RAJARATA UNIVERSITY OF SRI LANKA
FACULTY OF APPLIED SCIENCES

B.Sc. (Four-year) Degree in Information and Communication Technology
B.Sc. (Four-year) Degree in Applied Sciences
Fourth Year - Semester I Examination – June/July 2018

ICT 4305 – PARALLEL AND CLUSTER COMPUTING

**Time: Three (03) hours**

**To be completed by the candidate**

**Index Number**

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | |

## Important Instructions

- Answer **all 6** questions.
- This paper should have **6** questions on **15** 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 | |
| 6 | |
| Total | |

2

**1.** (i) If the fraction in a computation that cannot be divided into concurrent tasks is $f$, and if it is assumed that no overhead incurs when the computation is divided into concurrent parts, write down an equation for the *ideal speedup* when the computation is done with $n$ processors.

.....

.....

.....

.....

**[2 marks]**

(ii) Assuming that the *fork* function is successful and the *master* and *slave* functions are available, write down the expected outcome of the following C code, when it is executed on a *dual-core* computer.

```
#include <stdio.h>
main() {
  int x;
  x = fork();

  if (x==0)
    slave();
  else
    master();
}
```

.....

.....

.....

.....

.....

.....

.....

.....

**[3 marks]**

**(iii)** Out of *coarse-grain* and *fine-grain* parallelism, which is preferred? Justify your answer.

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

**[4 marks]**

**(iv)** The *sum* and the *average* of a large set of numbers is to be found. Using that example, outline *data* and *task parallelism*.

data parallelism: .....

.....

.....

.....

.....

.....

task parallelism: .....

.....

.....

.....

.....

.....

**[2 marks × 2 = 4 marks]**

**(v)** A parallel algorithm to find the *minimum* of $n$ numbers using a cluster of **four (4)** computers is needed. Assume that you are planning to implement the algorithm using a *general partitioning* approach where the master node distributes the portions of numbers to the three slave nodes, the slave nodes find the smallest of their portions and send them to the master node to find the *global minimum* of all minima.

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.

**[7 marks]**

**2.** **(i)** *Shared memory* and *message-passing* are the two dominant computing environments suitable for parallel computing. With suitable illustrations, briefly outline **two (2)** different computing systems for each of them.

**Shared memory:**

**i.** .....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

**ii.** .....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

**Message-passing:**

i. .....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

ii. .....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

**[3 marks ×4 = 12 marks]**

**(ii)** Describe the concept of *hyper-threading*.

.....

.....

.....

.....

.....

.....

**[3 marks]**

**(iii)** What is meant by *Uniform Memory Access (UMA)* in a shared-memory computer?

.....

.....

.....

.....

.....

.....

**[3 marks]**

**(iv)** What is meant by a *Beoufulf Cluster*?

.....

.....

.....

.....

.....

.....

**[2 marks]**

**3.** **(i)** Following is a machine file (named *machines*) available in an MPI implementation:

```
sun.ucsc.cmb.ac.lk
moon.ucsc.cmb.ac.lk
```

Assuming that the two computers (*sun* and *moon*) are on and the network is also up, and the executable named *pmv* exists in the current directory, explain the outcome when the following command is executed at a terminal prompt on the *sun* computer:

```
mpirun -np 2 -machinefile machines pmv
```

.....

.....

.....

.....

**[2 marks]**

**(ii)** Assuming that *Comp()* is an available function, write down the *expected outcome* and *the benefit* of the following code:

```
MPI_Isend(&result,1,MPI_INT,2,400,MPI_COMM_WORLD,&request);
Comp();
MPI_Wait(&request,&status);
```

Expected outcome: .....

.....

.....

.....

.....

Benefit: .....

.....

.....

.....

.....

.....

**[2 marks ×2 = 4 marks]**

**(iii)** In MPI, when a variable has to be broadcasted from one process to all others using the MPI_Bcast routine, should the same MPI_Bcast code line appear in the receiver process codes too? Justify your answer.

.....

.....

.....

.....

**[3 marks]**

**(iv)** i. What is the issue with the following MPI code fragment? (Assume that *tag1* and *tag2* are appropriately defined.)

```
if (rank == 0) {
    int x  = 2, y;
    MPI_Send(&x, 1, MPI_INT, 1, tag1, MPI_COMM_WORLD);
    MPI_Recv(&y, 1, MPI_INT, 1, tag2, MPI_COMM_WORLD, &status);
} else if (rank == 1) {
    int y=3;
    MPI_Send(&y, 1, MPI_INT, 0, tag2, MPI_COMM_WORLD);
    MPI_Recv(&x, 1, MPI_INT, 0, tag1, MPI_COMM_WORLD, &status);
}
```

.....

