NATIONAL UNIVERSITY OF SINGAPORE

CS3211 - PARALLEL AND CONCURRENT PROGRAMMING
(Semester 2: AY2014/2015)

Time allowed: 2 hours

INSTRUCTIONS TO CANDIDATES

This assessment paper contains **SIX (6)** sections totalling **FIFTY (50)** marks, and comprises **FOURTEEN (14)** printed pages including this one.

This is an **OPEN BOOK** assessment, and you are to answer **ALL** questions. You may cite any result in the textbook, lecture notes or tutorials. Answer **ALL** questions within the space provided in this booklet (write on the backs of pages if you need more room).

Please write your Student Number below.

STUDENT NO: ______________________________

This portion is for examiners use only.

Question		**Marks**	**Remark**
General topics, short answers	Q1 (6)		
Process algebra and concurrency	Q2 (10)		
Modelling	Q3 (13)		
Java programming	Q4 (7)		
Parallel computing	Q5 (8)		
X10	Q6 (6)		
Total:	Q1-6 (50)		

Q1 (Short Answer Questions) (6 marks)

In the following 6 short questions, write a brief answer in the box provided. Each answer is worth 1 (ONE) mark.

1.1 In a CSP# model you find the code `#alphabet NP1 {enter.0,enter.1,enter.2};` Explain why the modeller/programmer used this line.

Answer:

1.2 Briefly explain the main reason why semaphores are considered an efficient way of handling (for example) mutual exclusion.

Answer:

1.3 Stencils are commonly used for a particular kind of parallel programming architecture/paradigm. Name, and briefly explain the most relevant parallel programming architecture.

Answer:

Q1 (Short Answer Questions) (Continued)

1.4 If a process P has 4 states, and a process Q has 5 states, what is the upper bound on the number of states for $P \,||\, Q$, the parallel composition of P and Q? Justify your answer.

Answer:

1.5 The CSP#/PAT assertion `#assert P() deterministic;` is used for testing if a *process* is deterministic or not. What does it mean for a *process* to be deterministic?

Answer:

1.6 You send your x10 source code to your friend Xerxes. On your computer, the program runs in 2.3 seconds. Xerxes has the same computer but after compiling your source, finds the program takes 20 seconds. What is a plausible/likely reason for the dramatic difference in timing?

Answer:

Q2 (Process algebra and concurrency) (10 marks)

2.1 Using the process definitions below, reduce PQ to its final irreducible form. (5 marks)

$$\begin{aligned} P &= b \rightarrow c \rightarrow d \rightarrow P \\ Q &= b \rightarrow R \\ R &= c \rightarrow Q \,[]\, d \rightarrow Q \\ \\ PQ &= P \,||\, Q \end{aligned}$$

You should use substitution, and the laws of process algebra (attached to the end of this exam), making it clear which laws you are using for each step of your proof.

Answer:

Q2 (Process algebra and concurrency) (Continued)

2.2 Semaphores are useful for managing some concurrency issues. They can be used in two distinct scenarios/ways, one in which we initialize the semaphore to be 0, and the other when we initialize it to some positive value. Explain a situation where we might use two semaphores, one initialized to be 0, and the other to be 10. What is each semaphore used for? (3 marks)

In your answer you should give a (single) example scenario, explaining how and why other processes might access each semaphore.

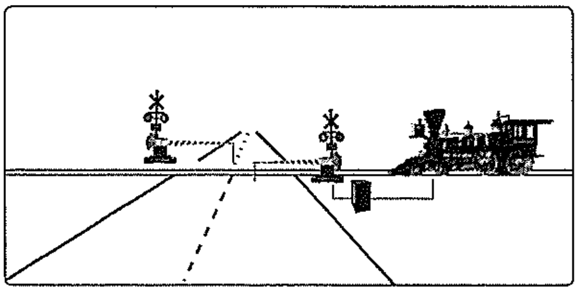
Answer:

2.3 One of the conditions for deadlock is given as "No pre-emption of resources", meaning that resources cannot be forcibly withdrawn by other processes. If this condition was relaxed (i.e. processes can forcibly acquire resources), then no deadlock is possible. Explain what other (negative) impact that relaxing the condition might lead to. (2 mark)

Answer:

Q3 (Modelling) (13 marks)

3.1 One road and one railway track cross each other, and there is a controlled gate which can be lowered to prevent cars crossing the railway track.



