Assignment 1

Aniket Sanghi (170110)

18 September 2020

I pledge on my honor that I have not given or received any unauthorized assistance.

1 Solution 1

```
double s = 0.0, A[size];
int i, it, stride;
for (it = 0; it < 10 * stride; it++) {
        for (i = 0; i < size; i += stride) {
                s += A[i];
        }
}
```

We know from the given data,

$$\text{Number of words per block} = 32B/8B = 4$$

$$\text{Cache Size} = 256KB$$

$$\text{Array Size} = size * wordsize = 32K * 8B = 256KB$$

This implies that the array fits perfectly in cache so, we won't have any conflict/capacity misses.
We need to only take care of cold misses. Hence when the array is loaded into memory, we won't get any more misses and hence the loop corresponding to *it* (outer loop) won't add to misses. We just need to calculate the misses in the inner loop which can be calculated by the following formula

$$\text{Number of cache misses} = \frac{\text{Number of array accesses}}{\text{Number of cache hits per miss} + 1}$$

$$\text{Number of array accesses} = \frac{size}{stride}$$

$$\text{Number of hits per miss} + 1 = Max(\frac{\text{No. of words per block}}{stride}, 1)$$

The below table shows the calculation and the number of misses for different values of stride

Stride	Number of Array Access	Number of hits per miss + 1	Total Number of Cache Misses
1	32K	4	8K
4	8K	1	8K
16	2K	1	2K
32	1K	1	1K
2K	16	1	16
8K	4	1	4
32K	1	1	1

Table 1: Total number of cache misses for the corresponding stride

2 Solution 2

The cache has a capacity of 32K words while each array has 256K elements. Thus, $1/8^{th}$ of the array can fit into the cache. This means that elements (i, j) and (i + 64, j) will map to the same cache block in a direct-mapped cache. If the inner loop accesses an array by column, by the time 1/4th columns are accessed, the elements in the first 1/8 column would have been removed from the cache due to conflict misses.
$N = 512, B = 8$

Array A: For a fixed i and k, as j is varied same element is accessed again and again so there will be only 1 cold miss and rest hits. As k is varied row of A will be accessed linearly resulting in N/B cold misses. Similar costs are incurred for each outer iteration i as different rows of A are accessed. So the total number of misses is N*N/B for both direct mapped cache and fully-associative cache(No benefit of associativity here).

	A	B	C
i	512	512	512
k	64	512	1
j	1	64	64
Total Misses	32K	16M	32K

Table 2: Cache Misses for ikj with Direct Mapped Cache

Array B: For a fixed i and k, as j is varied row of B will be accessed linearly resulting in N/B cold misses. Similar costs are incurred for each iteration k as different rows of B are accessed. Then as i is varied each loop will incurs similar cost will incur as the starting 7/8th of the elements have been replaced so no reuse. So the total number of misses is N*N*N/B for both direct mapped cache and fully-associative cache(No benefit of associativity here).

	A	B	C
i	512	512	512
k	64	512	1
j	1	64	64
Total Misses	32K	16M	32K

Table 3: Cache Misses for ikj with Fully Associative Cache

Array C: For a fixed i and k, as j is varied, row of C will be accessed linearly resulting in N/B cold misses. . As k is varied row of A will be accessed repeatedly linearly resulting in all hits as full row perfectly fits in cache. Similar costs are incurred for each outer iteration i as different rows of C are accessed. So the total number of misses is N*N/B for both direct mapped cache and fully-associative cache(No benefit of associativity here).

Array A: For a fixed j and i, as k is varied, row of A will be accessed linearly resulting in N/B cold misses. Similar costs are incurred for each outer iteration i as different rows of A are accessed. Similar cost for each j as 7/8th of array will have been evicted by conflicts resulting in no reuse. So the total number of misses is N*N*N/B for both direct mapped cache and fully-associative cache(No benefit of associativity here).

Array B: For fixed j and i, as k is varied, subsequent elements in a column of B are accessed. For each iteration of i, there will be no misses for fully-associative cache since all elements fits in cache associativity but there will be N misses in direct-mapped as when 1/4th column elements are stored 1/8th of the columns elements are evicted. When j is changed by one, the adjacent column of B is accessed, but will incur misses for a direct mapped cache (hits for a fully associative cache since only N/B lines would be used). So misses for a direct-mapped cache will be N*N*N, and (N/B)*1*N for a fully-associative cache.

	A	B	C
j	512	512	512
i	512	512	512
k	64	512	1
Total Misses	16M	128M	256K

Table 4: Cache Misses for jik for Directly Mapped Cache

	A	B	C
j	512	64	64
i	512	1	512
k	64	512	1
Total Misses	16M	32K	32K

Table 5: Cache Misses for jik for Fully Associative Cache

Array C: For fixed j and i, as k is varied same element is accessed again and again so only 1 miss and rest all hits. Then as i is varied, subsequent elements in a column of C are accessed. When j is changed by one, the adjacent column of C is accessed, but will incur misses for a direct mapped cache (hits for a fully associative cache since only N/B lines would be used). So misses for a direct-mapped cache will be N*N*1, and (N/B)*N*1 for a fully-associative cache.

3 Solution 3

Given, A direct Mapped Cache

$$\text{Cache Size} = 16MB/8B = 2M = 2^{21}\text{words}$$
$$\text{Block Size} = 64B/8B = 8\text{words}$$
$$\text{2D array Size} = 4096 \times 4096 = 2^{24}\text{words}$$

The cache has a capacity of 2M words while each array has 16M elements. Thus, $1/8^{th}$ of the array can fit into the cache.
$N = 4K, B = 8$

Array A: For a fixed k and j, as i is varied different columns of A is accessed each being a miss. As j is varied the consecutive columns of A are accessed and all being miss as by the time previous 1/4 columns of the elements are added 1/8th of previous are evicted. Similar cost for each k. Giving total of N*N*N misses.

Array X: For a fixed k and j, as i is same element is accessed giving all hits and only 1 miss. As j is varied there will be N/B misses (1 cold miss for every B accesses) for the row. As k is varied successive rows will be accessed with similar costs. Giving total of N*(N/B)*1 misses.

	A	X	y
i	4K	1	512
j	4K	512	1
k	4K	4K	1
Total Misses	64G	2M	512

Table 6: Cache Misses for each array