CS610 - Programming for Performance
2020-2021-I: Assignment 1

Name: Sarthak Singhal Roll No.: 170635

1. **Problem 1**

Size of array$A = 32K * 8B = 256KB$, which is same as the capacity of cache. So, after first iteration of the outer loop, there will no miss as the elements that are going to be accessed after the first iteration will be already present in the cache. So, now we just have to consider the inner loop.
Blocksize $= 32B = 4words$(where $1word = 8B$). So when $A[0]$ will be accessed then $A[1], A[2], A[3]$ will also be fetched in the same block.
Hence, if $stride = 1$, then there will a miss in every 4 iterations. Number of misses $= 32K/4 = 8K$.
If $stride \geq 4$ then the number of misses will be equal to the number of iterations of the inner loop, as in each iteration the accessed element will not be in the cache as a block can contain only $4words$. So, the number of cache misses will be equal to the no. of iterations $= size/stride = 32K/stride$.
$stride = 4 \rightarrow misses = 32K/4 = 8K$
$stride = 16 \rightarrow misses = 32K/16 = 2K$
$stride = 32 \rightarrow misses = 32K/32 = 1K$
$stride = 2K \rightarrow misses = 32K/2K = 16$
$stride = 8K \rightarrow misses = 32K/8K = 4$
$stride = 32K \rightarrow misses = 32K/32K = 1$

2. **Problem 2**

Given: $N = 512$, size of any of the array $= 512 * 512 = 256K$, number of sets in case of direct-mapped cache $= cachesize/blocksize = 32K/8 = 4K$.
Consider direct-mapped cache, block size is 8 words, so $A[0][0]...A[0][7]$ will belong to same block and set 0. $A[0][8]...A[0][15]$ will map to set 1. Similarly first row of A will occupy $512/8 = 64$ sets i.e. till set 63. Then $A[1][0]$ will map to set 64. In general, $A[k][0]$ will map to set $(64 * k)$ mod $4K$. So when $64k = 4K$ i.e. $k = 64$, then $A[64][0]$ will map to set 0 and will evict the block present there i.e. $A[0][0]...A[0][7]$.

Analysis for ikj form:

Direct-mapped cache

Loop	**A**	**B**	**C**
i	N = 512	N = 512	N = 512
k	N/8 = 64	N = 512	1
j	1	N/8 = 64	N/8 = 64
Total	$N^2/8 = 32768$	$N^3/8 = 16777216$	$N^2/8 = 32768$

Fully-associative cache

Loop	**A**	**B**	**C**
i	N = 512	N = 512	N = 512
k	N/8 = 64	N = 512	1
j	1	N/8 = 64	N/8 = 64
Total	$N^2/8 = 32768$	$N^3/8 = 16777216$	$N^2/8 = 32768$

Array A: For fixed i and k, if $A[i][k]$ is not present in the cache then it will be loaded in first iteration of j. For rest iterations of j, $A[i][k]$ will be already there in the cache. So, the multiplier for j is 1. Now keeping i fixed, and iterating over k means we are accessing i^{th} row of A. As the block size is 8 there will be a miss in every 8 iterations, leading to $N/8$ multiplier for k. Now iterating over i means iterating over each row of A, which will lead to multiplier of N as every row will be fetched first time in the cache. So, answers for A are same for both type of caches.

Array B: For fixed i and k, accessing $B[k][j]$ means iterating over k^{th} row of B, which will lead to $N/8$ misses. Next keeping i fixed, iterating over k means iterating over different rows of B. This will lead to multiplier of N for k. Now when i is varied, temporal reuse is not possible even with a fully-associative cache because of insufficient capacity to hold all of B till i changes. This will lead to multiplier of N for i.

Array C: For fixed i and k, accessing $C[i][j]$ means accessing i^{th} row of C, which will lead to $N/8$ misses. As k is varied same row will be accessed resulting in hits because both type of caches can store a single row of C. So multiplier for k is 1. When i is varied then similar costs are incurred for each iteration of i, which leads to multiplier of N for both types of caches.

