CS610 Semester 2024–2025-I
Assignment-1

Sandeep Nitharwal
(210921)

August 26, 2024

Problem.1

Consider the following loop.

```
1    float s = 0.0, A[size];
2    int i, it, stride;
3    for (it = 0; it < 1000 * stride; it++) {
4        for (i = 0; i < size; i += stride) {
5            s += A[i];
6        }
7    }
```

Assume an 8-way set-associative cache with a capacity of 256 KB, line size of 64 B, and word size of 4 B (for `float`). The cache is empty before execution and uses an LRU replacement policy. Given `size`=32K, determine the total number of cache misses on `A` for the following access strides: 1, 4, 16, 64, 2K, 8K, 16K, and 32K. Consider all the three kinds of misses: cold, capacity, and conflict.

Solution
Given
Cache Capacity = 256KB
Block Size = 64 B
Word Size = 4 B
Array size = 32K words

Now,
Array size = 32K * 4 B = 128KB which is less than Cache Capacity. Therefore, Only cold misses will be present, no capacity as well as conflict misses.
Number of words per Block = BLK = 64/4 = 16 words.

Total Number of cache misses on A for strides = 1
For strides = 1, we are accessing all the elements of A. For each block we will have a cold miss for the 1st element rest all other element will give us a hit. So, there are 16 words per block, resulting in 1 miss per 16 words.
Total miss = Array size/ BLK = 32K/16 = **2K** misses for the 1st iteration.
Now, Since all the element are cached, so no miss for the other iterations.
Total Cold misses = **2K**
Total Capacity misses = **0**
Total Conflict misses = **0**

Similarly,
For strides = 4 and 16, there will be 4 and 1 access correspondingly per block so all the blocks are accessed with 1 miss per block, resulting in 32K/16 = 2K misses.

Now, for strides >16 there will be 1 miss per strides no. of words. Resulting in (Array size)/(strides) misses.

Strides	Total Misses
1	2K
4	2K
16	2K
64	512
2K	16
8K	4
16K	2
32K	1

Problem.2

Consider a cache of size 64K words and lines of size 8 words. The matrix dimensions are 1024×1024. Perform cache miss analysis for the *kij* and the *jik* forms of matrix multiplication (shown below) considering direct-mapped and fully associative caches. The arrays are stored in row-major order. To simplify the analysis, ignore misses from cross-interference between elements of different arrays (i.e., perform the analysis for each array, ignoring accesses to the other arrays).

Listing 1: kij form

```
for (k = 0; k < N; k++)
  for (i = 0; i < N; i++)
    for (j = 0; j < N; j++)
      C[i][j] += A[i][k] * B[k][j];
```

Listing 2: jik form

```
for (j = 0; j < N; j++)
  for (i = 0; i < N; i++)
    for (k = 0; k < N; k++)
      C[i][j] += A[i][k] * B[k][j];
```

Your solution should have a table to summarize the total cache miss analysis for each loop nest variant and cache configuration, so there will be four tables in all. Justify your computations.

Solution

Given

Cache size = 64K words

Block size = BLK = 8 words

matrix size = 1024*1024 = 1024K words

Now, since matrix size > cache size so we can store only 64K/1024K = $\frac{1}{16}^{th}$ of the matrix at a time.

kij form Analysis

For A:

As j is varied, there is no relation with A so we can ignore it.

As i is varied, we are not able to use temporal locality as previous blocks are replaced, hence a multiplier of N.

As k 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 $\frac{1}{16}^{th}$ of a column will remain in the cache, whereas it will result in a multiplier of $\frac{N}{BLK}$ for a fully-associative cache.

Hence, a total of $N^2 = 2^{20}$ cache misses for a direct-mapped cache and $\frac{N^2}{BLK} = 2^{17}$ cache misses for a fully-associative cache.

For B:
As j is varied, we take the advantage of spatial locality as we are accessing a row of it and we get $\frac{N}{BLK}$ misses.
As i is varied, we will access the same row which is already cached, hence a multiplier of 1.
As k is varied, different columns are accessed in the loop k, hence a multiplier of N.
Hence, a total of $\frac{N^3}{BLK} = 2^{27}$ cache misses for both direct mapped and fully-associative cache.

For C:
As j is varied, we take the advantage of spatial locality as we are accessing a row of it and we get $\frac{N}{BLK}$ misses.
As i is varied, different columns are accessed in the loop i, hence a multiplier of N.
As k is varied, we will have same number of misses, hence a multiplier of N.
Hence, a total of $\frac{N^3}{BLK} = 2^{27}$ cache misses for both direct mapped and fully-associative cache.

