Quiz 4 - Solution

Fall 2021

Q1 [35 Points]. Assume that we have a multi-core system with these assumptions:

- It has a "release" consistency model (i.e., it is guaranteed that all reads and writes are executed and visible to all cores when the lock is released. Other than this guarantee, any ordering ***outside*** the acquire/release blocks is possible).
- All hazards including both reg-reg and mem-reg are fixed in each processor (i.e., ***none*** of the cores can have WAW/WAR/RAW hazards).

Assume that for the above machine, we have the following instruction sequences:

Initial Values: `M[A]=0, M[B]=0`

THREAD 1	THREAD 2	THREAD 3
`M[A] = 2;`	`while(M[A] == 0);`	`barrier (b1);`
`M[B] = 3;`	`acquire(lock1);`	`acquire(lock1);`
`barrier (b1);`	`print(M[A]);`	`M[A] = 30;`
`barrier (b2);`	`M[B]= 40;`	`print(M[A]);`
`acquire(lock1);`	`release(lock1);`	`M[B] = 50;`
`print(M[B]);`	`barrier (b2);`	`release(lock1);`
`release(lock1);`		

There are four possible outcomes.

T1 prints 40, T2 prints 2, T3 prints 30:

THREAD 1	THREAD 2	THREAD 3
`M[A] = 2;`		
`M[B] = 3;`		
	`while(M[A] == 0);`	
	`acquire(lock1);`	
	`print(M[A]); // 2`	
	`M[B] = 40;`	
	`release(lock1);`	
`barrier(b1);`		`barrier(b1);`
`barrier(b2);`	`barrier(b2);`	
`acquire(lock1);`		
`print(M[B]); // 40`		
`release(lock1);`		
		`acquire(lock1);`
		`M[A] = 30;`
		`print(M[A]); // 30`
		`M[B] = 50;`
		`release(lock1);`

T1 prints 3, T2 prints 2, T3 prints 30:

THREAD 1	THREAD 2	THREAD 3
`M[A] = 2;`		
	`while(M[A] == 0);`	
	`acquire(lock1);`	
	`print(M[A]); // 2`	
	`M[B] = 40;`	
	`release(lock1);`	
`M[B] = 3;`		
`barrier(b1);`		`barrier(b1);`
`barrier(b2);`	`barrier(b2);`	
`acquire(lock1);`		
`print(M[B]); // 3`		
`release(lock1);`		
		`acquire(lock1);`
		`M[A] = 30;`
		`print(M[A]); // 30`
		`M[B] = 50;`
		`release(lock1);`

T1 prints 40, T2 prints 30, T3 prints 30:

THREAD 1	THREAD 2	THREAD 3
M[A] = 2;		
M[B] = 3;		
barrier(b1);		barrier(b1);
		acquire(lock1);
		M[A] = 30;
		print(M[A]); // 30
		M[B] = 50;
		release(lock1);
	while(M[A] == 0);	
	acquire(lock1);	
	print(M[A]); // 30	
	M[B] = 40;	
	release(lock1);	
barrier(b2);	barrier(b2);	
acquire(lock1);		
print(M[B]); // 40		
release(lock1);		

T1 prints 50, T2 prints 2, T3 prints 30:

THREAD 1	THREAD 2	THREAD 3
M[A] = 2;		
	while(M[A] == 0);	
	acquire(lock1);	
	print(M[A]); // 2	
	M[B] = 40;	
	release(lock1);	
M[B] = 3;		
barrier(b1);		barrier(b1);
		acquire(lock1);
		M[A] = 30;
		print(M[A]); // 30
		M[B] = 50;
		release(lock1);
barrier(b2);	barrier(b2);	
acquire(lock1);		
print(M[B]); // 50		
release(lock1);		

Q2 [20 points]. Assume we have the following sequence for a multi-core system with TSO (and LSQ) memory model.

Assume that SUM is a global variable i.e., SUM = M[A];

THREAD 1	THREAD 2	THREAD 3
`SUM = 0 ;`	`pSUM1 = 20;`	`pSUM2 = 30;`
`print(SUM);`	`SUM = SUM + pSUM1;`	`SUM = SUM + pSUM2;`

Add fences, acquire/release, and barriers to this code such that it is guaranteed that T1 always prints 50.