Analysis for jik form:

Direct-mapped cache

Loop	**A**	**B**	**C**
j	N = 512	N = 512	N = 512
i	N = 512	N = 512	N = 512
k	N/8 = 64	N = 512	1
Total	$N^3/8$ = 16777216	N^3 = 134217728	N^2 = 262144

Fully-associative cache

Loop	**A**	**B**	**C**
j	N = 512	N/8 = 64	N/8 = 64
i	N = 512	1	N = 512
k	N/8 = 64	N = 512	1
Total	$N^3/8$ = 16777216	$N^2/8$ = 32768	$N^2/8$ = 32768

Array A: For fixed j and i, accessing $A[i][k]$ means iterating over i^{th} row of A, which will lead to $N/8$ misses. Next keeping j fixed, iterating over i means iterating over different rows of A. This will lead to multiplier of N for i. Now when j is varied, temporal reuse is not possible even with a fully-associative cache because of insufficient capacity to hold all of A till j changes. This will lead to multiplier of N for j.

Array B: For fixed j and i, as k is varied, subsequent elements in a column of B are accessed leading to N misses. When i is varied then same column of B will be accessed which will lead to hit in fully-associative cache as only N blocks are used while total blocks in cache are $4K$(i.e. $32K/8$) leading to multiplier of 1 for i in fully-associative cache. Whereas, in case of direct-mapped cache varying i will lead to misses as fixing j, $(i + 64)^{th}$ and i^{th} row will map to same set (which is shown in starting of this solution) which will lead to eviction of the blocks which are going to be used again. So, multiplier will be N for i in direct-mapped cache. When j is changed by 1, the adjacent column will be accessed. In case of fully-associative cache there will be a multiplier of $N/8$ as values of next 7 columns will be present in the cache. While in case of direct-mapped cache values of the next column will be present only for some last rows which will lead to multiplier of N for j.

Array C: For fixed j and i, if $C[i][j]$ is not present in the cache then it will be loaded in first iteration of k. For rest iterations of k, $C[i][j]$ will be already there in the cache. So, the multiplier for k is 1. Now keeping j fixed and varying i, subsequent elements in a column of C are accessed leading to multiplier of N. When j is changed by 1, the adjacent column will be accessed. In case of fully-associative cache there will be a multiplier of $N/8$ as values of next 7 columns will be already there in the cache. While in case of direct-mapped cache values of the next column will be present only for some last rows which will lead to multiplier of N for j.

3. Problem 3

Given: $N = 4096$, size of A and $X = 4096 * 4096 * 8 = 128MB$, number of sets = cache size/block size = $16MB/64B = 256K$.
Block size is 8 words, so $A[0][0]...A[0][7]$ will belong to same block and set 0. $A[0][8]...A[0][15]$ will map to set 1. Similarly first row of A will occupy $4096/8 = 512$ sets i.e. till set 511. Then $A[1][0]$ will map to set 512. In general, $A[k][0]$ will map to set $(512 * k)$ mod $256K$. So when $512k = 256K$ i.e. $k = 512$, then $A[512][0]$ will map to set 0 and will evict the block present there i.e. $A[0][0]...A[0][7]$.

Loop	**A**	**X**
k	N = 4096	N = 4096
j	N = 4096	N/8 = 512
i	N = 4096	1
Total	N^3 = 68719476736	$N^2/8$ = 2097152

Array A: For fixed k and j, as i is varied, subsequent elements in column of A are accessed leading to N misses. When j is changed by 1, the adjacent column is accessed, there will be misses as the next column will be present only for some last rows which will lead to multiplier of N for j. When k is varied, no temporal reuse is possible as there is insufficient capacity to hold all of A till the outer loop k changes, leading to multiplier of N.

Array X: For fixed k and j, if $X[k][j]$ is not present in the cache then it will be loaded in first iteration of i. For rest iterations of i, $X[k][j]$ will be already there in the cache. So, the multiplier for i is 1. Now keeping k fixed, and iterating over j means we are accessing k^{th} row of X. As the block size is 8 words there will be a miss in every 8 iterations, leading to $N/8$ multiplier for j. Now iterating over k means iterating over each row of X, which will lead to multiplier of N as every row will be fetched first time in the cache.