.....

.....

.....

.....

**[3 marks]**

ii. Write down a simple solution for the above using the **same** statements.

.....

.....

.....

.....

.....

**[3 marks]**

**4.** (i) Given below is a pthread code skeleton that can be used to add a matrix $a$ with matrix $b$ and to put the resultant matrix in $c$.

```
/* matrix_add.c -  Matrix addition using threads*/

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

#define N  2000      /* Matrix size */
#define THREADS  2   /* Number of Threads */

void *slave (void *AAA);

/*Shared data*/
float a[N][N];
float b[N][N];
float c[N][N];

void *slave( void *myid ) {
  long x, low, high;
  int i,j;

  /*Calculate work area */
  x = BBB;
  low = (long) myid * x;
  high = low + x;

  /*Calculation*/
  for (i=CCC; i<DDD; i++)
    for (j=0; j<N; j++)
      EEE = a[i][j] + b[i][j];
}

int main(int argc, char *argv[]) {
  long i,j;
  pthread_t tid[THREADS];

  for (i=0; i<N; i++)
    for (j=0; j<N; j++) {
      a[i][j] = 1+(int) (NUMLIMIT*rand()/(RAND_MAX+1.0));
      b[i][j] = (double) (rand() % RANDLIMIT);
    }

  for ( i=0; i<FFF ; i++)
    GGG( &tid[i], NULL, HHH, (void *) i);

  for ( i=0; i<FFF ; i++)
    III( tid[i], NULL);
}
```

*Choosing from the answer choices given,* write down in the answer boxes below, suitable fill-ins for the places labelled AAA to III in the above code in order to get a correct implementation.

| | | |
|---|---|---|
| AAA: | BBB: | CCC: |
| DDD: | EEE: | FFF: |
| GGG: | HHH: | III: |

**Answer choices:**

$$\{c[i][j], high, low, myid, THREADS\}$$

$$\{N/THREADS\}$$

$$\{pthread\_create, pthread\_join, slave\}$$

**[9 marks]**

**(ii)** Isn't *mutual exclusion* needed in the above code when the matrix c is being updated by the different threads? Explain.

.....

.....

.....

.....

.....

.....

.....

.....

.....

**[4 marks]**

**(iii)** Suggest a computer system that will be suitable to run the above code.

.....

.....

.....

**[2 marks]**

**5.** Explain what the following OpenMP code fragments are expected to do when they are run on a *dual-core computer*.

**Note:** Assume that the OMP_NUM_THREADS environment variable is **not** set.

**(i)**
```
N = 1000;
chunk = 100;

#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];
}
```

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

**[5 marks]**

**(ii)**
```
n=1000;
chunk=100;

#pragma omp parallel for      \
  default(shared) private(i)  \
  schedule(static,chunk)      \
  reduction(+:result)

  for (i=0; i < n; i++)
    result = result + (a[i] * b[i]);
```

.....

.....

.....

.....

.....

.....

.....

.....

**[5 marks]**

**(iii)**
```
#pragma omp parallel shared(a,b,c,d) private(i)
{
  #pragma omp sections nowait
  {
    #pragma omp section
    for (i=0; i < N; i++)
      c[i] = a[i] + b[i];

    #pragma omp section
    for (i=0; i < N; i++)
      d[i] = a[i] * b[i];
  }  /* end of sections */
}  /* end of parallel section */
```

.....

.....

.....

.....

.....

.....

.....

.....

**[5 marks]**

**6.** **(i)** The *collapse* of a set of integers is defined to be the sum of integers in the set. Similarly, the collapse of a single integer is defined to be the integer that is the sum of its digits. For example, the collapse of the integer 9299 is 29. This can be carried out recursively until a single digit results; the collapse of 29 is 11, and its collapse is the single digit 2. The *ultimate collapse* of a set of integers is just their collapse followed by the result being collapsed recursively until only a single digit $\{0, 1, ..., 9\}$ remains.

Assume that you have to find the *ultimate collapse* of a one dimensional array of $N$ integers.

Write down the **design** of a *message-passing* solution for the above problem on a cluster of computers.

(Note: Detailed MPI statement syntax is **not** needed.)