Homework 3: Solutions Dominik Gresch

Exercise 1: Cache mechanics

a) The general solution is given by

$$M = \frac{1}{2(bs+t)}\left(s + \lceil t/b \rceil + \sum_{j=0}^{\lceil t/b \rceil - 1} \Theta\left(\sum_{i=(j+1)b-1}^{bs+t-1} \delta_{\beta_i, s+j}\right)\right.$$
$$\left. + \sum_{j=0}^{\lceil t/b \rceil - 1} \Theta\left(\sum_{i=\min[b(s+j+1)-1, bs+t-1]}^{2bs+t-1} \delta_{\beta_i, j}\right) + \sum_{j=0}^{\lceil t/b \rceil - 1} \delta_{\alpha_i, \beta_{i-1}} \cdot \left(1 - \delta_{\gamma_i, 0}\right)\right)$$

where

$$\Theta(x) = \begin{cases} 0 & \text{if } x \leq 0 \\ 1 & \text{else} \end{cases}$$

$$\begin{aligned} \alpha_i =& \lfloor (bs + ((bi)\%(bs+t)))/b \rfloor \%2s \\ \beta_i =& \lfloor ((bi)\%(bs+t))/b \rfloor \%2s \\ \gamma_i =& (i\%(bs+t))\%2s \end{aligned}$$

and δ is the Kronecker Delta. α_i and β_i describe in which set `y` and `x` are at iteration `i`, δ_i the block offset of `y`.

The first 2 terms come from the misses that happen because `y` gets loaded into cache the first time.

The third term comes from cacheblocks at the beginning of `y` that have been evicted between the time when `y` first accessed them and the time `y` accesses them again (second part of the loop).

The fourth term is the same, but with the end of `y` that overlaps with `x` (in terms of cache address).

Finally, the last term describes cache misses due to a "race condition", between `x` and `y`, those are misses where `x` and `y` are at the same cache location at the same time (but not when `y` accesses the first element of a block, because those are already accounted for).

I have provided a `Mathematica` file for evaluation: see `/ex01/ex01.nb`
As a last remark, for `(s,b,t)` which fulfil the condition $\mathtt{t} \leq \mathtt{b} \leq \mathtt{s}$ or $\mathtt{b} + \mathtt{t} \leq \mathtt{s}$ (I only tested these limits numerically), this term becomes much easier:

$$M = \frac{s + \lceil t/b \rceil}{2(bs+t)}$$

b) `x`: for the first 7 `i`, there are only misses (access the first element of a cache block each time). After that, there will be 6 hits (accessing the second element). For $\mathtt{i} \in \{\mathtt{13}, \mathtt{25}\}$, there will be misses only for the elements that have been evicted due to conflicts with `y` (i.e. `i = 13, 19`)

⇒ `x` : `MMMMMMMHHHHHH MHHHHHMHHHHHH`

y: For the first 13 i, there are alternating misses and hits. After that, there will be a miss only for i = 13 and i = 25 (the elements that have been evicted due to conflict with x).

$$\Rightarrow \texttt{y} : \texttt{MHMHMHMHMHMHM MHHHHHHHHHHHM}$$

c) Since sum is in a register, the traffic will be exactly 2 doubles · #misses. There are 18 misses, hence

$$Q = 18 \cdot 2 \cdot 8 \texttt{ bytes} = 288 \texttt{ bytes}$$

There are 2 flops in each loop iteration, 26 iteration steps

$$\Rightarrow W = 52 \texttt{ flops} \Rightarrow I = \frac{13}{72} \frac{\texttt{flop}}{\texttt{byte}}$$

Exercise 2: Cache mechanics

a) The upper bound for `I` is determined by the minimum number of bytes read (reading the three variables exactly once) and the number of flops.

$$Q \geq (12 + 4 + 3)\ \texttt{doubles} = 152\ \texttt{bytes}$$
$$W = 12\ \texttt{add} + 12\ \texttt{mult} = 24\ \texttt{flop}$$
$$\Rightarrow I \leq \frac{12}{76}\ \frac{\texttt{flop}}{\texttt{byte}}$$

b) In every iteration of the innermost loop, there is fist an access to `A[i][j]` and `x[j]` and then an access to `y[i]` (the multiplication gets evaluated first).

Each cache block fits exactly one `double`.

Looking only at `A` and `x`, each even `j` fills up the `0` - th set and each odd `j` fills up the `1`st set. Since `A` does not use the same value twice and `x` only uses the same value after 4 `j` (when all previous values have already been evicted), there are no cache hits for either `A` or `x`. This results in 12 misses for `A` and 12 misses for `x`.

