UNIVERSITY OF MORATUWA

FACULTY OF ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

B.Sc. Engineering
2010 Intake Semester 7 Examination

CS4532 CONCURRENT PROGRAMMING

Time allowed: 2 Hours September 2014

ADDITIONAL MATERIAL: *None*

INSTRUCTIONS TO CANDIDATES:

1. This paper consists of **5** questions in **7** 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) Explain three factors that contributed to the migration from uniprocessor systems to shared memory multiprocessor/multi-core systems? [6]

(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 20% 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 an 8-core CPU? [2]

b) Is it really worth investing an 8-core CPU to solve this problem? Briefly explain. [3]

(iii) Suppose you belongs to a team of developers developing a new programming language that supports threads. The new language also provides a *print*() function with the following function prototype:

```
void print(String s);
```

This function is expected to send the given string *s* to the terminal/console via Standard Output Stream (*stdout*) buffer. In a typical system, *stdout* buffer is shared among all threads and processes in the system. Moreover, bulk atomic memory copying is not supported in typical systems.

a) Provide four possible outcomes of the following program written using the new language. Clearly state any assumptions.

```
Thread one{
      print("Blue");
}

Thread two{
      print("Red");
}
```

[4]

b) Provide a suitable pseudo code for the implementation of *print*(). Make sure there are no race conditions. [6]

c) Given the possibility that many threads may simultaneously call *print*(), discuss about the efficiency of your implementation in (b). [4]

Question 2 (25 marks)

(i) Compare and contrast (i.e., identify the similarities and dissimilarities) locks, semaphores, and monitors. [6]

(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 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 9 because 2 + 9 < 1 + 5 + 7. Your implementation should be efficient. [14]

(iii) Following implementation of Account class is to be used to keep track of customers' accounts in a bank. Discuss whether this implementation is free from deadlocks.

```
class Account {
  double balance;  //account balance
  int id;          //accounts no

  void withdraw(double amount){    // withdraw money
     balance -= amount;
  }

  void deposit(double amount){     //deposit money
     balance += amount;
  }

  //Transfer money between accounts
  void transfer(Account from, Account to, double amount){
     lock(from);
     lock(to);
     from.withdraw(amount);
     to.deposit(amount);
     release(to);
     release(from);
  }
}
```

[5]

Question 3 (25 marks)

(i) “A semaphore is a counter capable of providing mutual exclusion and synchronization”. Briefly explain the meaning of this statement. [4]

(ii) Consider the following program with 2 threads.

```
Thread 1
      while(true){
            print "Red";
      }

Thread 2
      while(true){
            print "Blue";
            print "Blue";

      }
```

a) Provide four possible outcomes of the above program. [2]

b) Rewrite the above program using a semaphore(s) and **only one print statement** per thread such that we get the following sequence of outputs.

Blue, Blue, Red, Blue, Blue, Red, Blue, …. [9]

(iii) The geometric mean is one of the several kinds of averages. It is often used when comparing different items where each item has multiple properties, e.g., while comparing the performance of two database servers. Geometric mean of *n* real numbers x_1, x_2, x_3, ... x_n can be calculated as follows:

$$\left(\prod_{i=1}^{n} x_i\right)^{1/n} = \sqrt[n]{x_1 x_2 x_3 \ldots x_n}$$

Outline an MPI program (using pseudo code) that can be used to calculate the geometric mean of one million 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. [10]

Question 4 (25 marks)

(i) Using a suitable diagram illustrate the process of finding concurrency in a given problem. Briefly explain each step. [6]

(ii) Explain how a Dependency Graph helps to evaluate the design of a concurrent solution. [5]

(iii) Consider the following SQL-like query.

```
Query: MODEL = "Honda Civic" AND YEAR = 2001 AND
        (COLOR = "Green" OR COLOR = "White")
```

Evaluate the Dependency Graph of the above query using criteria mentioned in Question (ii). Your evaluation should consider the degree of concurrency and critical path. [8]

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

(a) Matrix-Matrix multiplication. [3]

(b) While cracking 10,000 password-protected word documents found from a suspected terrorist's laptop. Assume brute-force approach is used to crack passwords. [3]

Question 5 (25 marks)

(i) There are several techniques to convert a colour image to grayscale. The *lightness* technique averages the most prominent and least prominent colours using the following equation:

$$\frac{\max(R, G, B) + \min(R, G, B)}{2}$$

where *R*, *G*, *B* refers to three fundamental colours of a pixel. A grayscale image is obtained by applying this equation to each pixel separately.

Outline a CUDA program to convert a given colour image to grayscale using the *lightness* technique. Your solution should include the code for the Kernel function and the code required to invocate the Kernel function.

Hint: a typical CUDA supported GPU can only handle 1,024 threads per block. [13]

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

(a) SETI is one of the largest volunteer computing platforms that remotely executes jobs using idle computing resources. These jobs include analysing images/data from optical and radio telescopes for the presence of extra-terrestrial life. Over 200,000 SETI volunteer nodes are active at any given time. Each node contacts the SETI server and asks for a new job based on its computing capabilities. The same job is submitted only to two nodes to increase the reliability while maintaining better resource utilization (volunteers nodes may fail at any time). Once the job is completed, node submits the answer and asks for another job. You are required to design a concurrent job dispatching solution that allocates only two copies of the same job to volunteer nodes. [6]

(b) Clustering is a fundamental approach to manage Mobile Ad-hoc Networks (MANETs). In clustered networks, nodes are classified as cluster members and cluster heads. A cluster member is an ordinary node which sends its request to its cluster head. A cluster head is responsible for managing the cluster, handling intra-cluster requests, and participating in inter-cluster operations. While all nodes may be willing to become a cluster head, only one of the nodes in a given neighbour should be selected as a cluster head. You are required to design a cluster head selection solution for a given neighbourhood. [6]

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 -------------------------