100

RAJARATA UNIVERSITY OF SRI LANKA
FACULTY OF APPLIED SCIENCES

B.Sc. (Four-year) Degree in Information and Communication Technology

Fourth Year - Semester 1 Examination - Sept/Oct 2019

ICT 4305 — PARALLEL AND CLUSTER COMPUTING

Time: Two (2) hours

To be completed by the candidate

Index Number

Important Instructions

- Answer **all 5** questions.
- This paper consists of **5** questions on **11** pages.
- **Write your answers using the space provided** in this question paper.
- Do not tear off any part of this answer book. Under no circumstances may this book (or any part of this book), used or unused, be removed from the examination hall by a candidate.
- Questions appear on both sides of the paper. If a page is not printed, please inform the supervisor immediately.

To be completed by the examiners

1	
2	
3	
4	
5	

1. Write **short** answers.

[4 marks × 5 = 20 marks]

(a) Define the *speedup* of a parallel computation. Give **one** reason as to why an *ideal speedup* cannot usually be achieved in a parallel computation.

(b) Assuming the *prog* executable is available, write down the outcome of the following command:

```
mpirun -np 3 -machinefile machines prog
```

when executed on the *nelum* computer of an MPI cluster.

Note: Assume the *machines* file is:

```
nelum.as.rjt.ac.lk
manel.as.rjt.ac.lk
```

(c) Write down the difference between MPI_Isend and MPI_Send and explain how the programmer use MPI_Isend in a *beneficial* way.

(d) Define *false sharing* and briefly indicate how it can be eliminated.

(e) Assume that the following program is run on a *dual-core computer*. Comment on the outcome of the **section marked as A** and suggest one correction that can be made to make the parallelism *beneficial*.

```
#include <omp.h>

int main () {
  int i;
  float a[N], b[N], c[N], d[N];

  /* Some initializations */
  for (i=0; i < N; i++) {
    a[i] = 15.0;
    b[i] = 21.3;
  }

  /***** Section A begin ****/
  #pragma omp parallel shared(a,b,c,d) private(i)
  {
    for (i=0; i < N; i++)
      c[i] = a[i] + b[i];

    for (i=0; i < N; i++)
      d[i] = a[i] * b[i];
  }
  /***** Section A end ****/
}
```

2. Assume that you plan to implement a parallel algorithm to find the *maximum* of n numbers using a *divide-and-conquer* approach on a **four (4)** computer cluster.

Draw a diagram to illustrate your design and compute the **theoretical cost** of this approach. (See hint given below.)

Hint: When computing the communication time, ignore the fact that source and destination may not be directly linked. Also note that only one point-to-point communication can occur at any given time. That is, a single computer cannot communicate with two or more computers at the same time.

[20 marks]

3. An MPI-based code is needed for **two computers** to compute in parallel using **task parallelism**, the *maximum* and *minimum* of a set of numbers input from a file. The following is a part of that code:

```
#include "mpi.h"
#include <stdio.h>
#define MAXSIZE 2000

void main(int argc, char *argv) {
  int myid;
  int weather[MAXSIZE], i, max, min;
  char fn[255];
  char *fp;

  MPI_Init(&argc, &argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &myid);

  if (myid == 0) { /* get the input */
    strcpy(fn,/home/sarath/weatherdata.txt"));
    if ((fp = fopen(fn,"r")) == NULL) {
      printf("Can't open the input file: %s\n\n", fn);
      exit(1);
    }
    for (i =0; i<MAXSIZE; i++) fscanf(fp, "%d", &weather[i]);
  }

  /*** Section A begin ***/
  /* Please write on the next page, the code for this section. */
  /*** Section A end ***/

  if (myid == 0) printf("maximum=%d minimum=\n",max,min);

  MPI_Finalize();
}
```

Write on the next page the code statements that should be placed in Section A of the above.

Hints: The following correct code fragments have been taken from other codes written for different purposes. You may use them as aids to build your Section A.

- `MPI_Bcast(&data, MAXSIZE, MPI_INT, 0, MPI_COMM_WORLD);`
-
```
MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
if (myrank == 0)   {
  int x  = 2;
  MPI_Send(&x,1,MPI_INT,1,3, MPI_COMM_WORLD);
} else if (myrank = = 1)   {
  int x;
  MPI_Recv(&x,1,MPI_INT,0,3,MPI_COMM_WORLD,&status);
}
```
-
```
int i, lastpositive;
...
for (i=0; i<100; i++) {
  if (data[i] > 0)
     lastPositive = data[i];
}
```

[20 marks]

4. The following is a correct pthread code to find the sum of a set of numbers input from a file:

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

#define ARRAYSIZE 1000
#define THREADS 2

void *slave(void *myid);

int data[ARRAYSIZE]; /* Array of numbers to sum */
int sum = 0;
pthread_mutex_t mutex;
int wsize;

void *slave(void *myid) {
  int i,low,high,myresult=0;
  char fn[255], *fp;

  low = (int) myid * wsize;
  high = low + wsize;

  for (i=low;i<high;i++)
    myresult += data[i];

  pthread_mutex_lock(&mutex);
  sum += myresult;
  pthread_mutex_unlock(&mutex);
  return;
}

main() {
  int i;
  pthread_t tid[THREADS];
  pthread_mutex_init(&mutex,NULL);

  wsize = ARRAYSIZE/THREADS;

  strcpy(fn,/home/sarath/weatherdata.txt"));
  if ((fp = fopen(fn,"r")) == NULL) {
    printf("Can't open the input file: %s\n\n", fn);
    exit(1);
  }
  for (i =0; i<ARRAYSIZE; i++) fscanf(fp, "%d", &data[i]);

  for (i=0;i<THREADS;i++) /* create threads */
    if (pthread_create(&tid[i],NULL,slave,(void *)i) != 0)
      perror("Pthread_create fails");

  for (i=0;i<THREADS;i++) /* join threads */
    if (pthread_join(tid[i],NULL) != 0)
      perror("Pthread_join fails");

  printf("The sum from 1 to %i is %d\n",ARRAYSIZE,sum);
}
```

Change the above code to compute using *task parallelism*, the *maximum* and the *minimum* of the set of numbers. (**Note:** You need not write the entire code again. Describe only the changes that are required.)

[20 marks]

5. Multiplication between a matrix A and a vector x in which the number of columns in A equals the number of rows in x is defined as shown in the figure.

$$\mathrm{Ax} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n \\ \vdots \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n \end{bmatrix}$$

Figure 1: Matrix - vector multiplication

Using the following correct OpenMP code (written for some other purpose) as an aid, write down an OpenMP matrix-vector multiplication code ($ax = b$ where a is an $N \times N$ matrix and x and b are two $N \times 1$ vectors).

```
#include <omp.h>
#define CHUNKSIZE 100
#define N     1000

main ()   {
  int i, chunk;
  float a[N], b[N], c[N];

  /* Initializations */
  for (i=0; i < N; i++)
    a[i] = b[i] = i * 1.0;
  chunk = CHUNKSIZE;

  #pragma omp parallel shared(a,b,c,chunk) private(i)
  {
    #pragma omp for schedule(dynamic,chunk) nowait
    for (i=0; i < N; i++)
      c[i] = a[i] + b[i];
  } /* end of parallel section */
}
```

110

[20 marks]