For `y`, there are two different scenarios. If `i % 2 == 0` (i.e. `y[i]` is in the first set), the hit/miss pattern for `j = 0,...,3` is `MHMH` (2 misses). If `i % 2 == 1` (i.e. `y[i]` is in the second set), the miss/hit pattern is `MMHM` (3 misses). Because the first scenario happens twice and the second once, this gives 7 misses for `y`.

$$M_{L_1} = \frac{12 + 12 + 7}{36}\ \frac{\texttt{misses}}{\texttt{acces}} = \frac{31}{36}\ \frac{\texttt{misses}}{\texttt{access}}$$

c) Every read miss creates 8 `bytes` of traffic. Additionally, `y` has to be written to memory exactly the same number of times as misses for `y` (each time the value gets brought in it will eventually have to be written back).

$$Q = (31 + 7) \cdot 8\ \texttt{bytes} = 304\ \texttt{bytes}$$

of traffic, and hence

$$I_{L_1} = \frac{3}{38}\ \frac{\texttt{flop}}{\texttt{byte}}$$

d) The variables all fit into the L_2 - cache (no conflicts): `A` takes up one block per set for the first 6 sets, `x` and `y` fit into the two blocks of the 7-th and 8-th set respectively.
The only misses come from bringing the variables in once. Since each block contains 2 `doubles`, this gives rise to

- $\frac{12}{2}\ \texttt{misses} = 6\ \texttt{misses}$ for `A`
- $\frac{4}{2}\ \texttt{misses} = 2\ \texttt{misses}$ for `x`
- $\lceil \frac{3}{2} \rceil\ \texttt{misses} = 2\ \texttt{misses}$ for `y`

$$\Rightarrow 10\ \texttt{misses} \Rightarrow M_{L_2} = \frac{10}{3 \cdot 12}\ \frac{\texttt{misses}}{\texttt{access}} = \frac{5}{18}\ \frac{\texttt{misses}}{\texttt{access}}$$

e) Memory traffic comes from reading each variable and writing `y`. For `y`, there is an additional `double` of memory traffic since it doesn't fit perfectly into the second block (effectively, 4 `double` of traffic for each complete read / write of `y`).

$$\Rightarrow Q = (12 + 4 + 4 \cdot 2) \cdot 8\ \texttt{byte} = 192\ \texttt{byte} \Rightarrow I_{L_2} = \frac{1}{8}\ \frac{\texttt{flop}}{\texttt{byte}}$$

Exercise 3: Associativity

I do not agree with your statement. Counterexample:

Let x, y and z be 3 arrays of size B/8 (they fit exactly into one block), and their addresses such that (for the direct mapped cache), x gets mapped into the 0 - th set, y and z into the S/2 - th. Of course, for the 2 - way associative cache, they will all be mapped into the 0-th cache set. Now consider the following code (assuming sum is in registers, otherwise cold cache):

```
double sum = 0;
for(int i = 0; i < B / 8; ++i) {
      sum += x[i];
      sum += y[i];
      sum += z[i];
}
```

Whilst the direct mapped cache produces hits for x (after the initial miss), and misses only for y and z, the 2 - way associative cache produces only misses.
direct mapped: MMMHMMHMM...
2 - way associative: MMMMMMMMM...
The reason for this is that x stays in cache in the direct mapped case whilst it gets evicted (because it is least recently used) in the 2 - way associative case.

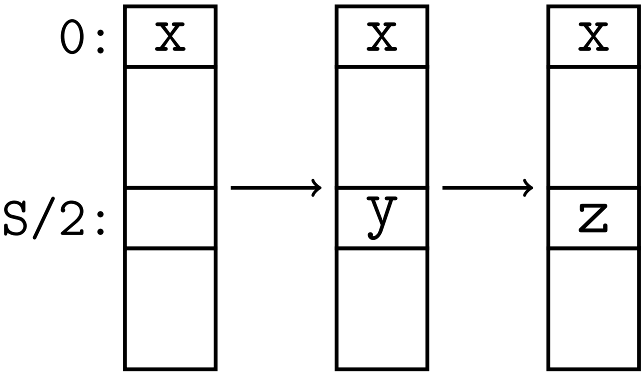

Figure 1: first loop iteration with direct mapped cache

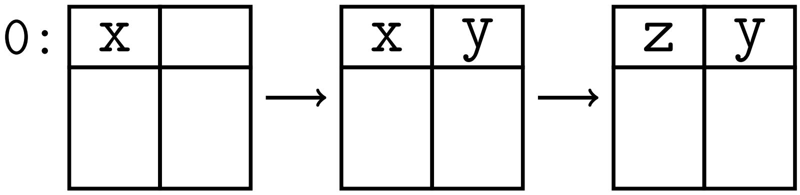

Figure 2: first loop iteration with 2 - way associative cache