Imperial College London

Tutorial: Caches

Imperial College London

Department of Computing

C113 Architecture

Lecturer:
Dr. Jana Giceva
j.giceva@imperial.ac.uk

Head Teaching Assistant:
Izaak Coleman
ic711@imperial.ac.uk

Date: March 11, 2018

1 Cache Operation

Assume you are working with a cache with the following characteristics:

- There is only one level of cache.
- Addresses are 8 bits long.
- The block size is 4 bytes.
- The cache is a directly mapped cache.
- The cache has 4 sets.

First, answer the following questions:

1. What is the total capacity of the cache? (in number of data bytes)
   Answer: *The cache size is 16 bytes.*
2. How long is a tag? (in number of bits)
   Answer: *The tag is 4 bits.*

Second, fill in the missing information in the following table. You can assume that the cache starts clean (cold). Addresses are given in both hexadecimal and binary format for your convenience.

Operation	Set index?	Hit or Miss?	Eviction?
`load 0x00` $(0000\ 0000)_2$	0	miss	no
`load 0x04` $(0000\ 0100)_2$	1	miss	no
`load 0x08` $(0000\ 1000)_2$	2	miss	no
`store 0x12` $(0001\ 0010)_2$	0	miss	yes
`load 0x16` $(0001\ 0110)_2$	1	miss	yes
`store 0x06` $(0000\ 0110)_2$	1	miss	yes
`load 0x18` $(0001\ 1000)_2$	2	miss	yes
`load 0x20` $(0010\ 0000)_2$	0	miss	yes
`store 0x1a` $(0001\ 1010)_2$	2	hit	no

2 Miss rate analysis

Listed below are two matrix multiply functions. The first, `matrix_multiply` computes $C = A \cdot B$, a standard matrix multiply. The second, `matrix_multiply_t`, computes $C = A \cdot B^T$, A times the transpose of B.

```
void matrix_multiply(float A[N][N], float B[N][N], float C[N][N])
{
    /* Computes C = A*B. Assumes C starts as all zeros. */
    int i, j, k;
    for (i=0; i<N; i++) {
        for(j=0; j<N; j++) {
            for (k=0; k<N; k++) {
                C[i][j] += A[i][k] * B[k][j];
            }
        }
    }
}

void matrix_multiply_t(float A[N][N], float B[N][N], float C[N][N])
{
    /* Computes C = A*transpose(B). Assumes C starts as all zeros. */
    int i, j, k;
    for (i=0; i<N; i++) {
        for(j=0; j<N; j++) {
            for (k=0; k<N; k++) {
                C[i][j] += A[i][k] * B[j][k];
            }
        }
    }
}
```

Our assumptions are the following: The cache is 8 MiB with 16-way associativity and a block size of 64 bytes. N is very large, so that a single row or column cannot fit in the cache. The `sizeof(float) == 4` and `C[i][j]` is stored in a register.

1. What miss rate do you expect the function `matrix_multiply` to have for large values of N?
   Answer: *There are 16 floats in a block. For row-wise A, the pattern is 1 miss, 15 hits, 1 miss, 15 hits, ... For col-wise B, the pattern is miss, miss, miss, ... With the assumption that C[i][j] is in a register, then there are zero memory accesses to C[i][j] during each inner loop. So the miss rate is* $\frac{17}{32}$.

2. What miss rate do you expect the function `matrix_multiply_t` to have for large values of N?
   Answer: *For row-wise A, B, the pattern is 1 miss, 15 hits, 1 miss, 15 hits, ... Therefore, the miss rate is* $\frac{2}{32} = \frac{1}{16}$.