CSC/ECE 506: Architecture of Parallel Computers
Problem Set 3
Due: Friday, July 18, 2025
Reflection due: Friday, July 25, 2025

There will be two submission deadlines for this problem set. For the first submission deadline, you should turn in solutions to all the problems. Then the "official" solutions will be distributed. For the second submission deadline, you should turn in corrections of your work, and explanations of anything you got wrong. The second submission is the one that will be graded.

Problem 1. *(20 points)* "Programmer's intuition" means (i) that instructions execute in program order, and (b) that memory accesses are atomic.

(a) Assume two processors are executing following code. Initial values of `v` and `u` are zero.

P1	**P2**
`a1: v = 10;`	`a4: if (v_available == 1)`
	`u = v*10;`
`a2: v_available = 1;`	`a5: u_available = 1;`
`a3: if (u_available == 1)`	`a6: print v;`
`print u;`	

For this part, do **not** assume that the instructions of P1 and P2 have to execute in order (e.g., `a6` can be executed before `a4`). What are all possible combinations for printing `v` and `u`? For each combination, give a sequence of instruction executions (e.g., `a1` → `a2` → `a3` → `a4` → `a5` → `a6`) that would produce that combination of values.

(b) Of the above results, which combination(s) of (`v`, `u`) is correct according to programmer's intuition?

(c) The other results are impossible according to programmer's intuition. Which of the following is not followed in producing these results, program-order execution or atomicity of memory operations?

Problem 2. *(20 points)* Assume that you have a system with $n = 2$ processors. Answer the following questions given the following code:

```
#pragma parallel for
for (int i = 0; i < n; i++) {
   sum += i;
}
```

(a) *(2 points)* What is the difficulty with this code?

(b) *(3 points)* Modify the code using the OpenMP **critical** clause. Will this fix the problem with the code? Explain why or why not. If not, provide a counterexample.

(c) *(5 points)* Assume you have the following implementation of software locks:

```
void lock (int *lockvar) {
   while (*lockvar == 1) {} ; // wait until released
   *lockvar = 1;
}

void unlock (int *lockvar) {
   *lockvar = 0; // release lock
}
```

Modify the original code to use this lock implementation. Will this work to solve the problem with the code? Explain why or why not. If not, provide a counterexample.

(d) *(5 points)* Assume that we want to use Peterson's algorithm instead of the implementation in part (c) for the lock implementation. Peterson's algorithm is defined as follows:

```
int turn;
int interested[n];

void lock (int process) {
  int other = 1 – process;
  interested[process] = TRUE;
  turn = process;
  while (turn == process && interested[other] == TRUE) {} ;
}

void unlock (int process) {
   interested[process] = FALSE;
}
```

Will this work to solve the problem with the code? Explain why or why not. If not, provide a counterexample.

(e) *(5 points)* Identify at least two potential shortcomings with the implementation of Peterson's algorithm defined in part (d). For each of them, what is a potential solution to mitigate the problem?

Problem 3. *(25 points)* The following program fragment is written for a cache-coherent multiprocessor. All variables are initialized to 0.

P_1	P_2	P_3	P_4
A=1	*u*=*A*	*w*=*A*	*A*=2
	v=A	x=A	

The program uses no shared variables other than A. Suppose that a writer magically knows the location of the cached copies and sends updates directly to them without looking them up in the directory. Construct a situation in which write atomicity may be violated, if an update-based protocol is in use.

(a) Show the violation of sequential consistency that occurs in the results.

(b) Can you construct a scenario where coherence is violated as well? How would you fix these problems?

(c) Can you construct the same scenario for an invalidation-based protocol?

(d) Can you construct it for an update protocol on a bus?

Problem 4. Relaxed consistency models. *(25 points)*

(a) Chris runs this code on a multiprocessor system:

P1	**P2**	**P3**
S1: *Z*=1	S4: *K*=7	S7: *M*=7
S2: *Y*=1	S5: *Z* =3	S8: Print *Z*
S3: Print *K*	S6: Print *M*	

He sees that the final values produced by the code are

Z = 1, *M* = 7 and *K* = 0.

Can you determine under which consistency model (Sequential, Processor and/or PRAM) this output is possible?

(b) The diagram below is not possible with any of the following consistency models: sequential, processor, PRAM or causal.

```
P1:  W(x)1              W(x)2     R(x)3
P2:           R(x) 1    W(x)3     R(x)2
P3:                               R(x)2     R(x)3     R(x)1
```

(i) What change should be made in the order of memory operations in P3 to make it only PRAM and causally consistent?

(ii) After making this change, why is the example still not sequentially consistent?

(c) Causal consistency:

(i) Two events are causally related if one can influence the other. In the code given below, identify instructions that are causally related.

```
P1   S1:A = 2                  S2:A = 3
P2            S3:A = 2 + A
P3            S4:print A                              S5:print A      S6:print A
P4                                        S7:print A  S8:print A      S9:print A
```

(ii) Assuming that *A* is initially 0, specify what is printed by a sequence of events that is valid in causal consistency, but not valid under sequential consistency. Explain why this example is not sequentially consistent.

```
P1   S1:A = 2                  S2:A = 3
P2            S3:A = 2 + A
P3            S4:print A (__)                         S5:print A      S6:print A (__)
P4                                        S7:print A (__)  S8:print A      S9:print A (__)
```

Note: Values returned by `print A` in S5 and S8 must be 4 and 2 respectively.

(d) Weak Ordering:

(i) Here is a part of the program which runs on a multiprocessor that uses weak ordering. What are the possible outcomes under these conditions? Again, all variables are initialized to 0.

P1	**P2**	**P3**
`a=1;`	`while (c==0){};`	`while(b==0){};`
`synch`	`synch`	`print c;`
`c=1;`	`b=1;`	
	`print a;`	
	`print b;`	

(ii) The programmer wants to ensure that `a`, `b` and `c` always print the same value under the same conditions following weak ordering. To achieve this, what additions need to be made to the code in each of the processors.