UNIVERSITY OF MORATUWA

FACULTY OF ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

B.Sc. Engineering
2009 Intake Semester 8 Examination

CS4532 CONCURRENT PROGRAMMING

Time allowed: 2 Hours February 2014

ADDITIONAL MATERIAL: *None*

INSTRUCTIONS TO CANDIDATES:

1. This paper consists of **5** questions in **6** pages.
2. Answer any **4** questions.
3. Start answering each of the main questions on a new page.
4. The maximum attainable mark for each question is given in brackets.
5. This examination accounts for 60% of the module assessment.
6. This is a closed book examination.

   NB: It is an offence to be in possession of unauthorised material during the examination.
7. Only calculators approved by the Faculty of Engineering are permitted.
8. Assume reasonable values for any data not given in or with the examination paper. Clearly state such assumptions made on the script.
9. In case of any doubt as to the interpretation of the wording of a question, make suitable assumptions and clearly state them on the script.
10. This paper should be answered only in English.

Question 1 (25 marks)

(i) Compare and contrast (i.e., identify the similarities and dissimilarities) Shared Memory and Distributed Memory systems. [4]

(ii) Amdahl's law is used to find the maximum expected improvement to an overall system when only part of the system is improved. In the context of concurrent programming, we can present it as follows:

$$\frac{1}{1-p+\frac{p}{n}}$$

Where p is the parallel fraction and n is the number of processors.

Suppose a computer program has a method M that cannot be parallelized. M accounts for 40% of the program's execution time. The remaining code is parallelized.

a) How much speedup can we gain, if we implement the above program on a quad-core CPU? [2]

b) Is it really worth investing a quad-core CPU to solve this problem? [3]

(iii) The Fibonacci sequence can be calculated using the following equation:

$$F_i = F_{i-1} + F_{i-2} \quad \text{for } i \geq 2$$

Where $F_0 = 0$ and $F_1 = 1$.

a) Write a recursive serial algorithm (using pseudo code) to find out the Fibonacci number F_n. [4]

b) Write a parallel version of your algorithm (using pseudo code) to find out the Fibonacci number F_n. [7]

c) Will you gain more speedup when n is large? Briefly discuss. [3]

d) Is your algorithm a good candidate to be implemented on a GPU (using CUDA)? Briefly discuss. [2]

Question 2 (25 marks)

(i) Compare and contrast (i.e., identify the similarities and dissimilarities) semaphores and monitors. [4]

(ii) Consider the following programs.

```
Thread 1
      while(true){
            print "Red" + math.rand(10);
      }

Thread 2
      while(true){
            print "Blue" + math.rand(10);
      }
```

Math.rand(10) generates a random value between 1 and 10.

Change the above program to make sure the sum of all Red values it had printed so far is always less than half of the sum of all Blue values it has printed. For example, if Blue had printed 1, 5, and 7 then it is OK for Red to print 2 and 3 because 2+3 < (1+5+7)/2. [15]

(iii) a) Can the value of a semaphore be negative? Discuss. [2]

b) Can we read the value of a semaphore? Explain. [2]

c) Is there any advantage/disadvantage in reading the value of a semaphore? Discuss. [2]

Question 3 (25 marks)

(i) Give an example of a problem that can be solved using many threads where you can make use of a barrier. [4]

(ii) Consider the following program.

```
Public class Barrier{
      Public acquire(){
            //TODO
      }
}

Barrier barrier = new Barrier()
```

Consider the *i*-th thread.

```
Thread I
      Print "Start" + i
      barrier.acquire()
      Print "End" + i
```

Write a barrier implementation such that all threads will print "Start" + i before any thread will print "End" + i. [8]

(iii) What are the four conditions required for deadlock to occur? Explain all four using a suitable example. [6]

(iv) Following is an MPI style message passing system. *mq* refers to a message queue.

```
// Thread 1
mq1.receive();
mq2.send(x);

// Thread 2
mq2.receive();
mq1.send(y);
```

a) Will this code lead to a deadlock? Explain. [2]

b) Does this example violate the four conditions required for a deadlock to occur? Explain. [5]

Question 4 (25 marks)

(i) Outline a suitable solution to each of the following problems. Explain how it will address the given problem and satisfy safety and liveness properties (formal proofs are not required).

