复习题（2）： 高速缓存、高速缓存一致性、存储器一致性

一、 简答题

1. 高速缓存失效（miss）的 4C 模型，即 4 种 cache 失效的原因，是哪四种？
2. 什么是非阻塞的高速缓存（Non blocking cache）？
3. 什么是 victim cache? Victim cache 和 write buffer 有区别吗？

二、 分析题

1. Here are three ways cache can be organized:

- Direct
- Fully associative
- Set associative

Let us assume that the cache contains 2048 words, 16 words per line(block), and 128 lines.

- Which organization strategy(s):

  (i) Has the least amount of tags needed to be compared for word look-up? Most?

  Direct has the least amount of tags, and full associative has the most amount of tags needed to be compared.

  (ii) Benefits least from using a least recently used (LRU) replacement policy? Most?

  Direct benefits least from using LRU policy, and full associative benefits most.

- If I have a Main-Memory address of [11 0011 0011 0011 0011], what line is the data fetched into with each cache organization? (The answer has to be between 0 and127).

  (i) Direct

  (ii) Fully associative

  (iii) Set associative (with 64 sets)

  Direct: $(011\ 0011)_2=51^{th}$ line

  Fully associative: Possibly into any line

  Set associative: 128/64 = 2 lines per set. $(11\ 0011)_2=51^{th}$ set, thus maybe 102^{th} or 103^{th} line.

2. Consider a unified, 64 Kbyte, direct mapped, instruction and data cache that exhibits the following miss rates as a function of block size, keeping the cache size fixed.

Block Size	Miss Rate
4	35.18%
8	22.57%
16	16.009%
32	12.47%
64	10.79%
128	10.27%
256	10.72%
512	12.75%
1K	16.41%
2K	23.15%
4K	32.97%

a. Why does the miss rate initially decrease as the block size increases? Why does it eventually start increasing again?

Initially increased spatial locality causes a decrease in miss rate that is subsequently overrun by decreased temporal locality.

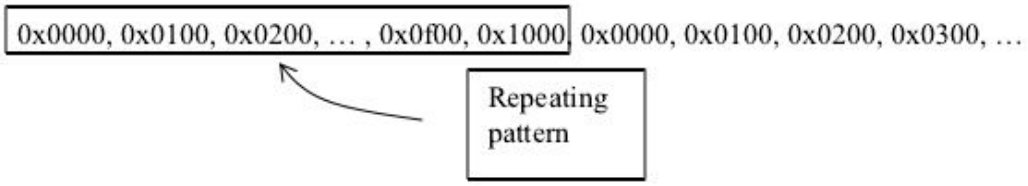
b. Can the 128 byte block size result be guaranteed to provide the best average memory access time? Justify your answer.

No. Smaller block sizes may have slightly higher miss rate (e.g., 64 byte blocks) that may be overshadowed by the much larger miss penalty of a 128 byte block. The answer depends on the design of the cache-memory interface. For average memory access time the key is the behavior of the miss penalty, $(1-h)*t_p$, where t_p is the time to fetch a block from memory and (1-h) is the miss rate.

3.

We have a 4 Kbyte cache operating with 256 byte lines. Consider two designs for this cache. The first is a direct mapped design. The second is a fully associative cache with an LRU replacement policy. Physical addresses are 16 bits. Provide a sequence of physical addresses (in hexadecimal notation) that when repeated indefinitely will result in a lower miss rate in the direct mapped cache than in the fully associative cache. The address stream should be accompanied by an explanation of this behavior.

The following stream of addresses will cause this to happen (the Ox designation signifies hexadecimal notation)



There are sixteen lines in the cache. Have a sequence of 17 addresses that address 17 consecutive blocks of memory and repeat itself. In a fully associative cache

with an LRU policy after the cache as filled up with the first 16 addresses you get the following behavior. The least recently used line is always removed before it is referenced on the next memory reference. Thus, every reference becomes a miss.

