Advanced Systems Lab

Homework 4

Albert Cerfeda (20-980-272)

Contents

9.04.2024
Eidgenössische Technische Hochschule Zürich
D-INFK Department
Switzerland

4.1 Stride access (30 pts)

a) Direct-mapped cache, $s = 1$ $n = 8$

The cache is direct-mapped, has blocks of size 32 bytes and a total capacity of 1 KiB. This means that there are $\frac{2^{10}}{32} = 32$ cache blocks. Each cache block/line contains 32 bytes (4 doubles).

The access pattern of matrix O is the following: Each element is accessed sequentially from row 0 to row $n - 3 = 5$. Therefore in total there are 12 compulsory cache misses, two for each row of O.

Regarding the access pattern of A instead, at each iteration, the corresponding elements at row $i - 1$ and $i + 1$ are accessed. Similar to before we only have 16 compulsory cache misses, two for each row of A.

We have a total of 192 memory accesses. 28 of which are compulsory cache misses, after which all the rows of A and O are in the cache. Therefore the cache miss rate is $\frac{28}{192} = 14.583333\%$.

b) Direct-mapped cache, $s = 2$ $n = 16$

The cache structure is analagous as before. The main difference is that since matrix A and O are 16×16, the cache can only store exactly half of one of them at a time. Therefore, elements from A and O with the same coordinates map to the same cache location.
By increasing the stride to 2, not every element of A is accessed. In fact, the columns are covered in an alternating pattern, resulting in only half of the elements being accessed. Matrix O is still accessed in a sequential manner.

In the first 4 iterations, the cache gets repeatedly evicted because the first 4 elements of O share the same cache block as the first elements of A. After the initial 4 iterations, this light form of thrashing stops, resulting only in compulsory cache misses.

We have 112 iterations and a total of 448 memory accesses, 107 of which are cache misses, therefore the cache miss rate is $\frac{107}{448} = 23.883929\%$.

c) 2-way associatice cache, $s = 2$ $n = 16$

Since the cache is 2-way associative, it has 16 sets with 2 cache lines each. Each row of A and O is 16 doubles, meaning that two cache blocks are needed to store one full row.

We have 112 iterations and a total of 448 memory accesses, 92 of which are cache misses, therefore the cache miss rate is $\frac{92}{448} = 20.535714\%$.

4.2 Cache mechanics (20 pts)

a) Access patterns and state of direct-mapped cache

i. Cache at line 18

i = 0
x: **MMM**
y: **MMM**

Set	Block 0
0	$y_2.a$ $y_2.b$
1	-
2	-
3	-
4	$x_2.a$ $x_2.b$
5	-
6	-
7	-

i = 1
x: **HHH**
y: **MHH**

Set	Block 0
0	$y_2.a \quad y_2.b$
1	-
2	$y_3.a \quad y_3.b$
3	-
4	$x_2.a \quad x_2.b$
5	-
6	-
7	-

ii. Cache at line 30

i = 0
x: **MMM**
y: **MMM**

Set	Block 0
0	$y_2.a \quad y_2.b$
1	$y_6.c \quad y_6.d$
2	$y_3.a \quad y_3.b$
3	$x_5.c \quad x_5.d$
4	$x_2.a \quad x_2.b$
5	$y_0.c \quad y_0.d$
6	-
7	-

i = 1
x: **MMM**
y: **MHM**

Set	Block 0
0	$y_2.a \quad y_2.b$
1	$y_6.c \quad y_6.d$
2	$y_3.a \quad y_3.b$
3	$y_3.c \quad y_3.d$
4	$x_2.a \quad x_2.b$
5	$x_2.c \quad x_2.d$
6	-
7	-

b) Access patterns and state of 2-way associative cache

i. Cache at line 18

i = 0
x: **MMM**
y: **MMM**

Set	Block 0	Block 1
0	$x_2.a \quad x_2.b$	$y_2.a \quad y_2.b$
1	-	-
2	-	-
3	-	-

i = 1
x: **HHH**
y: **MHH**

Set	Block 0	Block 1
0	$x_2.a \quad x_2.b$	$y_2.a \quad y_2.b$
1	-	-
2	$y_3.a \quad y_3.b$	-
3	-	-

ii. Cache at line 30

i = 0
x: **MMM**
y: **MMM**

Set	Block 0	Block 1
0	$x_2.a \quad x_2.b$	$y_2.a \quad y_2.b$
1	$y_6.c \quad y_6.d$	$y_0.c \quad y_0.d$
2	$y_3.a \quad y_3.b$	-
3	$x_5.c \quad x_5.d$	-

i = 1
x: **MMH**
y: **MMH**

Set	Block 0	Block 1
0	$x_2.a \quad x_2.b$	$y_2.a \quad y_2.b$
1	$x_2.c \quad x_2.d$	$y_6.c \quad y_6.d$
2	$y_3.a \quad y_3.b$	-
3	$x_5.c \quad x_5.d$	$y_3.c \quad y_3.d$

4.3 Rooflines (40 pts)

a) Roofline plot

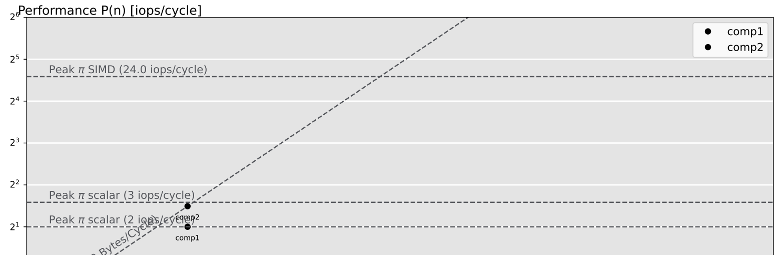

Figure 1: Roofline plot

b) Scalar performance bounds

Both functions perform the same amount of work and memory reads, that is $16n^2$ bytes read and $3n^2$ iops.

Based on the instruction mix, we notice how in `comp1` there is only one port for the `MAX` instruciton, resulting in a bottleneck. Because of it, the compute bound for `comp1` is $\pi = 2$. On the other hand, `comp2` does not use the `MAX` instruction, resulting in the highest possible scalar compute bound of $\pi = 3$.

Considering the rooflines for the two functions, we can see that `comp1` is compute bound, while `comp2` is memory bound. Therefore the hard bound on performance on `comp1` is 2 iops/cycle and on `comp2` is 2.8125 iops/cycle.

c) SIMD performance bounds

For this exercise we assume that every instruction has a SIMD counterpart. As such, since our architecture has 256-bit SIMD register, that allows us to perform 8-way integer operations. This results in the compute bound being higer, resulting in `comp1` and `comp2` having a speedup of 1.40625 and 1.0 respectively.

4.4 Cache miss analysis (25 pts)