If the gate is raised, then cars can freely cross the track. When a train approaches, it signals the controller for the gate, which then lowers the gate. Once the gate is lowered, the controller signals the train to proceed. An obvious safety property for the level crossing: There should never be a train and a car on the crossing at the same time. There are many other properties which we might like to specify, e.g., a liveness property: Whenever a car approaches the crossing, it should eventually be able to cross.

Assume you are going to model this system in CSP. Give the names and brief (one-line) descriptions of the processes, and shared actions you would define. (3 marks)

Answer:

Q3 (Modelling) (Continued)

3.2 For question 3.1 give the CSP for your processes, and one assertion to be verified. (4 marks)

Answer:

Q3 (Modelling)

3.3 Consider the following CSP/PAT model. Draw the Labelled Transition Systems (LTS) for T() and C(), marking/naming the states ($S_o, S_1, \ldots$), and the transitions ($a, p, b, \ldots$). (2 marks)

```
T() = a -> ( p -> T() [] b -> p ->T() );
C() = a -> c -> f -> p ->C();
Test() =   T() || C();
```

Answer:

3.4 In the CSP/PAT model for Q3.3, draw the Labelled Transition System (LTS) for Test(). Identify the states corresponding to the pairs of states from Q3.3 ($S_{o-7}, S_{1-3}, \ldots$), and the transitions ($a, p, b, \ldots$). (4 marks)

Answer:

Q4 (Java programming) (7 marks)

4.1 The following Java code was developed from previous multi-threaded code which kept on throwing `NullPointerExceptions`. The main change was to use a `synchronizedList`. The code *seemed* to be working well until it was delivered to the client, at which point the client reported occasional `NullPointerExceptions` which appeared to happen during the `displayItems` method. Briefly explain what causes the effect, and why the "`synchronizedList`" code did not fix it. (3 marks)

```
List list = Collections.synchronizedList(new ArrayList());
...
Iterator i = list.iterator();
while (i.hasNext())
   displayItems(i.next());
```

Answer:

4.2 Give two different techniques to fix the code. (4 marks)

Briefly describe each of your techniques, showing how you would change the code.

Answer:

Q5 (Parallel computing) (8 marks)

5.1 You are to compute the solution of a problem for each element of a 2-dimensional array of size $n \times n$. Each element is dependent on all the other elements in its column. On a single processor, each computation for each element takes one second, and with no communication overhead, the total time is thus n^2 seconds, an $\mathcal{O}(n^2)$ solution. Splitting the problem onto (say) p separate processors with $p = n$, and with each processor handling (in this case) one column, will reduce the computation time to n seconds (an $\mathcal{O}(n)$ solution). Now assume that processors handle some number of columns (meaning that they can access all the data elements in the column in negligible time), that $p \neq n$, and that p divides n. Calculate the runtime/complexity in terms of p and n, explaining your answer. (2 marks)

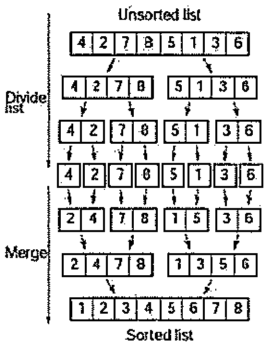
Q5 (Parallel computing) (8 marks)

5.2 In Q5.1, since the processors handle columns of the array, and the data dependencies are restricted to columns, then there is no communication overhead. Consider a variation on this problem, where now each element is now dependent on all the other elements in the same row. As in 5.1, assume that $p \neq n$, that p divides n, and processors handle some number of columns. The communication network is a shared bus, with processor-to-processor communication, and communication from one processor to another has a setup overhead of time t_s, and a per-element overhead of t_e. Calculate the communication overhead in terms of t_s, t_e, p and n, explaining your answer. (4 marks)

5.3 In the scenario described in Q5.2, explain what sort of network/communication hardware might result in reduced communication time. Explain your answer, and give the expected communication time for Q5.2 with the new network. (2 marks)

Q6 (X10) (6 marks)

6.1 Merging two small sorted arrays to make a larger sorted array is efficient and simple to implement. This leads to the parallel merge-sort algorithm which recursively partitions an array into smaller arrays, until there are just two elements left in each array. These two elements are swapped if they are out of order, and then merged with the adjacent two elements and so on. This diagram shows the idea, splitting and then merging an array of 8 elements.