In a direct mapped cache, the 17th and 1St reference will always miss but the remaining will always hit. In this case the rigidity of the direct mapped cache causes misses to be localized to two blocks while the flexibility of fully associative placement causes a miss every reference.

4. In the Table 1, we have given you four different sequences of addresses generated by a program running on a processor with a data cache. Cache hit ratio for each sequence is also shown below. Assuming that the cache is initially empty at the beginning of each sequence, find out the following parameters of the processor's data cache:

- Associativity (1, 2 or 4 ways)
- Block size (1, 2, 4, 8, 16, or 32 bytes)
- Total cache size (256 B, or 512 B)
- Replacement policy (LRU or FIFO)

Assumptions: all memory accesses are one byte accesses. All addresses are byte addresses.

Sequence No.	Address Sequence	Hit Ratio
1	0, 2, 4, . 8, 16, 32	0.33
2	0, 512, 1025, 1536, 2048, 1536, 1025, 512, 0	0.33
3	0, 64, 128, 256, 512, 256, 128, 64, 0	0.33
4	0, 512, 1024, 0, 1536, 0, 2048, 512	0.25

Table 1: Sequences of Addresses

Cache block size - 8 bytes: For sequence 1, only 2 out of the 6 accesses (specically those to addresses 2 and 4) can hit in the cache, as the hit ratio is 0.33. With any other cache block size but 8 bytes, the hit ratio is either smaller or larger than 0:33. Therefore, the cache block size is 8 bytes.
Associativity - 4: For sequence 2, blocks 0, 512, 1024 and 1536 are the only ones that are reused and could potentially result in cache hits when they are accessed the second time. Three of these four blocks should hit in the cache when accessed for the second time to give a hit rate of 0.33 (3/9).
Given that the block size is 8 and for either cache size (256 B or 512 B), all of these blocks map to set 0. Hence, an associativity of 1 or 2 would cause at most one or two of these four blocks to be present in the cache when they are accessed for the second time, resulting in a maximum possible hit rate of less than 3/9. However, the hit rate for this sequence is 3/9. Therefore, an associativity of 4 is the only one that could potentially give a hit rate of 0:33 (3/9).

Total cache size - 256 B: For sequence 3, a total cache size of 512 B will give a hit rate of 4/9 with a 4-way associative cache and 8 byte blocks regardless of the replacement policy, which is higher than 0.33. Therefore, the total cache size is 256 bytes.

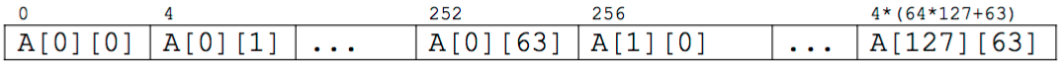
Replacement policy - LRU: For the aforementioned cache parameters, all cache lines in sequence 4 map to set 0. If a FIFO replacement policy were used, the hit ratio would be 3/8, whereas if an LRU replacement policy were used, the hit ratio would be 1/4. Therefore, the replacement policy is LRU.

5. This problem evaluates the cache performances for different loop orderings. You are asked to consider the following two loops written in C, which calculate the sum of the entries in a 128 by 64 matrix of 32-bit integers:

Loop A	***Loop B***

`sum = 0;`
`for (i = 0; i < 128; i++)`
`  for (j = 0; j < 64; j++)`
`  sum += A[i][j];`
	`sum = 0;`
	`for (j = 0; j < 64; j++)`
	`  for (i = 0; i < 128; i++)`
	`  sum += A[i][j];`

The matrix A is stored contiguously in memory in row-major order. Row major order means that elements in the same row of the matrix are adjacent in memory as shown in the following memory layout: A[i][j] resides in memory location [4* (64 * i + j)]. The memory location is as the following Figure shows.



For Problem A to Problem C, assume that the caches are initially empty. Also, assume that only accesses to matrix A cause memory references and all other necessary variables are stored in registers. Instructions are in a separate instruction cache.

Problem A

- Consider a 4 KB direct-mapped data cache with 8-word (32-byte) cache lines.
- Calculate the number of cache misses that will occur when running Loop A.
- Calculate the number of cache misses that will occur when running Loop B.

