UNIVERSITY OF MORATUWA

FACULTY OF ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

B.Sc. Engineering
2012 Intake Semester 8 Examination

CS4532 CONCURRENT PROGRAMMING

Time allowed: 2 Hours March 2017

ADDITIONAL MATERIAL: *None*

INSTRUCTIONS TO CANDIDATES:

1. This paper consists of **five (5)** questions in **six (6)** pages.
2. Answer any **four (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 50% of the module assessment.
6. This is a closed book examination.

   NB: It is an offence to be in possession of unauthorized 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) State whether the following statements are TRUE or FALSE. Give **one sentence justification** for your answer.

Write the corresponding sub-question number and the answer in your answer book.

a) Race condition is a situation in which two or more processes continuously change their states in response to changes in the other process(es) without doing any useful work. [2]

b) The maximum speedup obtainable from a parallel program is proportional to *Pk* where *P* is the number of processors used and *k* is the percent time needed to run instructions that cannot be perfectly parallelized. [2]

c) Under the ideal conditions, a *k*-stage pipeline can obtain a speedup of at most *k*. [2]

d) Condition variable is a data type that is used to block a process or thread until a particular condition is true. [2]

e) Monitor is a programming language construct that provides equivalent functionality to that of semaphores and is easier to control. [2]

f) Deadlock can never occur, if no process is allowed to hold a resource while requesting another resource. [2]

g) Processes participating in an *MPI_Barrier*() call will deadlock, if one of the processes in the communicator never makes the call. [2]

h) MIMD-type data parallelism is key to benefit from GPUs. [2]

i) In agglomeration phase, tasks are combined into larger tasks while considering computational infrastructure. [2]

j) Cache coherence problem occurs when there are multiple distributed caches on a shared memory architecture. [2]

(ii) Disabling interrupts is one proposal for achieving mutual exclusions. Describe how disabling interrupts works and why is it unwise to use it? [5]

Question 2 (25 marks)

Let us assume that there are two types of threads, oxygen and hydrogen. To assemble these threads into water molecules (H_2O), we have to create a barrier that makes each thread wait until a complete molecule is ready to proceed. As each thread passes the barrier, it should invoke bond. We must guarantee that all the threads from one molecule invoke bond before any of the threads from the next molecule do. In other words:

- If an oxygen thread arrives at the barrier when no hydrogen threads are present, it has to wait for two hydrogen threads.
- If a hydrogen thread arrives at the barrier when no other threads are present, it has to wait for an oxygen thread and another hydrogen thread.

When an oxygen and two hydrogen molecules are ready, they produce a waster molecule. Then the barrier is released and the process starts again to produce another water molecule.

(i) Write the synchronization code for oxygen and hydrogen molecules that enforces these constraints.

Hint: you may use the following variables:

mutex = *Semaphore*(1), *oxygen* = 0, *hydrogen* = 0, *barrier* = *Barrier*(3), *oxyQueue* = *Semaphore*(0), *hydroQueue* = *Semaphore*(0)

oxygen and *hydrogen* are counters, protected by *mutex*. *barrier* is where each set of three threads meets after invoking bond and before allowing the next set of threads to proceed. *oxyQueue* is the semaphore oxygen threads wait on and *hydroQueue* is the semaphore hydrogen threads wait on. [15]

(ii) Comment on the potential speed up of your code. [3]

(iii) Suggest a suitable concurrent programming pattern to speed up the water molecule generation. [4]

(iv) Propose a suitable technique to balance the workload of oxygen and hydrogen threads. [3]

Question 3 (25 marks)

iPizza is an online only, pizza delivery service. Customers can order their favorite pizza combination(s) at iPizza.com website. Then iPizza chefs make the pizza based on the customer preferences. Once a pizza is made, it is put into the oven for baking, where the baking time depends on the type of pizza. Once the pizza is ready, it is put into a box and handed over to a member of the iPizza delivery team. Given the customer's address, the team member is expected to deliver the pizza as soon as possible. iPizza guarantees that a pizza order will be delivered to customer's doorsteps within 45 minutes of placing the order. Else, the pizza is free.

During peak hours, iPizza typically gets order for 120 pizzas per hour. Following latencies are known to occur during the pizza making and delivery process:

Step	Min Time	Average Time	Max Time
Making pizza	3 min	5 min	10 min
Baking pizza	2 min	4 min	7 min
Delivering pizza	1 min	12 min	20 min

Answer following questions based on this description. Clearly state any assumptions.

(i) Suggest a suitable concurrent programming pattern to speed up the operations at iPizza. [3]

(ii) How many chefs and delivery team members are required to handle the demand during peak hours? Show key calculation steps. [4]

(iii) What is the worst-case latency to deliver a pizza from the time of order placement?

Keep 3 mins to handle various overheads such as starting a new order, time to put pizza into oven, and time to put pizza into box. [2]

(iv) Discuss what changes are required to iPizza system to handle burst (i.e., bunch) arrival of orders. For example, a single customer may order multiple pizza. [4]

(v) What type of load balancing would you recommend for iPizza? Justify. [3]

(vii) Outline a pseudocode to synchronize the three steps of making, baking, and delivering pizza.

Hints: Consider the existence of multiple chefs and delivery team members. You may assume oven and boxing area have unlimited space to accommodate pizza. [9]

Question 4 (25 marks)

The following explains the process of making Ready-Mix Concrete (RMC) and loading them into trucks (i.e., trucks with the rotating cylinder).

First, the RMC plant mixes a batch of concrete based on customer orders. The volume of a typical batch varies between one to 10 truckloads. Time to make a batch is linearly proportional to batch size. Once the batch is ready, it is loaded to trucks, one at a time. When a truck comes to the plant it may proceed to load, if another truck is not loading and batch of concrete is ready. Else, it has to wait. Once loaded, a truck goes to the construction site, dumps the concrete, and then returns to the plant for another load. A truck's return time may vary depending on the location of delivery, delays at construction site, and traffic it may experience along the route. A concrete load that cannot be delivered to the construction site within 90 minutes of mixing has to be thrown away. Then the process restarts.

Answer the following questions assume that the orders of varying sizes come within the 12 hours that the plant is in operation per day.

(i) Outline a pseudocode for the operation of both the RMC plant and trucks using semaphores. [13]

(ii) Explain why this is a distributed mutual exclusion problem. [3]

(iii) How would you map/modify your answer for question (i) above to solve the distributed multiple exclusion problem? Discuss. [4]

(iv) Suppose truck drivers are paid for the number of concrete loads they deliver per day and the distance of delivery. Therefore, your solution need to be fair.

How would your solution enforce FIFO property? [5]

Question 5 (25 marks)

Suppose we want to generate the first 10,000 values of a geometric sequence. Given a number *a* and *d*, the rule is $x_n = 1/(a + nd)$. In General we can write the sequence as:

```
1/a, 1/(a + d), 1/(a + 2d), 1/(a + 3d), …
```

(i) Outline an MPI program (using pseudo code) that can be used to generate the first 10,000 values of the sequence. Once the list is generated, it should be send to all the nodes.

Use relevant MPI functions that are given in the Appendix. Note that it is impractical to create 10,000 concurrent processes/threads. [13]

(ii) Outline a CUDA kernel to generate the first 10,000 values of the sequence. Also, show how you would invoke the kernel.

Note that it is impractical to create 10,000 threads per block in CUDA. [12]

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