UNIVERSITY OF MORATUWA

FACULTY OF ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

B.Sc. Engineering
2010 Intake Semester 8 Examination

CS4532 CONCURRENT PROGRAMMING

Time allowed: 2 Hours March 2015

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) Suppose a weather forecasting program typically takes **18 hours** to produce tomorrow's weather forecast. Therefore, the Meteorology department is forced to run it sufficient time ahead and only once a day. As more severe weather events are reported recently, meteorology department is interested in producing weather forecasts more frequently and with less turnaround time. They are thinking of achieving this by benefiting from more advanced hardware. Before deciding to upgrade their computing facility, the meteorology department wants your feedback on the following concerns they have.

a) "Should we go for an *n*-core processor while parallelizing our weather forecasting program or should we find a faster processor without modifying the program?"

Suggest your recommendation while considering the advantages and disadvantages of each approach. State any assumptions. [3]

b) "We want to provide a new weather forecast every **6 hours**. In case we decide to go for an *n*-core design with a modified program, how many cores are needed?"

It was also noted that a fraction ***f*** of the weather forecasting program cannot be parallelized. ***f*** accounts for **25%** of the program's execution time. The remaining code ***p*** is parallelized.

Hint: Amdahl's law in the context of concurrent programming can be given as:

$$\frac{1}{1-p+\frac{p}{n}}$$ [3]

c) "Should we upgrade our server to *n*-cores or should we replace the server with a cluster of nodes having a total of *n*-cores?"

Suggest your recommendation while considering the advantages and disadvantages of each approach. State any assumptions. [4]

(ii) Consider the following program with 3 threads.

```
Lock l1, l2, l3;

Thread 1                Thread 2                Thread 3
    while(1)                while(1)                while(1)
        l1.lock()               l2.lock()               l3.lock()
        l2.lock()               l3.lock()               l1.lock()
        print "Red"             print "Green"           print "Blue"
        l3.unlock()             l1.unlock()             l2.unlock()
        l1.unlock()             l2.unlock()             l3.unlock()
```

a) Provide 3 possible outcomes of the above program. [3]

b) Will this code lead to a deadlock? Explain using a Deadlock Modelling graph. [4]

c) Rewrite the above program using a semaphore(s) such that we get the sequence *Red, Green, Blue, Red, Green, Blue, ….* [8]

Question 2 (25 marks)

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

(ii) Consider the following program with 3 threads.

```
Thread 1
      while(1){
            print "Red" + math.rand(5);
      }

Thread 2
      while(1){
            print "Green" + math.rand(5);
      }

Thread 3
      while(1){
            print "Blue" + math.rand(5);
      }
```

Math.rand(5) generates a random value between 1 and 5.

Change the above program to make sure the sum of all the *Red* values (Sum_{Red}) it had printed so far is always greater than the sum of all the *Green* values (Sum_{Green}) it has printed. Similarly, Sum_{Green} must be always greater than the sum of all *Blue* values (Sum_{Blue}).

$$Sum_{Red} > Sum_{Green} > Sum_{Blue}$$

For example, if *Red* had printed 3 and 5 then it is OK for *Green* to print 4 and 3. Similarly, *Blue* may print 2, 1, and 2 because 3 + 5 > 4 + 3 > 2 + 1 + 2. [16]

(iii) Can you implement the solution for part (ii) using a conditional variable(s)? Briefly discuss your answer. [3]

Question 3 (25 marks)

(i) What are the advantages and disadvantages of using GPUs for solving embarrassingly parallel programs? [4]

(ii) *n*-body interaction is one of the most common simulations run using GPUs. It can be used to simulate movement of objects such as planets during the Big Bang. For example, given the masses and locations of planets following equation can be used to calculate the forces that a planet *i* experience due to another planet *j*.

$$f_{i,j} = \frac{Gm_i m_j}{d_{i,j}^2}$$

Where G is the gravitational constant, m_i and m_j are masses of the 2 planets, and $d_{i,j}$ is the distance between the 2 planets. These pairwise forces can be represented as a matrix F.

$$F = \begin{bmatrix} 0 & f_{0,1} & \cdots & f_{0,n-1} & f_{0,n} \\ f_{1,0} & 0 & \cdots & f_{1,n-1} & f_{1,n} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ f_{n-1,0} & f_{n-1,1} & \cdots & 0 & f_{n-1,n} \\ f_{n,0} & f_{n,1} & \cdots & f_{n,n-1} & 0 \end{bmatrix}$$

Where n is the number of planets. Once the force matrix is calculated, it can be used to calculate the acceleration of each planet using Newton's second law (i.e., $F = ma$). Which intern can be used to calculate the velocities and new locations of planets after a given time t. Then this process can be repeated again and again to calculate the location of planets at time t, $2t$, $3t$, and so on.

a) Outline a CUDA kernel to calculate the force matrix F. Your solution should also include the code required to invocate the Kernel function. Assume n = 1,000,000.