Table for direct-mapping cache misses

Loop	A	B	C
j	1	$\frac{N}{BLK}$	$\frac{N}{BLK}$
i	N	1	N
k	N	N	N
Total	N^2 2^{20}	$\frac{N^2}{BLK}$ 2^{17}	$\frac{N^3}{BLK}$ 2^{27}

Table for fully-associative cache misses

Loop	A	B	C
j	1	$\frac{N}{BLK}$	$\frac{N}{BLK}$
i	N	1	N
k	$\frac{N}{BLK}$	N	N
Total	N^2 2^{17}	$\frac{N^2}{BLK}$ 2^{17}	$\frac{N^3}{BLK}$ 2^{27}

jik form Analysis

For A:
As k is varied, a row of A is accessed, therefore one miss per block, hence a multiplier of $\frac{N}{BLK}$.
As i is varied, we are not able to use temporal locality as previous blocks are replaced, 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 will be the same for direct-mapped and fully-associative caches. Hence, a total of $\frac{N^3}{BLK} = 2^{27}$ cache misses in both direct-mapped and fully-associative caches.

For B:
As k is varied, different columns of B are accessed, hence a multiplier of N. This analysis will 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 $\frac{1}{16}^{th}$ of the column will be in the cache as every $\frac{1}{16}^{th}$ 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 accommodate 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 replaced 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 $\frac{1}{16}^{th}$ access will be a miss, hence a multiplier of $\frac{N}{BLK}$.
Hence, a total of $N^3 = 2^{30}$ cache misses for a direct-mapped cache and $\frac{N^2}{BLK} = 2^{17}$ 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.
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 $\frac{1}{16}^{th}$ of a column will remain in the cache, whereas it will result in a multiplier of $\frac{N}{BLK}$ for a fully-associative cache.
Hence, a total of $N^2 = 2^{20}$ cache misses for a direct-mapped cache and $\frac{N^2}{BLK} = 2^{17}$ cache misses for a fully-associative cache.

Table for direct-mapping cache misses

Loop	A	B	C
k	$\frac{N}{BLK}$	N	1
i	N	N	N
j	N	N	N
Total	$\frac{N^3}{BLK}$ 2^{27}	N^3 2^{30}	N^2 2^{20}

Table for fully-associative cache misses

Loop	A	B	C
k	$\frac{N}{BLK}$	N	1
i	N	1	N
j	N	$\frac{N}{BLK}$	$\frac{N}{BLK}$
Total	$\frac{N^3}{BLK}$ 2^{27}	$\frac{N^2}{BLK}$ 2^{17}	$\frac{N^2}{BLK}$ 2^{17}

Problem.3

Consider the following code.

```
#define N (4096)
double y[N], X[N][N], A[N][N];
for (k = 0; k < N; k++)
  for (j = 0; j < N; j++)
    for (i = 0; i < N; i++)
      y[i] = y[i] + A[i][j] * X[k][j];
```

Assume a direct-mapped cache of capacity 16 MB, with 32 B cache lines and a word of 8 B. Assume that there is negligible interference between the arrays `A`, `X`, and `y` (i.e., each array has its 16 MB cache for this question), and arrays are laid out in the row-major form.

Estimate the total number of cache misses for `A`, `X`, and `y`.

Solution

Given

word size = 8 B

Block size = 32 B = 4 words

cache size = 16MB = $2^{24}B = 2^{21}$ words

Array Size = 2^{24} words

So only $\frac{1}{8}^{th}$ of the array will fit into the cache, which will lead to conflict misses in case of direct mapping, due to which the entire blocks that will be brought in inside the cache will be replaced.

For A:

For the innermost loop i we are accessing via column and as a result all N will result in miss, and moreover since it is a direct mapping so, the blocks which are previously fetched in will be replaced. So even for the outer two loops of j and k, the same misses will continue for $N * N$ times, so total number of misses for A is $N * N * N = N^3 = 2^{36}$.

For X:

The innermost loop i has no importance and doesn't lead to any hit or misses. Our main concern is the j^{th} and k^{th} loop, see that as the j^{th} loop changes we are missing the first word of that block and then fetching the entire block, resulting in $\frac{N}{BLK}$ misses. For the k^{th} loop since the whole array is not cached we will have a multiplier of N, so total number of misses for X is $\frac{N^2}{BLK} = 2^{22}$.

For y:

Since the whole y can fit in the cache, we will have only cold miss resulting in $\frac{N}{BLK} = 2^{10}$.

Loop	A	X	y
i	N	1	$\frac{N}{BLK}$
j	N	$\frac{N}{BLK}$	1
k	N	N	1
Total	N^3 2^{36}	$\frac{N^2}{BLK}$ 2^{22}	$\frac{N}{BLK}$ 2^{10}

Problem.4

Assume a naïve O(n^3) sequential matrix multiplication kernel (ijk form) for matrices of dimensions 4096×4096 (given).