There are multiple valid solutions. Two are given here:

THREAD 1	THREAD 2	THREAD 3
`SUM = 0;`	`pSUM1 = 20;`	`pSUM2 = 30;`
`barrier(b1);`	`barrier(b1);`	`barrier(b1);`
	`acquire(lock1);`	`acquire(lock1);`
	`SUM = SUM + pSUM1;`	`SUM = SUM + pSUM2;`
	`release(lock1);`	`release(lock1);`
`barrier(b2);`	`barrier(b2);`	`barrier(b2);`
`print(SUM);`		

THREAD 1	THREAD 2	THREAD 3
`SUM = 0;`	`pSUM1 = 20;`	`pSUM2 = 30;`
`barrier(b1);`	`barrier(b1);`	
	`SUM = SUM + pSUM1;`	
	`barrier(b2);`	`barrier(b2);`
		`SUM = SUM + pSUM2;`
`barrier(b3);`		`barrier(b3);`
`print(SUM);`		

Q3 [30 points]. Assume we have MOESIF coherency protocol. Assume we have 4 cores and all cores are sharing the memory line `M[Z]`. Complete the following table (show the writebacks with *, and if one cache forwards a value to another cache show it by ^):

If you chose to indicate each writeback at the access that causes the eviction:

Access	P1	P2	P3	P4
R1	E	I	I	I
R2	F^	S	I	I
R2	F	S	I	I
W4	I	I	I	M
R1	S	I	I	O^
R2	S	S	I	O^
W2	I	M	I	I*
All lines are being evicted and invalidated				
	I	I*	I	I
R4	I	I	I	E
W4	I	I	I	M

Or, if you chose to indicate each writeback at the last valid state:

Access	P1	P2	P3	P4
R1	E	I	I	I
R2	F^	S	I	I
R2	F	S	I	I
W4	I	I	I	M
R1	S	I	I	O^
R2	S	S	I	O^*
W2	I	M*	I	I
All lines are being evicted and invalidated				
	I	I	I	I
R4	I	I	I	E
W4	I	I	I	M

Q4. Provide short answers to the following questions:

A. We mention in the class that in modern systems, to maintain coherency, each cache line (i.e., multiple words) has only one coherency state. Describe the advantages and disadvantages of changing this method (i.e., one coherency state per line) to one coherency state per word (i.e., each word in a line has a separate coherency state). [5 points]

   Advantages:
   - Less false sharing, thus:
     - reduced bus traffic
     - higher effective memory bandwidth
     - fewer cache misses
   - Same as subblocking, thus:
     - reduced memory traffic

   Disadvantages:
   - Larger memory array
   - Extra gates needed to select the individual state and update it

B. Suppose you are the VP of the parallel software development group, and you need to decide which type of memory model your product should have. The major product of your company is a very complex system that runs a multi-process parallel application that will be used for weather forecasting to fight against global warming.

   - You can pick machine A that has a weak consistency model where any ordering is possible OR machine B that has TSO.

   Which one would you choose and why (describe what factors should be considered)? [5 points]

Since this is numerical computing software, parallelism can most likely easily be exploited with little communication between threads (compared to the large amount of floating-point arithmetic operations). With a weak model, explicit synchronization mechanisms would thus be infrequent. With TSO, they would be even less frequent.

It is easier to design those synchronization mechanisms for TSO than for a weak memory model. But TSO may have a certain cost in terms of performance and power consumption due to the extra work required by the processor to ensure correctness. TSO may also have an extra monetary cost as the processor may be larger and more complex.

The main question is thus: How much extra developing time and financial cost are acceptable to make the program run slightly faster, or consume slightly less power?

Note that it is not correct to say that a weak model is necessarily less trustworthy than TSO, or to say that a weak model causes errors or unintended results. What is true is that it takes more effort to write correct code on a weak system than on a TSO system.

C. What is your favorite snack? [5 points ;)]

Chips. (pun intended)