NewLogoIST-eps-converted-to.pdf

INSTITUTO SUPERIOR TÉCNICO

DEPARTAMENTO DE ENGENHARIA INFORMÁTICA

ORGANIZAÇÃO DE COMPUTADORES

LEIC

Conjunto de Exercícios X
Multiprocessadores

Version 1.0

Resoluções

2022/2023

Exercise 1[*]

1.1. Cache coherence concerns the views of multiple processors on a given cache block. The following table shows two processors and their read/write operations on two different words of a cache block X (initially X[0] = X[1] = 0).

P1	P2
`X[0]++; Y = X[0]; X[1] = Y + 2;`	`X[0] = 5; X[1] += 2;`

(a) Assuming that the protocol does not ensure cache coherency, provide an example of an execution that leads to incoherent cache state. For each read/write operation of the sequence, list the values of the block X in the local cache of each processor and in main memory.

(b) For a snooping protocol, list a valid operation sequence on each processor/cache to finish the above read/write operations.

(c) What are the best-case and worst-case numbers of cache misses needed to execute the listed read/write instructions?

Solution:

(a) Incoherent when different caches have different values for the same block.

To reach an incoherent state, we must find a situation where (1) a processor's cache line is updated, (2) the cache line value is modified by another processor, and (3) later the cache line is read by the processor.

Possible answer for incoherent operation sequence:

Time Step	**Event**	**Cache line X for P1**	**Cache line X for P2**	**Memory contents for X**
0	P1: R(X[0])	X[0] = 0, X[1] = 0	–	X[0] = 0, X[1] = 0
1	P1: W(X[0])	X[0] = 1, X[1] = 0	–	X[0] = 1, X[1] = 0
2	P2: W(X[0])	X[0] = 1, X[1] = 0	X[0] = 5, X[1] = 0	X[0] = 5, X[1] = 0
3	P1: R(X[0])	X[0] = 1, X[1] = 0	X[0] = 5, X[1] = 0	X[0] = 5, X[1] = 0
4	P1: W(X[1])	X[0] = 1, X[1] = 3	X[0] = 5, X[1] = 0	X[0] = 1, X[1] = 3
5	P2: R(X[1])	X[0] = 1, X[1] = 3	X[0] = 5, X[1] = 0	X[0] = 1, X[1] = 3
6	P2: W(X[1])	X[0] = 1, X[1] = 3	X[0] = 5, X[1] = 2	X[0] = 5, X[1] = 2

(b) Possible answer (fix to the incoherent operation sequence shown above):

Time Step	**Event**	**Cache line X for P1**	**Cache line X for P2**	**Memory contents for X**
0	P1: R(X[0])	X[0] = 0, X[1] = 0	–	X[0] = 0, X[1] = 0
1	P1: W(X[0]), Inval	X[0] = 1, X[1] = 0	–	X[0] = 1, X[1] = 0
2	P2: W(X[0]), Inval	–	X[0] = 5, X[1] = 0	X[0] = 5, X[1] = 0
3	P1: R(X[0])	X[0] = 5, X[1] = 0	X[0] = 5, X[1] = 0	X[0] = 5, X[1] = 0
4	P1: W(X[1]), Inval	X[0] = 5, X[1] = 7	–	X[0] = 5, X[1] = 7
5	P2: R(X[1])	X[0] = 5, X[1] = 7	X[0] = 5, X[1] = 7	X[0] = 5, X[1] = 7
6	P2: W(X[1]), Inval	–	X[0] = 5, X[1] = 9	X[0] = 5, X[1] = 9

[*]Exercises 5.16.1—5.16.6 from the textbook [?].

(c) Best case: execute one before the other - 2 misses, one for each processor.

Worst case: interleaving writes among the processors (write forces invalidation of other processors' caches) - 2 misses (bootstrap) + 3 misses (write invalidation) = 5 misses

1.2. Memory consistency concerns the views of multiple data items. The following table shows two processors and their read/write operations on different cache blocks (A, B, and X initially 0).

P1	**P2**
`A = 1; if (B == 0) X++;`	`B = 1; if (A == 0) X++;`

(a) List the value of X for an execution sequence of the read/write operations shown in the table. Assume that the system implementation ensures the following consistency assumptions: if a processor writes location x followed by location y, any processor that sees the new value of y must also see the new value of x.

(b) List at least one possible value for X if such assumptions are not maintained.

Solution:

(a) X = 1 (only one gets the lock)

(b) X = 2 (both get the lock)

Exercise 2[†]

Consider the following three CPU organizations:

- **CPU SS**: A 2-core superscalar microprocessor that provides out-of-order issue capabilities on 2 function units (FUs). Only a single thread can run on each core at a time.
- **CPU MT**: A fine-grained multithreaded processor that allows instructions from 2 threads to be run concurrently (i.e., there are two functional units), though only instructions from a single thread can be issued on any cycle.
- **CPU SMT**: An SMT processor that allows instructions from 2 threads to be run concurrently (i.e., there are two functional units), and instructions from either or both threads can be issued to run on any cycle.

Assume we have two threads X and Y to run on these CPUs that include the following operations:

Thread X	**Thread Y**
A1 – takes 3 cycles to execute	B1 – takes 2 cycles to execute
A2 – no dependencies	B2 – conflicts for a functional unit with B1
A3 – conflicts for a functional unit with A1	B3 – depends on the result of B2
A4 – depends on the result of A3	B4 – no dependencies, takes 2 cycles to execute

Assume all instructions take a single cycle to execute unless noted otherwise or they encounter a hazard.

(a) Assume that you have a 2-core SS CPU. How many cycles will it take to execute these two threads? How many issue slots are wasted due to hazards?

[†]Exercises 7.12.1—7.12.3 from the textbook [**?**].

(b) Now assume you have one MT CPU. How many cycles will it take to execute these two threads? How many issue slots are wasted due to hazards?

(c) Assume that you have one SMT CPU. How many cycles will it take to execute these two threads? How many issue slots are wasted due to hazards?

Solution:

(a) Answer:

Core 1	**Core 2**
A3, A2	B2, B4
A1, A4	B1, B4
A1	B1, B3
A1	

(b) Answer 1:

FU 1	**FU 2**
A3	A2
B2	B4
A1	A4
B1	B4
A1	
B1	B3
A1	

(c) Answer:

FU1	**FU2**
A1	B1
A1	B1
A1	B2
A2	B3
A3	B4
A4	B4

Another possible answer:

FU1	**FU2**
A1	B1
A1	B1
A1	B2
A2	B3
B4	A3
B4	A4

References