CS110 sp25 HW5

Due: 2025/5/7 23:59

Complete this homework either by writing neatly by hand or using **typst**. You can find the `.typ` file on Piazza.

1 True or False

Fill in your answer (T or F) in the table below.

1. When the same address is accessed multiple times using a cache, it primarily benefits from spatial locality.
2. If a cache system changes from direct-mapped to N-way set-associative ($N > 1$), the number of index bits in the address breakdown increases.
3. Higher cache associativity decreases hit time, miss rate, and miss penalty simultaneously.
4. Assume a direct-mapped cache with 8-byte blocks, 2 sets, using LRU replacement, and initially empty. Given 8-bit addresses, accessing addresses from `0x00` to `0xFF` sequentially will result in no cache hits.

1	2	3	4
F	F	F	F

2 Set-Associative Caches

Assume a 12-bit address space. All cache lines are shown in the table below.

Set index	Tag 1	Tag 2
0		
1		
2		
3		

1. Assume loading a 16-byte struct at address `0x00F` requires only 1 cache lookup, but loading one at `0x011` requires 2 lookups. Fill in the cache system parameters in the table below.

Cache block size	32 btyes
Total Capacity	256 bytes
Level of set associativity	2-way set associative
# of cache blocks	8
# of sets	4
Address Breakdown (Tag | Index | Offset)	5, 2, 5

2. For the sequence of addresses below, indicate whether each access results in a hit, miss, or replacement. Assume the cache is initially empty.

Address	Cache Access Result (Hit/Miss/Replacement)
0x3A8	Miss
0x1A6	Miss
0x04C	Miss
0x5AD	Replacement
0x3B9	Replacement
0x44F	Miss
0x1B3	Replacement
0x055	Hit
0x241	Replacement
0x45E	Replacement
0x1A1	Hit
0x1A5	Hit
0x642	Replacement
0x444	Hit

3. Calculate the cache hit rate for the sequence of memory accesses above. How can you improve the cache hit rate without increasing the cache **capacity**?

Solution: Change the cache associativity to fully associative.

3 AMAT

Consider a 3-level cache system with the following parameters, where the CPU runs at 4GHz:

Cache Level	Hit Time	Local Miss Rate
L1	2 cycles	8%
L2	12 cycles	35%
L3	44 cycles	10%
Memory	80ns	-

1. Calculate the global miss rate for the 3-level cache.

   Solution: $8\% \times 35\% \times 10\% = 0.28\%$

2. Calculate the average memory access time (AMAT) for the CPU.

   Solution:

   $$\text{AMAT} = 2 \text{ cycle} + 8\% \times (12 \text{ cycle} + 35\% \times (44 \text{ cycle} + 10\% \times (80 \text{ ns})))$$

   Since the CPU runs at 4GHz, a cycle takes 0.25ns.

   $$\text{AMAT} = 0.5 \text{ ns} + 0.08 \times (3 \text{ ns} + 0.35 \times (11 \text{ ns} + 0.10 \times (80 \text{ ns}))) = 1.272 \text{ ns}$$

3. Now, suppose the L1 cache is changed from 2-way set-associative to fully associative. This change reduces the L1 local miss rate to **6%** but increases the L1 hit time by **60%**. Calculate the new AMAT.

   Solution:

   $$\text{AMAT} = 0.8 \text{ ns} + 0.06 \times (3 \text{ ns} + 0.35 \times (11 \text{ ns} + 0.10 \times (80 \text{ ns}))) = 1.379 \text{ ns}$$

4 Code analysis

Consider the following code:

```
struct body {
   float x, y, z, r;
};

struct body bodies[64];

// check whether two physics bodies overlap in 3D space
bool is_collide(struct body a, struct body b);

int check_collision() {
   int count = 0;
   for (int i = 0; i < 64; i++) {
      for (int j = i + 1; j < 64; j++) {
         if (is_collide(bodies[i], bodies[j])) {
            count++;
         }
      }
   }
   return count;
}
```

Note:

- Assume the cache parameters are: 128 bytes capacity, 32 bytes per block, 2-way set associative.
- Elements in the `bodies` array are aligned to the cache lines.
- `sizeof(float) == 4`
- You can ignore what `is_collide` exactly does and assume each body structure is loaded only **once** within the inner loop iteration (i.e., `bodies[i]` and `bodies[j]` are loaded into registers).
- The variables `i`, `j` and `count` are stored in registers.
- Instruction cache is not considered.
- Assume the cache is initially empty.

1. Calculate the hit rate for the above code.

Solution: The size of struct body is 16 bytes, which is exactly the size of a cache line.

$$\text{Total accesses} = \sum_{i=0}^{63} 2 \times (64 - i - 1) = 4032$$

1. For outer loop iterations where $i \leq 54$
   - Each time i reaches an even number (0, 2, 4, ..., 54), we'll have 1 miss
   - This gives us 28 misses from accessing body[i]
   - For the inner loop, we'll have a miss for every other element due to our 2-element-per-block alignment
   - Since i and i+1 loops have the same miss pattern, the total inner loop misses are 972
2. For outer loop iterations where $55 \leq i \leq 56$
   - Total 7 misses.
   - After this two iterations, all elements from body[57] to body[63] are in the cache.
3. For outer loop iterations where $i \geq 57$
   - Total 16 cache lines.
   - when $i \geq 57$, the last 16 elements of body are all in the cache and all accesses are hits.

$$\text{Hit Rate} = 1 - \frac{28 + 972 + 7}{4032} = \frac{3025}{4032} = 75.0248\%$$

2. How can the hit rate be improved by modifying only the code (without changing the cache configuration)? Briefly explain your solution.

Solution:

```
#define tile_size 8
int check_collision() {
  int count = 0;

  for (int ii = 0; ii < 64; ii += tile_size) {
    for (int jj = ii; jj < 64; jj += tile_size) {
      for (int i = ii; i < ii + tile_size && i < 64; i++) {
        for (int j = (i < jj ? jj : i + 1); j < jj + tile_size && j < 64; j++) {
          if (is_collide(bodies[i], bodies[j])) {
            count++;
          }
        }
      }
    }
  }
  return count;
}
```

Use a smaller tile to make sure that the inner loop elements are always in the cache.