We can calculate that there're 4*1024/32 = 128 lines, each of which can contain jsut 8 integers. Since the matrix has 128 lines, the number of cache misses of Loop A and Loop B are as follows:
Loop A: 128 *64/8 = 1024 misses.
Loop B: 128*64 = 8192 misses.

Problem B

Consider a direct-mapped data cache with 8-word (32-byte) cache lines.

- Calculate the minimum number of cache lines required for the data cache if Loop A is to run without any cache misses other than compulsory misses.

- Calculate the minimum number of cache lines required for the data cache if Loop B is to run without any cache misses other than compulsory misses.

The minimum number of cache lines required for Loop A is 1024.
Same as Loop A.

Problem C
Consider a 4 KB fully-associative data cache with 8-word (32-byte) cache lines. This data cache uses a first-in/ first-out (FIFO) replacement policy.

- Calculate the number of cache misses that will occur when running Loop A.
- Calculate the number of cache misses that will occur when running Loop B.
- Will a larger cache size help to reduce the miss rate? Why or why not?
- Will a larger block size help to reduce the miss rate? Why or why not?

1024
1024
No. We assume that the block size doesn't change. A larger cache size means that there will be more than 128 cache lines, but the miss rate will be still 1/8. Let's consider the special case, which the number of line is 1024, all the cache misses are compulsory miss. There is one miss of every block (8-word), thus the miss rate is still 1/8.
No. We only consider the situation that the cache size doesn't change. If the block size gets larger, the number of lines will be less than 128, and for Loop B, this will lead to that every memory access will miss, but for Loop A, there's no difference.

6. Consider a processor with a one-level cache with a total cache size of 16 words, making memory references to the following series of word addresses: 1, 4, 8, 5, 20, 17, 19, 56, 9, 11, 4, 43, 5, 6, 9, and 17. Label each reference as a cache hit or cache miss (assuming the cache is initially empty), and show the final contents of the cache, assuming:

1. a direct-mapped cache with 16 one-word blocks
2. a direct-mapped cache with four-word blocks
3. a two-way set associative cache with one-word blocks and LRU replacement
4. a fully-associative cache with one-word blocks and LRU replacement

Solution.
1. gets 3 hits (marked by ! after the address):
   1, 4, 8, 5, 20, 17, 19, 56, 9, 11, 4, 43, 5!, 6, 9!, 17!
2. gets 6 hits
   1, 4, 8, 5!, 20, 17, 19!, 56, 9, 11!, 4, 43, 5!, 6!, 9, 17!
3. gets 4 hits
   1, 4, 8, 5, 20, 17, 19, 56, 9, 11, 4!, 43, 5!, 6, 9!, 17!
4. gets 4 hits
   1, 4, 8, 5, 20, 17, 19, 56, 9, 11, 4!, 43, 5!, 6, 9!, 17!

7. Performance **Impact of Cache Parameters**

Cache performance is evaluated with a number of metrics:

- *Miss rate.* The fraction of memory references during the execution of a program, or a part of a program, that miss. It is computed as # misses / # references.
- *Hit rate.* The fraction of memory references that hit. It is computed as 1- miss rate.
- *Hit time.* The time to deliver a word in the cache to the CPU, including the time for set selection, line identification, and word selection. Hit time is typically 1 to 2 clock cycle for L1 caches.
- *Miss penalty.* Any additional time required because of a miss. The penalty for L1 misses served from L2 is typically 5 to 10 cycles. The penalty for L1 misses served from main memory is typically 25 to 100 cycles.

Average memory access time is a comprehensive measure of performance of cache system.

$$\text{Average memory access time} = \text{Hit time} + \text{Miss rate} \times \text{Miss penalty}$$

This formula gives us three metrics to cache optimizations. Optimizing the cost and performance trade-offs of cache memories is a subtle exercise that requires extensive simulation on realistic benchmark codes and is beyond our scope. However, it is possible to identify some of the qualitative tradeoffs. Fill out the following form below to give a qualitative analysis on the tradeoffs.

