# CS610: Programming For Performance

Assignment 1
Devesh Shukla (200322)

August 26, 2024

## Problem 1

Cache capacity = 256 KB, associativity = 8, line size = 64B, word size = 4B
Number of words in a block = BL = line size / word size = 64/4 = 16
Number of elements in array A = size = 32K
Size of array A = 32K words * 4B per word = 128KB

Since the size of array A is less than the cache capacity, the whole array would fit in the cache and hence, there will be no capacity misses or conflict misses. So, we need to consider only the cold misses that occur during the first iteration of the outer loop as all other accesses in later iterations would result in a cache hit.

For stride = 1, we access all 32K elements of A and since BL = 16, every 16th access will result in a cold miss.
So, total number of misses = 32K/16 = 2K

For stride = 4, we access 8K (=32K/4) elements of A and since BL = 16, every 4th access will result in a cold miss.
So, total number of misses = 8K/4 = 2K

For stride = 16, we access 2K (=32K/16) elements of A and every access will result in a cold miss as stride $\geq$ BL.
So, total number of misses = 2K

For stride = 64, we access 512 (=32K/64) elements of A and every access will result in a cold miss as stride $\geq$ BL.
So, total number of misses = 512

For stride = 2K, we access 16 (=32K/2K) elements of A and every access will result in a cold miss as stride $\geq$ BL.
So, total number of misses = 16

For stride = 8K, we access 4 (=32K/8K) elements of A and every access will result in a

cold miss as stride $\geq$ BL.
So, total number of misses = 4

For stride = 16K, we access 2 (=32K/16K) elements of A and every access will result in a cold miss as stride $\geq$ BL.
So, total number of misses = 2

For stride = 32K, we access 1 (=32K/32K) element of A and every access will result in a cold miss as stride $\geq$ BL.
So, total number of misses = 1

# Problem 2

Cache size = 64K words, N = 1024
Number of words in a block (BL) = 8
Total number of blocks in the cache = 64K/8 = 8K
Total number of words in matrix A (or B or C) = 1024 * 1024 = 1M.
So, a row of any matrix can fit in the cache, but not the whole matrix.
As 64K/1M = 1/16, at a time, only 1/16th of any matrix can fit in the cache.

**kij form**

For A:
As j is varied, no new element of A will be accessed, so the multiplier corresponding to j will be 1. As i is varied, columns in successive rows of A are accessed. Each of these will be a cold miss, so the multiplier for i is N. As k is varied, successive columns of A get accessed. As BL = 8, every 8th iteration of the outermost loop will result in all cold misses. So, the multiplier for k is N/BL. Therefore, total number of cache misses = $N^2/BL = 2^{17}$.

For B:
As j is varied, successive elements of B in the same row get accessed leading to a cold miss for every $BL^{th}$ access. So, the multiplier for j is N/BL. As i is varied, no new elements of B are accessed and since the cache is big enough to hold a single row of B, no cache misses occur. So, the multiplier for i is 1. As k is varied, same number of misses repeat as new rows of B are accessed. So, the multiplier for k is N.
Therefore, total number of cache misses = $N^2/BL = 2^{17}$.

For C:
As j is varied, successive elements of C in the same row get accessed leading to a cold miss for every $BL^{th}$ access. So, the multiplier for j is N/BL. As i is varied, successive rows of C are accessed and so the same number of misses repeat leading to a multiplier of N for i. As k is varied, all these misses repeat as the cache is not big enough to fit in all of the array C inside it. Therefore, the multiplier for k is N. So, total number of cache misses = $N^3/BL = 2^{27}$.

As is clear from above, the associativity of the cache does not play any role in determining the number of misses for this form of matrix multiplication.

Cache miss summary for direct-mapped cache

| | A | B | C |
|---|---|---|---|
| k | N/BL | N | N |
| i | N | 1 | N |
| j | 1 | N/BL | N/BL |
| Total | $N^2/BL$ | $N^2/BL$ | $N^3/BL$ |

Cache miss summary for fully associative cache

| | A | B | C |
|---|---|---|---|
| k | N/BL | N | N |
| i | N | 1 | N |
| j | 1 | N/BL | N/BL |
| Total | $N^2/BL$ | $N^2/BL$ | $N^3/BL$ |

**jik form**

For A: As k is varied, a row of A is accessed, hence a multiplier of N/BL. As i is varied, different columns of A are accessed, each having the same number of misses, hence a multiplier of N. As j is varied, all the misses will be repeated as the cache cannot contain the whole matrix, hence a multiplier of N. This analysis does not depend on associativity of cache. So, total number of cache misses = $N^3/BL = 2^{27}$ for both direct mapped and associative caches.