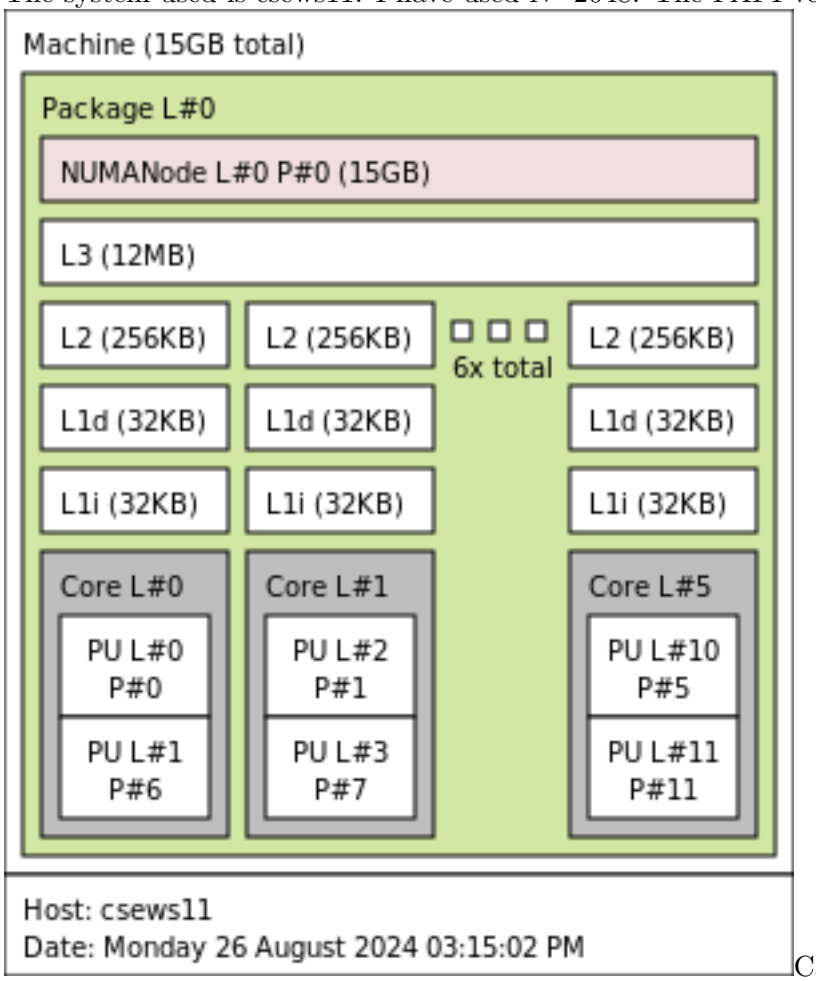
(i) Implement an optimized version of the sequential matrix multiplication implementation using loop blocking. Experiment with different block dimensions (for e.g., 4×4 to 32×32).

Use a table to report the speedup of your optimized implementation over the sequential implementation for the different block sizes that you have tried. The first three columns will be the block sizes for matrices A, B, and C, and the last column will report the speedup compared to the sequential version.

(ii) Note the block size that works best for your setup. Remember that the cache hierarchy will influence the optimal block dimensions. Justify the improved performance by tracking performance counters with PAPI. List the PAPI version and the performance counters you have used.

Solution

The system used is csews11. I have used N=2048. The PAPI version used is **7.1.0.0**.



Cache Hierarchy

Below is the speedup recorded:-

A	B	C	fac
4*4	4*4	4*4	2.92
4*8	8*4	4*4	3.52
4*16	16*4	4*4	4.05
4*32	32*4	4*4	3.32
4*64	64*4	4*4	2.58
4*4	4*8	4*8	2.70
4*8	8*8	4*8	4.15
4*16	16*8	4*8	4.41
4*32	32*8	4*8	3.57
4*64	64*8	4*8	2.91
4*4	4*16	4*16	3.25
4*8	8*16	4*16	3.63
4*16	16*16	4*16	4.28
4*32	32*16	4*16	3.77
4*64	64*16	4*16	2.81
4*4	4*32	4*32	3.35
4*8	8*32	4*32	4.79
4*16	16*32	4*32	5.03
4*32	32*32	4*32	3.54
4*64	64*32	4*32	2.88
4*4	4*64	4*64	3.93
4*8	8*64	4*64	5.15
4*16	16*64	4*64	5.19
4*32	32*64	4*64	3.97
4*64	64*64	4*64	2.99
8*4	4*4	8*4	2.90
8*8	8*4	8*4	3.95
8*16	16*4	8*4	4.40
8*32	32*4	8*4	3.57
8*64	64*4	8*4	2.86
8*4	4*8	8*8	3.42
8*8	8*8	8*8	4.36
8*16	16*8	8*8	4.87
8*32	32*8	8*8	3.91
8*64	64*8	8*8	2.96
8*4	4*16	8*16	3.94
8*8	8*16	8*16	4.74
8*16	16*16	8*16	5.13
8*32	32*16	8*16	3.89
8*64	64*16	8*16	3.05
8*4	4*32	8*32	3.79
8*8	8*32	8*32	5.04
8*16	16*32	8*32	5.53
8*32	32*32	8*32	4.17
8*64	64*32	8*32	3.05