	Hit time	Miss rate	Miss penalty	Bandwidth	Power Consumption	Hardware cost/comment
Larger Cache size	+	-			+	
Larger Block size		compulsory misses-, capacity and conflict misses +	+			32 to 64 bytes usually
Higher associativity	+	conflict misses -	+,		+	Expensive to implement and hard to make fast.
Write strategy: write through			Read miss penalty -			Simpler to implement
Write strategy: write back				+		

8. Cache 一致性（8 分）： 在 MESI 协议中 cache 块有 4 种状态，指出每种状态下数据在 cache 中的有效性、数据在存储器中的有效性、块是否在其他 cache 中也有副本的情况，将结果填入表中：（8 分）

	M	E	S	I
Cache 块 的有效性	有效	有效	有效	无效
存储器块的有效性	无效	有效	有效	有效
块在其他 cache 中存在副本的可能性	不可能	不可能	可能	可能

9. 考虑一个由 3 个处理器构成的基于总线的共享存储器系统。该共享存储器被分为 x,y,z,w 四块。每个处理器有一个高速缓存，且在任何给定时间只能安装一块。每一块可能处于两个状态之一：有效（V）或无效（I）。假定高速缓存开始时已被清空，且存储器的内容如下：

存储器块	x	y	z	w
内容	20	30	50	20

考虑遵循以下给定顺序的存储器访问事件序列：

1) P1：Read(x)； 2) P2: Read(x) 3) P3: Read(x) 4) P1: x=x+25 ;
5) P1: Read(z); 6) P2: Read(x) 7) P3: x=15 8) P1: z=z+10

高速缓存的采用的协议为： 写回法和写无效协议。说明在上述的每一次操作后高速缓存和存储器的内容以及高速缓存块的状态。 （15 分）

事件	P1 的高速缓存	P2 的高速缓存	P3 的高速缓存	主存			
				x	y	z	w
初始	(I) 清空	(I) 清空	(I) 清空	20	30	50	20
P1:Read(x)	(V) 20	(I) 清空	(I) 清空	20	30	50	20
P2：Read(x)	(V) 20	(V) 20	(I) 清空	20	30	50	20
P3：Read(x)	(V) 20	(V) 20	(V) 20	20	30	50	20
P1： x=x+25	（V) 45	(I)	(I)	20	30	50	20
P1： Read(z)	(v) 50	(I)	(I)	45	30	50	20
P2： Read(x)	(v) 50	(v) 45	(I)	45	30	50	20
P3： x=15	(v) 50	(I)	(v) 15	45	30	50	20
P1： z=z+10	(v) 10	(I)	(v) 15	45	30	50	20

10、The cache coherence mechanism for a multiprocessor system uses a MESI protocol, where the four states correspond to:

M = modified (modified data; exclusive copy)
E = exclusive (unmodified data; exclusive copy)
S = shared (unmodified; more than one disk has a copy)
I = invalid

Consider a system with two processors, P1 and P2, with the initial cache state shown in the following table. For this problem, assume each cache holds only 4 cache blocks and uses direct-mapped organization.

P1		set	P2	
block	state		block	state
B1	M	0	B5	I
B2	E	1	B6	M
B3	S	2	B3	S
B4	I	3	B8	E

A. What is the state of each cache after the following sequence of memory references is completed? Fill in the table below.

P2 reads block B1 (read the memory block into the cache of P2)
P1 writes block B2
P2 writes block B3
P1 reads block B8

P1		set	P2	
block	state		block	state
B1	S	0	B1	S
B2	M	1	B6	M
B3	I	2	B3	M
B8	S	3	B8	S

B. Assume that the caches are returned again to the original state above. Describe a simple action by P2 that would leave an exclusive copy of block B3 in P1's cache even though the cache state would be S.

P2 read B7.

11: Cache coherence miss

a. In our class we talked about the three C's of caches misses: *capacity misses*, *conflict misses*, and *compulsory misses*. Now in multiprocessors, we can add a fourth C: *coherence misses*. A coherence miss is a cache miss due to another core having invalidated the data in your cache.