For B:
As k is varied, different columns of B are accessed, hence a multiplier of N. This would be the same for direct-mapped and fully-associative caches.
As i is varied, the same column of B will be accessed as B is not indexed by i. For a direct-mapped cache, when a new iteration of loop i starts, only the last 1/16th of the column will be in the cache as every 1/16th part of the matrix maps to the same sets. Hence, all the accesses will be misses for a direct-mapped cache, hence a multiplier of N. For a fully-associative cache, a whole column can fit in the cache as it can accomodate a total of 8K blocks. Hence, there will be no cache misses except for the first iteration, hence a multiplier of 1.
As j is varied, different columns are accessed in the loop i. For a direct-mapped cache, every iteration will have the same number of cache misses (except for the first iteration) as the required rows of the column would have been evicted leading to conflict misses as described above, hence a multiplier of N. For a fully-associative cache, as a whole column fits in the cache, every $BL^{th}$ access will be a miss, hence a multiplier of N/BL.
Hence, a total of $N^3 = 2^{30}$ misses for a direct-mapped cache and $N^2/BL = 2^{17}$ cache misses for a fully-associative cache.

For C:
No new element will be accessed as k is varied, so the multiplier corresponding to k will be 1. As i is varied, different columns of C are accessed, hence a multiplier of N. Both of these will be same for direct mapped as well as fully associative caches.
As j is varied, different columns are accessed in the loop i. As in the case of B, this will result in a multiplier of N for a direct-mapped cache due to conflict misses as only 1/16th of a column will remain in the cache, whereas it will result in a multiplier of N/BL for a fully-associative cache.
Hence, a total of $N^2 = 2^{20}$ misses for a direct-mapped cache and $N^2/BL = 2^{17}$ misses for a fully-associative cache.

Cache miss summary for direct-mapped cache

| | A | B | C |
|---|---|---|---|
| j | N | N | N |
| i | N | N | N |
| k | N/BL | N | 1 |
| Total | $N^3/BL$ | $N^3$ | $N^2$ |

Cache miss summary for fully associative cache

| | A | B | C |
|---|---|---|---|
| j | N | N/BL | N/BL |
| i | N | 1 | N |
| k | N/BL | N | 1 |
| Total | $N^3/BL$ | $N^2/BL$ | $N^2/BL$ |

# Problem 3

Cache capacity = 16 MB, block size = 32B, word size = 8B
Number of words in a block = BL = block size / word size = 32/8 = 4

Max number of elements in cache = 16 MB/8B = 2M = $2^{21}$
Number of elements in vector y = 4096 = $2^{12}$
Number of elements in array A = Number of elements in array X = 4096 * 4096 = $2^{24}$
So, the cache can hold whole of y but, it cannot hold more than 1/8th of the total number of elements of either array A or X.

For A:
As i is varied, elements of A get accessed in a column-wise fashion. So, all of these accesses result in cold misses as $N \geq BL$. So, the multiplier for i is N. As j is varied, different columns of A are accessed, but all of these will lead to conflict misses . It is because the cache is not big enough to hold more than 1/8th of array A at any time, so by the time elements of the adjacent column are accessed, their previously cached lines would have been evicted. So, the multiplier for j is also N. As k is varied, these steps repeat. So, the multiplier for k is also N.
So, the total number of misses for A = $N^3 = 2^{36}$

For X:
As i is varied, no new element of X is accessed, so the multiplier for i is 1. As j is varied, elements of X are accessed in a row-wise order. So, the multiplier for j is N/BL. As k is varied, successive rows of X get accessed which leads to same number of misses occuring for each row. Hence, the multiplier for k is N.
So, the total number of misses for X = $N^2/BL = 2^{22}$

For y:
As i is varied, successive elements of y stored contiguously in memory get accessed. So, the multiplier for i is N/BL. As j and k are varied, previously accessed elements of y are accessed again and since, the cache is big enough to store all elements of y at the same time, no further cache misses occur. So, the multiplier for both j and k is 1.
So, the total number of misses for y = N/BL = $2^{10}$

