CSE 513 Parallel Runtimes for Modern Processors, Quiz 02 Monsoon 2024
Time allocated: 12:00pm – 12:20pm

Name	
Roll Number	

Instructions:

- This is a closed book and closed notes quiz. Please be aware of strict plagiarism policy.
- For questions requiring justification, please be as concise as possible. 2-3 sentences would be the ideal size of a justification. No extra pages will be provided.

Question 1: (2 mark)
Write two different possible ways a **programmer/user** can control/limit/reduce the overheads in his recursive task parallel Fibonacci to calculate the value of Fib(45) as shown in Figure-1 using four workers?
Answer:

```
int fib(int n) {
  if(n<2) return n;
  future* f1 = async {fib(n-1);};
  future* f2 = async {fib(n-2);};
  return f1.get() + f2.get();
}
```

Figure-1

Note that the question explicitly mentioned *"**programmer/user**…. in his parallel Fibonacci*".
The only two possible ways under programmer's control:

a) Increase the task threshold instead of using n<2. [+1 marks]
b) Execute fib(n-2) sequentially instead of an async. [+1 marks]

Question-2: (2 marks)
The program in Figure-1 during runtime would generate a tree like computation graph, where each node is an async. If the thieves in a parallel runtime has the option to decide which nodes to steal from the tree, what should the priority for stealing? Why?
Answer:
Thieves should prioritize stealing nodes from the top of the tree instead of the bottom of tree [+1 marks], as nodes in the top of the tree would contain more tasks/computation than those in the bottom of tree [+1 marks].

Question-3: (2 marks) Briefly justify if it's possible that R1=1, R2=2, R3=2, and R4=1 in x86-TSO?
Answer: NOT possible [+0.5 marks]. R1=1 & R2=2 implies WR1 happened before WR2. R3=2 & R4=1 implies reordering of RD4a and RD4b which is not possible on x86-TSO [+1.5 marks].

Core-1	Core-2	Core-3	Core-4
WR1: A=1	**WR2**: A=2	**RD3a**: R1=A	**RD4a**: R3=A
		RD3b: R2=A	**RD4b**: R4=A

Question-4: (2 marks)
Consider atomic variables A & B being operated (load/store) using C++11's memory_order_relaxed at Core-0, as shown below. What are the valid set of reordering at that core? Answer using line numbers only.
L1: A.store(1, memory_order_relaxed)
L2: A.load(memory_order_relaxed)
L3: B.load(memory_order_relaxed)
L4: B.store(2, memory_order_relaxed)
Answer: All possible reordering is possible except that L1 must execute before L2 and L3 must execute before L4.
L1,L3,L2,L4; L3,L4,L1,L2; L3,L1,L2,L4; L1,L3,L4,L2; L3,L1,L4,L2; [0.4 x 5 marks]

Question-5: (2 marks)
What are the two things that a full memory barrier (a.k.a. memory fence) specifies?
Answer:

a) Instructions above the fence must execute before the execution of fence instruction. [+1 marks]
b) The store buffer at that core is drained. [+1 marks]