Mark whether the following modifications to the cache parameters will cause an **increase, decrease**, or whether the modification will have **no effect** on the number of *coherence misses*. You can assume the baseline cache is set associative. Assume that in each case the other cache parameters (number of sets, number of ways, number of bytes/line) and the rest of the machine design remain the same.

	Coherent miss
Increasing number of bytes per line	**More false sharing, greater # of coherence misses. (Theoretically, could be less if sharing large objects - fewer misses needed to transfer data - but in practice false sharing misses are the dominating factor)**
Increasing number of sets	**No effect to a first order. However can increase slightly as greater probability of holding on to data that causes a coherence miss.**
Increasing number of ways	**Same as above**

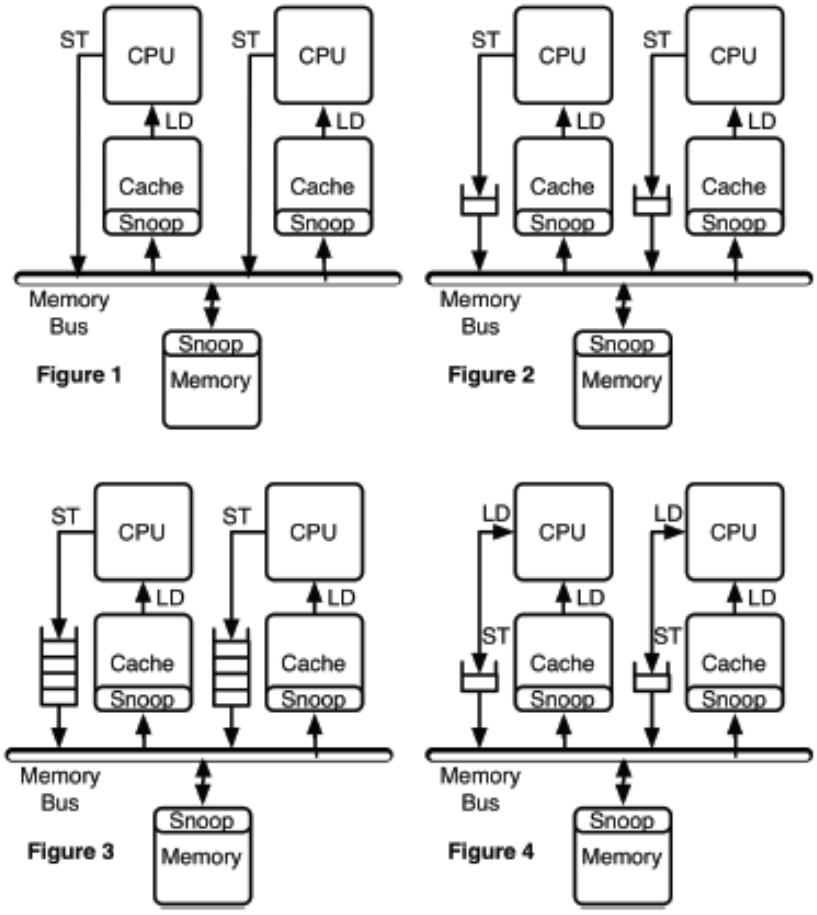
b. In the following problems, we consider a parallel program running on a cache coherent,shared-memory multiprocessor. The code performs vector-vector addition. The integer *P* equals the number of processors running the program, and the integer id contains a given processor's ID number, which lies in the range [0, P-1]. *N* is the vector size; you may assume that *N* is large and that *P* divides *N*.

```
int P, id;
int Z[N], X[N], Y[N];
for(int i = id; i < N; i += P)
Z[i] = X[i] + Y[i];
```

Unfortunately, false sharing eliminates any hope of parallel speedup. **Rewrite the code to avoid false sharing to the extent possible.**

```
for(int i = (N/P)*id; i < (N/P)*(id+1); i++)
Z[i] = X[i] + Y[i];
```

12: Snooping Cache



In this problem we will explore designing coherent caches using snooping. To simplify the protocol, the caches have been made write-through with write update. There is no longer a Modified (M) state in the protocol because any time a store occurs, the data is put out on the memory bus and all interested parties (main memory and all caches that contain that block) can update their value. You may assume that the processor is single cycle and in-order.