A	B	C	fac
8*4	4*64	8*64	4.23
8*8	8*64	8*64	5.50
8*16	16*64	8*64	5.82
8*32	32*64	8*64	4.16
8*64	64*64	8*64	3.12
16*4	4*4	16*4	3.57
16*8	8*4	16*4	4.45
16*16	16*4	16*4	5.05
16*32	32*4	16*4	3.70
16*64	64*4	16*4	2.91
16*4	4*8	16*8	3.84
16*8	8*8	16*8	4.91
16*16	16*8	16*8	5.02
16*32	32*8	16*8	3.85
16*64	64*8	16*8	2.98
16*4	4*16	16*16	3.92
16*8	8*16	16*16	4.98
16*16	16*16	16*16	5.52
16*32	32*16	16*16	4.16
16*64	64*16	16*16	3.09
16*4	4*32	16*32	4.08
16*8	8*32	16*32	5.24
16*16	16*32	16*32	5.55
16*32	32*32	16*32	4.11
16*64	64*32	16*32	3.05
16*4	4*64	16*64	3.98
16*8	8*64	16*64	5.37
16*16	16*64	16*64	5.41
16*32	32*64	16*64	4.05
16*64	64*64	16*64	2.89
32*4	4*4	32*4	3.42
32*8	8*4	32*4	4.31
32*16	16*4	32*4	4.54
32*32	32*4	32*4	3.55
32*64	64*4	32*4	2.82
32*4	4*8	32*8	3.61
32*8	8*8	32*8	4.64
32*16	16*8	32*8	4.63
32*32	32*8	32*8	3.73
32*64	64*8	32*8	2.9
32*4	4*16	32*16	3.81
32*8	8*16	32*16	4.60
32*16	16*16	32*16	4.97
32*32	32*16	32*16	3.77
32*64	64*16	32*16	2.90

A	B	C	fac
32*4	4*32	32*32	3.85
32*8	8*32	32*32	4.95
32*16	16*32	32*32	5.18
32*32	32*32	32*32	3.89
32*64	64*32	32*32	2.95
32*4	4*64	32*64	4.04
32*8	8*64	32*64	5.06
32*16	16*64	32*64	5.42
32*32	32*64	32*64	3.96
32*64	64*64	32*64	2.97
64*4	4*4	64*4	3.37
64*8	8*4	64*4	4.20
64*16	16*4	64*4	4.62
64*32	32*4	64*4	3.68
64*64	64*4	64*4	2.93
64*4	4*8	64*8	3.76
64*8	8*8	64*8	4.68
64*16	16*8	64*8	4.69
64*32	32*8	64*8	3.79
64*64	64*8	64*8	2.95
64*4	4*16	64*16	4.10
64*8	8*16	64*16	5.13
64*16	16*16	64*16	5.70
64*32	32*16	64*16	4.35
64*64	64*16	64*16	3.15
64*4	4*32	64*32	4.18
64*8	8*32	64*32	5.39
64*16	16*32	64*32	5.89
64*32	32*32	64*32	4.40
64*64	64*32	64*32	3.22
64*4	4*64	64*64	4.55
64*8	8*64	64*64	5.67
64*16	**16*64**	**64*64**	**6.04**
64*32	32*64	64*64	4.48
64*64	64*64	64*64	3.23

The block size that works the best for the setup: **64*16 for A, 16*64 for B and 64*64 for C**.

Part(ii)

Performance counters used were $PAPI\_L1\_DCM$, $PAPI\_L2\_DCM$, $PAPI\_L1\_TCM$ and $PAPI\_L2\_TCM$.

Below are result for non-blocking and best variant of the blocks:-

Type	$L1\_DCM$	$L2\_DCM$	$L1\_TCM$	$L2\_TCM$
Blocking	5429803300	771115779	5429814385	771126707
Avg-Blocking	5061077267	4179807802	5061135935	4179849446
Non-Blocking	11228750067	13421454139	11228779312	13421477449

By the above data we can note that cache miss of our best blocking is **2.07** times, **17.40** times, **2.07** times and **17.40** times less than non-blocking justifying 6 times speedup.
We can also note that majority of speedup is due to difference in L2 misses.