(a) A distributed set of nodes is used to process various messages received by a system (e.g., e-mails, Twitter messages, etc.). Each of those nodes keeps track of the number of messages it receives. Propose a solution where one of those nodes can periodically display the total number of messages received by the entire system. [5]

(b) The Panama Canal is a ship canal that connects the Atlantic Ocean to the Pacific Ocean. At one point, the Canal is so narrow that only one large ship can go across the Canal. However, two small ships can go across the Canal at the same time. Propose a solution to make sure ships can go through the Canal without getting stuck. [5]

(ii) While locks, semaphores, mutexes, and monitors satisfy safety and liveness properties, they are inefficient as they serialize the access to the critical region. Describe suitable solutions to the following problems that not only satisfy safety and liveness properties, but also improves the performance (e.g., by reducing waiting time or by increasing throughput).

a) Concurrent access to a linked list by multiple threads. [3]

b) Concurrent access to a message queue by multiple threads. [3]

(iii) The harmonic mean is one of the several kinds of averages. It is appropriate for situations when the average of rates is desired. Harmonic mean H of n positive real numbers x_1, x_2, x_3, ... $x_n > 0$ can be calculated as follows:

$$H = \frac{n}{\frac{1}{x_1} + \frac{1}{x_1} + \ldots + \frac{1}{x_1}} = \frac{n}{\sum_{i=1}^{n} \frac{1}{x_i}}$$

Outline an MPI program (using pseudo code) that can be used to calculate the Harmonic mean H of one million positive real numbers. Once the calculation is complete, all process involved in the computation need to know the value. Use relevant MPI functions that are given in the Appendix. Note that it is impractical to create one million concurrent processes/threads. [9]

Question 5 (25 maks)

(i) Compare and contrast (i.e., identify the similarities and dissimilarities) programming styles involved in PThreads, CUDA (for GPUs), and MPI. [6]

(ii) You are given a matrix $\mathbf{A}_{m \times n}$ and a vector $\mathbf{V}_{n \times 1}$. Suppose both m and n are expected to be in the thousands (i.e., $m, n > 1000$). Outline a CUDA program to calculate the matrix-vector multiplication $\mathbf{AV}$. Your solution should include the code for the Kernel function and the code required to invocate the Kernel function. [10]

(iii) Static or dynamic load balancing is essential in most systems to increase the utilization and quality of service. What type of load balancing would you recommend for the following problems? Justify your recommendation.

(a) Processing Twitter messages to find hashtags (hashtag is used to mark keywords or topics). [3]

(b) Matrix-vector multiplication. [3]

(c) Indexing web pages found by a web crawler. [3]

Appendix – MPI Functions

```
. . .
#include <mpi.h>
. . .
int main(int argc, char* argv[]) {
   . . .
   /* No MPI calls before this */
   MPI_Init(&argc, &argv);
   . . .
   MPI_Finalize();
   /* No MPI calls after this */
   . . .
   return 0;
}
```

```
int MPI_Init(int *argc, char **argv)
int MPI_Comm_size(MPI_Comm comm, int *size)
int MPI_Comm_rank(MPI_Comm comm, int *rank)
int MPI_Finalize()
int MPI_Send (void *buf,int count, MPI_Datatype datatype, int dest, int
            tag, MPI_Comm comm)
int MPI_Recv (void *buf,int count, MPI_Datatype datatype, int source, int
            tag, MPI_Comm comm, MPI_Status *status)
int MPI_Reduce(void *sendbuf, void *recvbuf, int count, MPI_Datatype
            datatype, MPI_Op op, int root, MPI_Comm comm)
int MPI_Allgather(void *sendbuf, int sendcount, MPI_Datatype sendtype, void
            *recvbuf, int recvcount, MPI_Datatype recvtype, MPI_Comm comm)
int MPI_Allreduce (void *sendbuf, void *recvbuf, int count, MPI_Datatype
            datatype, MPI_Op op, MPI_Comm comm)
int MPI_Bcast( void *buffer, int count, MPI_Datatype datatype, int root,
            MPI_Comm comm)
int MPI_Gather(void *sendbuf, int sendcnt, MPI_Datatype sendtype, void
            *recvbuf, int recvcnt, MPI_Datatype recvtype, int root,
            MPI_Comm comm)
int MPI_Scatter(void *sendbuf, int sendcnt, MPI_Datatype sendtype, void
            *recvbuf, int recvcnt, MPI_Datatype recvtype, int root,
            MPI_Comm comm)
```

Operation Value	Meaning
MPI_MAX	Maximum
MPI_MIN	Minimum
MPI_SUM	Sum
MPI_PROD	Product
MPI_LAND	Logical and
MPI_BAND	Bitwise and
MPI_LOR	Logical or
MPI_BOR	Bitwise or
MPI_LXOR	Logical exclusive or
MPI_BXOR	Bitwise exclusive or
MPI_MAXLOC	Maximum and location of maximum
MPI_MINLOC	Minimum and location of minimum

---------- END OF THE PAPER ----------