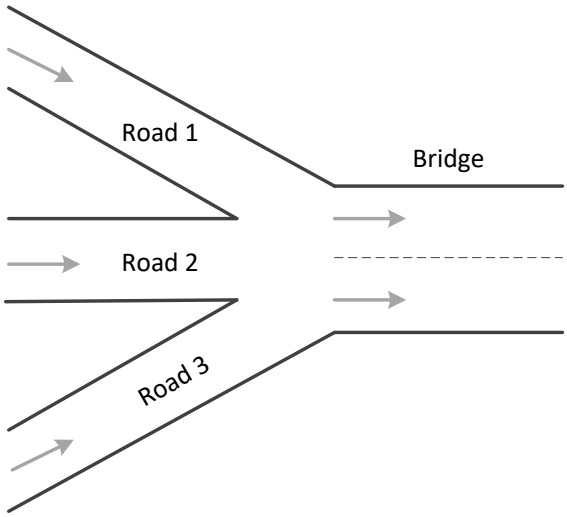
Hint: a typical CUDA supported GPU can only handle 1,024 threads per block. [15]

b) During the *n*-body simulation, one round of computation of matrix F needs to finish before initiating another round of computations (as the location of planets will change with time). How can we ensure that different rounds of calculating F will not overlap with each other? [3]

c) Is it worthwhile to calculate both $f_{i,j}$ and $f_{j,i}$ in a GPU? Briefly discuss. [3]

Question 4 (25 marks)

(i) Consider the following 3 single-lane, one-way roads joining just before a bridge which is one way but has 2 lanes. To reduce congestion on the bridge, vehicles from only 2 access roads are allowed at a time.



a) In this problem, what is the shared resource and who are the accesses trying to access that resource? [2]

b) Propose a solution to this problem using a semaphore(s) while assuming this is a shared memory problem. Pseudo code is sufficient. [8]

c) How would you solve the same problem, if it is interpreted as a distributed mutual exclusion problem, where you do not have access to a shared memory? [5]

(ii) The following table shows the current allocation of resources for 3 processes (*P*, *Q*, and *R*) and their maximum resource requirements. Is the current state is safe or unsafe? Show the steps. [4]

	Has	**Max**
P	2	9
Q	1	4
R	2	5

Free: 3

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

a) Indexing web pages found by web crawlers. [3]

b) Calculating the area under a given curve (i.e., integration) by breaking it into a large set of trapezoids. [3]

Question 5 (25 marks)

(i) Recommend a suitable solution pattern to parallelize the following code snippets. Provide a suitable justification for each case. State any assumptions.

a)
```
for(k = 1, k < 500; k++){
    x[k] = y[k – 1] + 1;
}
```
[3]

b)
```
for(k = 1, k < 500; k++){
    x[k] = y[k – 1] + 1;
    y[k] = z[k – 1] + 2;
}
```
[3]

c)
```
x = readData("xIn.txt");
y = readData("yIn.txt");
for(k = 1, k < x.getSize()(; k++){
    x[k] = x[k] + y[k - 1];
}
writeData("xOut.txt", x);
```
[4]

(ii) The Contraharmonic mean is one of the several kinds of averages. It is often used in image processing and Bioinformatics. Contraharmonic mean of n real numbers x_1, x_2, x_3, ... x_n can be calculated as follows:

$$C(x_1, x_2, x_3, \ldots, x_n) = \frac{x_1^2 + x_2^2 + x_3^2 + \ldots + x_n^2}{x_1 + x_2 + x_3 + \ldots + x_n}$$

Outline an MPI program (using pseudo code) that can be used to calculate the Contraharmonic mean of one million real numbers. Once the calculation is complete, mean should be stored on a variable at process 0. Use relevant MPI functions that are given in the Appendix. Note that it is impractical to create one million concurrent processes/threads. [15]

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 ----------