Assuming that you have an array declared as `val ll <: Rail[Long] = [4,2,7,8,5,1,3,6];` and a method `public static def merge(lst:Rail[Long], left:Long, mid:Long, right:Long )` which merges elements using a `left`, `mid` and `right` index. In the example above, `merge(list,4,6,7)` called at step number 4, would merge `<1,5>,<3,6>` to `<1,3,5,6>`. Outline an x10 method `(public static def sort...)` which will sort using the parallel merge-sort algorithm. (4 marks)

Answer:

Q6 (X10) (Continued)

6.2 For your X10 solution in question 6.1, give the maximum and average degree of parallelism obtained for an array of size 2^n, explaining your answers. (2 marks)

Answer:

Laws of process algebra...

Parallel Composition:

$$\begin{array}{rcllr}
P \| Q & = & Q \| P & \text{(symmetric)} & L1_{\|} \\
P \| (Q \| R) & = & (P \| Q) \| R & \text{(associative)} & L2_{\|} \\
P \| \text{Stop} & = & \text{Stop} & & L3A_{\|} \\
(c \rightarrow P) \| (c \rightarrow Q) & = & c \rightarrow (P \| Q) & & L4A_{\|} \\
(c \rightarrow P) \| (d \rightarrow Q) & = & \text{Stop} \quad \text{if } c \neq d & & L4B_{\|} \\
(a \rightarrow P) \| (c \rightarrow Q) & = & a \rightarrow (P \| (c \rightarrow Q)) & & L5A_{\|} \\
(c \rightarrow P) \| (b \rightarrow Q) & = & b \rightarrow ((c \rightarrow P) \| Q) & & L5B_{\|} \\
(a \rightarrow P) \| (b \rightarrow Q) & = & a \rightarrow (P \| (b \rightarrow Q)) [] b \rightarrow ((a \rightarrow P) \| Q) & & L6_{\|}
\end{array}$$

Non-deterministic choice:

$$\begin{array}{rcllr}
P \langle\rangle P & = & P & \text{(idempotent)} & L1_{\langle\rangle} \\
P \langle\rangle Q & = & Q \langle\rangle P & \text{(symmetric)} & L2_{\langle\rangle} \\
P \langle\rangle (Q \langle\rangle R) & = & (P \langle\rangle Q) \langle\rangle R & \text{(associative)} & L3_{\langle\rangle} \\
x \rightarrow (P \langle\rangle Q) & = & (x \rightarrow P) \langle\rangle (x \rightarrow Q) & \text{(prefix distributes)} & L4_{\langle\rangle} \\
P \| (Q \langle\rangle R) & = & (P \| Q) \langle\rangle (P \| R) & & L6_{\langle\rangle} \\
(P \langle\rangle Q) \| R & = & (P \| R) \langle\rangle (Q \| R) & & L7_{\langle\rangle} \\
f(P \langle\rangle Q) & = & f(P) \langle\rangle f(Q) & & L8_{\langle\rangle}
\end{array}$$

General choice:

$$\begin{array}{rcllr}
P [] P & = & P & \text{(idempotent)} & L1_{[]} \\
P [] Q & = & Q [] P & \text{(symmetric)} & L2_{[]} \\
P [] (Q [] R) & = & (P [] Q) [] R & \text{(associative)} & L3_{[]} \\
P [] \text{Stop} & = & P & & L4_{[]} \\
P [] (Q \langle\rangle R) & = & (P [] Q) \langle\rangle (P [] R) & & L6_{[]} \\
P \langle\rangle (Q [] R) & = & (P \langle\rangle Q) [] (P \langle\rangle R) & & L7_{[]}
\end{array}$$

...If P engages alone in a, Q engages alone in b, but c and d require simultaneous participation of both P and Q.

=== END OF PAPER ===