To guarantee sequential consistency, when a CPU performs a store, it will stall until its store completes by writing on the bus. As shown in the Figure 1, when a CPU writes, that update goes straight to the bus, and the bus updates the CPU's own cache (as well as main memory and any other copies in other processors' caches).

Since only one component can write to the shared memory bus at a time, a CPU wanting to execute a store may have to wait a large number of cycles. To speed things up, we will add a write buffer. When a CPU performs a store, it is added to the write buffer, and when the bus is available, the store will be broadcast from the write buffer.

A. Initially the write buffer is only a single entry, as shown in Figure 2. After adding a store to the write buffer, the CPU is allowed to continue execution until it reaches any memory operation (load or store). If it reaches a memory operation while there is still a store in the write buffer, it will stall until it is empty. Is this system sequentially consistent (SC)?

Yes it is SC. This scheme allows only one memory operation to be in flight at a time and since they are done in order there is no possibility of them being done out of order.

B. The write buffer is expanded to have multiple entries (Figure 3). The CPU can now continue execution until it encounters a load, where it will stall until the write buffer is empty. It can continue after stores as long as the write buffer is not full, in which case it stalls. Is this system SC?

Yes this is also SC. Even though multiple stores could be in the buffer at the same time, until the CPU executes a load, it can't differentiate whether the stores have completed or not.

C. The write buffer is expanded to have multiple entries (Figure 3). The CPU can now continue execution until it encounters a load, where it will stall until the write buffer is empty. It can continue after stores as long as the write buffer is not full, in which case it stalls. Is this system SC?

Yes this is also SC. Even though multiple stores could be in the buffer at the same time, until the CPU executes a load, it can't differentiate whether the stores have completed or not.

D. Finally, the write buffer is reduced back to a single entry (Figure 4), but the CPU is now allowed to continue execution during loads while the store buffer is full (a store must wait for the previous store to clear the write buffer). When performing a load, the CPU will first look for a matching address in the write buffer and forward the data from the write buffer, and if it is not there, it will check its cache as usual. Is this system SC?

No this could violate SC. When the CPU loads from the write buffer, it is seeing a value before other processors can, which breaks SC.
Consider the following example (initially M[X] = 0 and M[Y] = 0):

CPU 0	**CPU1**
ST X,1	ST Y,1
LD Y	LD X

If the stores stay in the buffer, it is possible for both CPUs to load 0's, but that there is no ordering that is SC that would create this.

---

13 Sequential Consistency (SC) model.

These are three small programs, each executed on a different processor, each with its own cache and register set. In the following **R** is a register and **X** is a memory location. Each instruction has been named (e.g., B3) to make it easy to write answers. Assume data in location X is initially 0. Assuming the program is executing under the Sequential Consistency (SC) model.

Processor A	Processor B	Processor C
A1: ST X, 1	B1: R := LD X	C1: ST X, 6
A2: R := LD X	B2: R := ADD R, 1	C2: R := LD X
A3: R := ADD R, R	B3: ST X, R	C3: R := ADD R, R
A4: ST X, R	B4: R:= LD X	C4: ST X, R
	B5: R := ADD R, R	
	B6: ST X, R	

A. Can X hold value of 4 after all three threads have completed? Please explain briefly.

Yes, A1-A4, B1-B4, C1-C4, B5, B6

B. Can X hold value of 5 after all three threads have completed?

No. All results must be even!

C. Can X hold value of 6 after all three threads have completed?

No. All of C, All of A, All of B

D. For this particular program, can a processor that reorders instructions but follows local dependencies produce an answer that cannot be produced under the SC model?

No. All stores/loads must be done in order because they're to the same address, so no new results are possible.