# Problem 4

(i)

| A block | B block | C block | $\frac{L1misses(blk)}{L1misses(seq)}$ | $\frac{L2miss(blk)}{L2miss(seq)}$ | Speedup |
|---|---|---|---|---|---|
| 4 X 4 | 4 X 4 | 4 X 4 | 6.114 | 3.231 | 3.704 |
| 4 X 4 | 4 X 4 | 8 X 8 | 3.52 | 2.342 | 2.894 |
| 4 X 4 | 4 X 4 | 16 X 16 | 1.305 | 0.918 | 2.814 |
| 4 X 4 | 4 X 4 | 32 X 32 | 1.064 | 2.454 | 2.593 |
| 4 X 4 | 8 X 8 | 4 X 4 | 9.368 | 7.832 | 3.469 |
| 4 X 4 | 8 X 8 | 8 X 8 | 4.627 | 2.516 | 3.793 |
| 4 X 4 | 8 X 8 | 16 X 16 | 1.317 | 1.428 | 3.070 |
| 4 X 4 | 8 X 8 | 32 X 32 | 1.071 | 2.414 | 2.561 |
| 4 X 4 | 16 X 16 | 4 X 4 | 17.272 | 15.767 | 4.468 |
| 4 X 4 | 16 X 16 | 8 X 8 | 6.076 | 4.206 | 3.955 |
| 4 X 4 | 16 X 16 | 16 X 16 | 1.312 | 2.952 | 3.043 |
| 4 X 4 | 16 X 16 | 32 X 32 | 1.073 | 2.254 | 2.687 |
| 4 X 4 | 32 X 32 | 4 X 4 | 23.544 | 23.171 | 4.581 |
| 4 X 4 | 32 X 32 | 8 X 8 | 6.809 | 6.289 | 4.138 |
| 4 X 4 | 32 X 32 | 16 X 16 | 1.303 | 6.037 | 3.152 |
| 4 X 4 | 32 X 32 | 32 X 32 | 1.065 | 3.948 | 2.521 |
| 8 X 8 | 4 X 4 | 4 X 4 | 5.760 | 8.102 | 3.627 |
| 8 X 8 | 4 X 4 | 8 X 8 | 3.705 | 3.092 | 3.658 |
| 8 X 8 | 4 X 4 | 16 X 16 | 1.264 | 1.969 | 2.773 |
| 8 X 8 | 4 X 4 | 32 X 32 | 1.045 | 7.809 | 2.477 |
| 8 X 8 | 8 X 8 | 4 X 4 | 9.207 | 13.554 | 4.236 |
| 8 X 8 | 8 X 8 | 8 X 8 | 4.764 | 3.876 | 4.208 |

