Design of Parallel and High Performance Computing
HS 2014
Torsten Hoefler, Markus Püschel
Department of Computer Science
ETH Zurich

Homework 4
Out: 2014-10-16
Revision: 1

Sequential Consitency

Exercise 1

For the following executions, please indicate if they are sequentially consistent. All variables are intially set to 0.

1. P1: W(x,1);
   P2: R(x,0); R(x,1)
2. P1: W(x,1);
   P2: R(x,1); R(x,0);
3. P1: W(x,1);
   P2: W(x,2);
   P3: R(x,1); R(x,2);
4. P1: W(x,1);
   P2: W(x,2);
   P3: R(x,2); R(x,1);
5. P1: W(x,1);
   P2: W(x,2);
   P3: R(x,2); R(x,1);
   P4: R(x,1); R(x,2);
6. P1: W(x,1); R(x,1); R(y,0);
   P2: W(y,1); R(y,1); R(x,1);
   P3: R(x,1); R(y,0);
   P4: R(y,0); R(x,0);
7. P1: W(x,1); R(x,1); R(y,0);
   P2: W(y,1); R(y,1); R(x,1);
   P3: R(y,1); R(x,0);

Solution

1. P2: R(x,0); P1: W(x,1); P2: R(x,1)
2. not sequentially consistent: it is not possible to read 0 from x once the value 1 has been written to it
3. P1: W(x,1); P3: R(x,1); P2: W(x,2); P3: R(x,2);
4. P2: W(x,2); P3: R(x,2); P1: W(x,1); P3: R(x,1);
5. not sequentially consistent: P3 and P4 observe the writes performed by P1 and P2 in different order.
6. P4: R(y,0); R(x,0); P1: W(x,1); P1: R(x,1); P1: R(y,0); P3: R(x,1); P3: R(y,0); W(y,1); R(y,1); R(x,1)
7. not sequentially consistent because based on P1s observations W(x,1) happens before W(y,1), but based on P3s observations W(y,1) must happen before W(x,1)

Exercise 2

P1	**P2**	**P3**
x = 1;	y = 1;	z = 1;
print(y,z);	print(x,z);	print(x,y);

All variables are stored in a memory system which offers sequential consistency. All operations, even the print statements, are atomic. Atomicity means, that no operation can overlap with other operations. Are the following sequences legal outputs?

1. 001011
2. 001111
3. 001110

Explain your answer by showing one possible interleaving of the instructions that might lead to the legal output. In case of an illegal output, explain why no possible interleaving exists.

Solution

1. x = 1;
   print(y,z);
   y = 1;
   print(x,z);
   z = 1;
   print(x,y);
2. x = 1;
   print(y,z);
   y = 1;
   z = 1;
   print(x,z);
   print(x,y);
3. There is no interleaving that outputs 001110. The last statement of any execution prints two different variables because sequential consistency requires program order to be preserved. All three variables are set to 1 before the last print statement, so independent of which variables the last statement prints, both of them are already 1. In conclusion, the last two character printed cannot be 10.

Non-blocking caches

Consider a system with three processors. Each processor has a non-blocking cache (i.e., a processor does not stop upon encountering a cache miss; while the missing cache line is brought in from memory, the processor continues to execute instructions that are not data or control dependent upon a missing load).

The memory locations x and y are originally 0. The processors execute the following sequence:

P1: W(x,1); R(y,0);
P2: W(y,1); R(y,1); R(x,1);
P3: R(y,1); R(x,0);

Is this system sequentially consistent?

Solution

The system is not sequentially consistent. P1: R(y,0) must happen before P2: W(y,1). This implies that P1: W(x,1) also happens before P2: W(y,1). P3: R(y,1) must happen after P2: W(y,1), but at this point of time P2: R(x,0) is not possible anymore (because P1: W(x,1) must have already happened).

The x86 Memory Model: TLO+CC

1. Describe the memory model of the x86 architecture. Where does it differ from sequential consistency.
2. Two threads execute the following code (given in AT&T assembly syntax) on a machine using TLO+CC. Is it possible that the registers eax $= 0 \wedge$ ebx $= 0$ after both threads have finished?. Would it be possible in sequential consistency?

Initial: All registers and memory locations are 0	
Thread A	Thread B
movl \$1, 0x42 movl 0x50, %eax	movl \$1, 0x50 movl 0x42, %ebx

Solution

The x86 memory model provides the following guarantees:

- Reads are not reordered with other reads. (R $\rightarrow$ R)
- Writes are not reordered with other writes. (W $\rightarrow$ W)
- Writes are not reordered with older reads. (R $\rightarrow$ W)
- Reads may be reordered with older writes to different locations but not with older writes to the same location (W $\not\rightarrow$ R) **This point is different from sequential consistency. To still allow synchronized programming, the following guarantees are given**
- Memory ordering obeys causality
- Writes to the same location have a total order.
- In a multiprocessor system, locked instructions have a total order.
- Reads and writes are not reordered with locked instructions.

In sequential consistency, it is impossible that both threads observe eax=0 $\wedge$ ebx=0, however, in TLO+CC this is possible.

First, we will prove that it is impossible in sequential consistency. For this we first translate the above fragments into read and write statements:

Initial: All registers and memory locations are 0	
Thread A	Thread B
W(a, 1)	W(b,1)
R(b, x)	R(a,x)
W(eax, x)	W(ebx, x)

Now we assume that it **is** possible for both threads to finish and have eax=ebx=0 and show that this leads to a contradiction. Without loss of generality, we also assume Thread B is the last one to finish.

The first assumption tells us that the processes must execute the following read/write sequences:

$W_A(a,1) \rightarrow R_A(b,0) \rightarrow W_A(eax,0)$
$W_B(b,1) \rightarrow R_B(a,0) \rightarrow W_B(ebx,0)$

the second assumption translates to

$W_A(eax,0) \rightarrow W_B(ebx,0)$

if we combine the three equations, we see that it is neccessary to have $W_A(a,1) \rightarrow R_B(a,0)$ at the same time $W_B(b,1) \rightarrow R_A(b,0)$ which means that our initial assumption was false, and it is eax=ebx=0 is impossible in sequential consistency.

In the x86 memory model however, it is legal to reorder the given sequences into

Initial: All registers and memory locations are 0	
Thread A	Thread B
R(b, x)	R(a,x)
W(a, 1)	W(b,1)
W(eax, x)	W(ebx, x)

which allows the following interleaving:

$R_A(b,0) \rightarrow R_B(a,0) \rightarrow W_A(a,1) \rightarrow W_B(b,1) \rightarrow W_A(eax,0) \rightarrow W_B(ebx,0)$.