Faculty of Media Engineering and Technology
Dept. of Computer Science and Engineering
Dr. Milad Ghantous

CSEN 702: Microprocessors
Winter 2023

Practice assignment 4-Solution

Exercise 1

Suppose that in 1000 memory references there are 40 misses in the first-level cache and 20 misses in the second-level cache.
1) What are the various miss rates?

Assume the miss penalty from the L2 cache to memory is 200 clock cycles, the hit time of the L2 cache is 10 clock cycles, the hit time of L1 is 1 clock cycle, and there are 1.5 memory references per instruction.
2) What is A) the average memory access time and B) average stall cycles per instruction? Ignore the impact of writes.

Solution

1) We can talk about **local** and **global** miss rates. L1 cache has local and global miss rates the same, but for L2 caches and above, the local and global rates will differ.

L1 cache:
The miss rate (either local or global) for the L1 cache is 40/1000 or 4%.

L2 cache:
The local miss rate for the L2 cache is 20/40 or 50%. This means that 50% of the misses in L1, will miss in L2.
The global miss rate of the L2 cache is 20/1000 or 2%. (calculated normally just like in L1)

2) A) From the lecture we know that:

$$\text{Average memory access time} = \text{Hit time}_{L1} + \text{Miss rate}_{L1} \times \text{Miss penalty}_{L1}$$

and

$$\text{Miss penalty}_{L1} = \text{Hit time}_{L2} + \text{Miss rate}_{L2} \times \text{Miss penalty}_{L2}$$

so

$$\text{Average memory access time} = \text{Hit time}_{L1} + \text{Miss rate}_{L1} \times (\text{Hit time}_{L2} + \text{Miss rate}_{L2} \times \text{Miss penalty}_{L2})$$

Avg. memory access time = 1 + 4% x (10 + 50% x 200) = 5.4 clock cycles.

(Explanation: note you have to rely on local miss rates, hence we used the 50% for the L2 cache and not the 2%. Think about it that out of all misses in L1 cache, 50% of them will miss in L2, that's why we have to use the local and not the global)

B) Now, let's compute the avg. stall cycles per instruction

The formula from the lecture is:

$$\text{Average memory stalls per instruction} = \text{Misses per instruction}_{L1} \times \text{Hit time}_{L2} + \text{Misses per instruction}_{L2} \times \text{Miss penalty}_{L2}$$

However, we don't have the misses per instruction value for neither L1 or L2, we must compute them.
It's given that we have 1.5 memory references per instruction, which means for 1000 memory references, we must have (1000/1.5) = 667 instructions then.

- Out of those 667 instructions, it's given that 40 miss in L1 cache, which means on average, the number of misses per instruction in L1 is (40/667) = 0.06 misses per instruction.

- While in L2, 20 miss out of 667, which means, on avg., we have 0.03 misses per instruction.

Plugging those values in the formula above:

→ Avg. mem. stalls per inst. = 0.06 x 10 + 0.03 x 200 = 6.6 clock cycles.

Exercise 2

The transpose of a matrix interchanges its rows and columns; this is illustrated below:

$$\begin{bmatrix} A11 & A12 & A13 & A14 \\ A21 & A22 & A23 & A24 \\ A31 & A32 & A33 & A34 \\ A41 & A42 & A43 & A44 \end{bmatrix} \Rightarrow \begin{bmatrix} A11 & A21 & A31 & A41 \\ A12 & A22 & A32 & A42 \\ A13 & A23 & A33 & A43 \\ A14 & A24 & A34 & A44 \end{bmatrix}$$

This C code is able to perform the above for a 3x3 case.

```
for (i = 0; i < 3; i++)
{
    for (j = 0; j < 3; j++)
    {
        output[j][i] = input[i][j];
    }
}
```

A) What effect does loop interchange have here? Assume much larger matrix sizes.
B) Suggest a blocking mechanism for transposing 256x256 matrices with block size B. (Each block being size BxB)

Solution

A) Before loop interchange, iterations go like this:

```
i=0
Output[0][0] = input [0][0]
Output[1][0] = input [0][1]
Output[2][0] = input [0][2]
i=1
Output[0][1] = input [1][0]
Output[1][1] = input [1][1]
Output[2][1] = input [1][2]
i=2
Output[0][2] = input [2][0]
Output[1][2] = input [2][1]
Output[2][2] = input [2][2]
```

Note that access to the input matrix is efficient as each row is accessed sequentially but the access to the output matrix is not as one item of each row is accessed.
Loop interchange will reverse this, which means the input matrix access will not be efficient while the output ones will be. So loop interchange has no effect in this case.

B) If we apply blocking:

```
for (i = 0; i < 256; i=i+B)
{
    for (j = 0; j < 256; j=j+B)
    {
        for (m=0; m<B; m++)
        {
            for (n=0; n<B; n++)
            {
                output[j+n][i+m] = input[i+m][j+n];
            }
        }
    }
}
```

For example, for B=127, the whole matrix will be processed into 4 steps, shown below sequentially.

Step 1 (128x128)	Step 2 (128x128)
Step 3 (128x128)	Step 4 (128x128)

Exercise 3

Assume a fully associative write-back cache with many cache entries. Cache starts empty. Below is a sequence of five memory operations.

```
Read  Mem[80];
Write Mem[80];
Write Mem[100];
Write Mem[100];
Read  Mem[100];
```

What are the number of hits and misses when using:
A) no-write allocate strategy
B) write allocate strategy?

Solution

A) When using no-write allocate, on a write miss, the data is not brought into the cache.

1. The first read [80] is a miss and hence [80] is brought into the cache.
2. The write to [80] is a hit.
3. Writing to 100 is a miss
4. Writing again to 100 is a miss since 100 is not brought back.
5. Reading 100 is also a miss.

Total: 4 misses and 1 hit.

B) *When using write-allocate, on a write miss, the data is brought into the cache.*

1. The first read [80] is a miss and hence [80] is brought into the cache.
2. The write to [80] is a hit.
3. Writing to 100 is a miss but this time it's brought into cache.
4. Writing again to 100 is a hit now
5. Reading 100 is also a hit

Total: 2 misses and 3 hits.

Exercise 4

Assume the following average miss rates for 32 KB data caches: 3.5% for with a direct-mapped, and 3.2% for two-way set associative.
Assume the miss penalty is 10 cycles. When using “hit under miss”, miss penalty of non-blocking direct mapped compared to blocking direct mapped is reduced by 9%.
What’s better: using a blocking set associative or using non-blocking direct? Compare both against the reference “blocking direct mapped” .

Solution

Average memory stall time:
DM: Miss rate x Miss penalty = 3.5% x 10 = 0.35
2-way: Miss rate x Miss penalty = 3.2% x 10 = 0.32
This means the access latency of the 2-way is (0.32/0.35) = 91% of that of the direct mapped.

If you compare the non-blocking direct mapped to the blocking direct mapped, it says the miss penalty is reduced by 9%, which means also the latency of the non-blocking DM is 91% of that of the blocking DM.

So both approaches are approximately the same.