
    rc = MPI_Send(&payload, 1, MPI_INT, dest, tag, MPI_COMM_WORLD);
    printf("message sent by process with rank %d\n", rank);
  }
  else if(rank == 0){
    dest = 1;
    source = 1;
    rc = MPI_Recv(&inmsg, 1, MPI_INT, source, tag, MPI_COMM_WORLD, &Stat);
    printf("Rank is %d, received number %d\n", rank, inmsg);
    inmsg++;

     #pragma omp parallel {
       #pragma omp for
       for(i = 0 ; i < inmsg ; i++) {
         int tId = omp_get_thread_num();
         printf("i = %d \n ",i);
       }
    }
  }
  MPI_Finalize();
}
```

Figure Q5.a): Sample code