| | | | | | |
|---|---|---|---|---|---|
| 8 X 8 | 8 X 8 | 16 X 16 | 1.278 | 3.381 | 2.979 |
| 8 X 8 | 8 X 8 | 32 X 32 | 1.055 | 4.130 | 2.450 |
| 8 X 8 | 16 X 16 | 4 X 4 | 16.127 | 25.831 | 4.877 |
| 8 X 8 | 16 X 16 | 8 X 8 | 6.162 | 6.207 | 4.312 |
| 8 X 8 | 16 X 16 | 16 X 16 | 1.287 | 6.064 | 3.205 |
| 8 X 8 | 16 X 16 | 32 X 32 | 1.060 | 3.808 | 2.629 |
| 8 X 8 | 32 X 32 | 4 X 4 | 24.108 | 37.880 | 5.303 |
| 8 X 8 | 32 X 32 | 8 X 8 | 6.980 | 11.811 | 4.530 |
| 8 X 8 | 32 X 32 | 16 X 16 | 1.295 | 10.494 | 3.193 |
| 8 X 8 | 32 X 32 | 32 X 32 | 1.061 | 4.927 | 2.646 |
| 16 X 16 | 4 X 4 | 4 X 4 | 3.575 | 9.471 | 4.316 |
| 16 X 16 | 4 X 4 | 8 X 8 | 3.001 | 3.222 | 4.141 |
| 16 X 16 | 4 X 4 | 16 X 16 | 1.190 | 3.094 | 2.884 |
| 16 X 16 | 4 X 4 | 32 X 32 | 1.024 | 10.205 | 2.524 |
| 16 X 16 | 8 X 8 | 4 X 4 | 5.826 | 17.048 | 4.824 |
| 16 X 16 | 8 X 8 | 8 X 8 | 4.140 | 5.733 | 4.375 |
| 16 X 16 | 8 X 8 | 16 X 16 | 1.237 | 5.679 | 3.140 |
| 16 X 16 | 8 X 8 | 32 X 32 | 1.044 | 5.124 | 2.698 |
| 16 X 16 | 16 X 16 | 4 X 4 | 10.334 | 33.029 | 5.419 |
| 16 X 16 | 16 X 16 | 8 X 8 | 5.630 | 10.706 | 4.384 |
| 16 X 16 | 16 X 16 | 16 X 16 | 1.268 | 9.770 | 3.232 |
| 16 X 16 | 16 X 16 | 32 X 32 | 1.054 | 4.734 | 2.663 |
| 16 X 16 | 32 X 32 | 4 X 4 | 16.689 | 49.572 | 5.448 |
| 16 X 16 | 32 X 32 | 8 X 8 | 6.631 | 19.706 | 4.685 |
| 16 X 16 | 32 X 32 | 16 X 16 | 1.286 | 16.378 | 3.160 |
| 16 X 16 | 32 X 32 | 32 X 32 | 1.057 | 5.830 | 2.665 |
| 32 X 32 | 4 X 4 | 4 X 4 | 4.019 | 7.467 | 4.482 |
| 32 X 32 | 4 X 4 | 8 X 8 | 3.159 | 4.610 | 3.947 |
| 32 X 32 | 4 X 4 | 16 X 16 | 1.169 | 4.806 | 2.866 |
| 32 X 32 | 4 X 4 | 32 X 32 | 1.013 | 9.412 | 2.575 |
| 32 X 32 | 8 X 8 | 4 X 4 | 6.247 | 13.011 | 4.708 |
| 32 X 32 | 8 X 8 | 8 X 8 | 4.252 | 8.152 | 4.348 |
| 32 X 32 | 8 X 8 | 16 X 16 | 1.220 | 7.784 | 3.303 |
| 32 X 32 | 8 X 8 | 32 X 32 | 1.039 | 6.278 | 2.795 |
| 32 X 32 | 16 X 16 | 4 X 4 | 10.788 | 24.970 | 5.348 |
| 32 X 32 | 16 X 16 | 8 X 8 | 5.717 | 14.245 | 4.554 |
| 32 X 32 | 16 X 16 | 16 X 16 | 1.254 | 13.201 | 3.156 |
| 32 X 32 | 16 X 16 | 32 X 32 | 1.049 | 5.520 | 2.605 |
| 32 X 32 | 32 X 32 | 4 X 4 | 17.492 | 37.410 | 5.376 |
| 32 X 32 | 32 X 32 | 8 X 8 | 6.726 | 25.531 | 4.700 |
| 32 X 32 | 32 X 32 | 16 X 16 | 1.277 | 20.658 | 3.194 |
| 32 X 32 | 32 X 32 | 32 X 32 | 1.0527 | 5.742 | 2.657 |

Dimensions of integer matrices A (and B and C) = 2048 X 2048

The fourth column in the above table denotes the ratio of number of L1 cache misses in the sequential version to those in the blocking version for various blocksize configurations. Similarly, the fifth column in the table denotes the ratio of number of L2 cache misses for the sequential version of matrix multiplication to the number of L2 cache misses for the blocking version.

(ii)

PAPI version used - 7.1.0.0

Performance counters used -
(a) L1 data cache misses (PAPI_L1_DCM)
(b) L2 data cache misses (PAPI_L2_DCM)

Cache hierarchy of the system:
(a) 32KB L1 cache (per core)
(b) 256KB L2 cache (per core)
(c) 12MB shared L3 cache

From the table in (i), it is clear that the best speed up is obtained for the following configuration :
Blocksize for A = 16 X 16
Blocksize for B = 32 X 32
Blocksize for C = 4 X 4

From the same table, we can see that for these block sizes, the ratio of L2 cache misses in the sequential version to the blocking version is the highest among all configurations. This indicates that for this case, the number of L2 cache misses is reduced by the biggest factor by using blocking. Moreover, the ratio of L1 cache misses in the sequential version to the blocking version is also one among the highest among all mentioned configurations. So, this configuration also reduces L1 cache misses by a significantly high factor.
So, we can clearly observe the direct impact that number of cache misses (measured by performance counters) has on the time taken for the program. The best blocking configuration is able provide a speedup of of around 5.5 by just utilising the cache hierarchy to its advantage.