CS 610 Semester 2025–2026-I: Assignment 1

August 23, 2025

Problem 1: Solution

Given Parameters

- Number of Iterations, N = 1000.
- SIZE = $(1 \ll 15)$ = 32K elements.
- Float/Element size = 4 B ⇒ Array A & B size = $2^{15} \times 2^2 = 2^{17}$ B = 128 KB
- Cache size = 2^{17} B = 128 KB, line size = 2^7 B = 128 B, associativity = 8 (8-way).
- Floats/Elements per line = 128/4 = 32 Floats/Elements per cache line.
- Number of sets:
$$= \frac{2^{17}}{2^7 \times 8} = 2^7 = 128 \text{ sets}$$
- Each array has 1024 lines over 128 sets ⇒ 8 lines per set (for A, and similarly for B).
- Base addresses (block aligned): A = 0x12345678, B = 0xabcd5678.

Array Address Mapping

- **Offset Bits**: 7 bits.
- **Index Bits**: 7 bits.
- **Tag Bits**: 32 - (7 + 7) = 18 bits.

The lower 16 bits of the cache are identical (0x5678). The cache index is determined by bits from 7th to 13th of an address. Since these bits are the same for both arrays A and B, they map to the same set in the cache.

Stride-wise analysis and results

1. Stride = 1

- **Iteration it = 0**: An access to A[0] causes a cold miss. The cache line containing A[0] to A[31] is loaded into the corresponding cache set. The subsequent 31 access from A[1] to A[31] are all hits.
- A cold miss occurs for A every 32 elements. Therefore, number of misses for A in the first iteration = $\frac{\text{SIZE}}{\text{floats-per-line}} = \frac{2^{15}}{2^5} = 1024$ misses.

- **Iterations it $\geq$ 1**: The total working set size is the size of A + B = 128KB + 128KB = 256KB. Since the cache is only 128 KB, it cannot hold both arrays. So, by the time the first iteration of the inner loop finishes, all the earlier blocks of data will be evicted. Therefore, for every subsequent outer loop iteration, the same number of misses will occur as in the first iteration, leading to capacity misses.

- **Total Misses =** 1024 * 1000 = 1,024,000.

2. Stride = 16

- Accessing A[0] causes a miss and loads the line with A[0] to A[31]. The next access, A[16], is a hit. A miss occurs every 32 elements. Therefore, the number of misses per inner loop iteration = $\frac{\text{SIZE}}{\text{floats-per-line}} = 1024$.

- Similar to stride = 1, the working set is 256 KB, which is larger than the cache size (128 KB), causing capacity misses on every subsequent outer loop iteration.

- **Total Misses =** 1024 * 1000 = 1,024,000.

3. Stride = 32

- Every access to A (A[0], A[32], A[64] etc.) is to a new cache line. Number of accesses to A in the inner loop = $\frac{\text{SIZE}}{32} = 1024$, and each of these 1024 accesses results in a cold miss in the first iteration. Subsequent iterations, will lead to capacity misses.

- **Total Misses =** 1024 * 1000 = 1,024,000.

4. Stride = 64

- Number of accesses to A per inner loop = $\frac{\text{SIZE}}{64} = 512$ distinct A lines.

- The stride pattern causes only half the cache sets to be used, halving the usable cache size to 64 KB. The working set is 128 KB.

- Since Working Set (128 KB) > Usable Cache (64 KB), conflict misses occur in every iteration.

- **Total Misses =** 512 * 1000 = 512,000.

5. Stride = 2K

- Number of accesses to A per inner loop = $\frac{\text{SIZE}}{2^{11}} = 16$ distinct A lines.

- The stride pattern uses only two sets of the cache. Usable cache size = 2 KB. Working set size (for A and B) = 4KB.

- Since Working set (4KB) > Usable Cache (2KB), the 16 accesses to A will result in conflict misses on each iteration.

- **Total Misses =** 16 * 1000 = 16,000.

6. Stride = 8K

- Number of accesses to A per inner loop = $\frac{\text{SIZE}}{2^{13}} = 4$ distinct A lines.
- All accesses map to a single set. Usable cache size = 1 KB. The working set is 8 blocks (4 from A, 4 from B), totaling 1 KB.
- Since Working Set (1 KB) == Usable Cache (1 KB), the data fits perfectly. Misses only occur on the first iteration.
- **Total Misses =** 4.

Table 1: Total cache misses on `A` (over all $N = 1000$ iterations)

Stride	Total misses on A
1	1,024,000
16	1,024,000
32	1,024,000
64	512,000
2K	16,000
8K	4

Problem 2: Solution

Given Parameters

- **Matrix Dimensions (N):** $1024 = 2^{10}$.
- **Total Elements in the Array:** $1024 * 1024 = 2^{20}$
- **Cache Size:** 64K words = 2^{16} words.
- **Line Size:** $2^4 = 16$ words.
- **Number of Cache Lines:** $\frac{\text{Cache Size}}{\text{Line Size}} = \frac{2^{16}}{16} = 2^{12}$ lines.
- **Storage:** Row-major.

1. Analysis of `kij` Form

1.1. Direct Mapped Cache

- **Array A[i][k]:** Access to array A is independent of j. For each i, since the array is accessed column-wise, there will be N cache misses. Because the size of A exceeds the cache capacity, when k changes, previously loaded blocks will have been evicted. As a result, this pattern repeats for each k, leading to a total of $\mathbf{N^2}$ misses.
- **Array B[k][j]:** In the innermost loop over j, B is accessed row-wise for a fixed k, resulting in a miss for each cache line, i.e., $N/16$ misses. When i changes, since k is fixed and the entire k-th row is already loaded into the cache, all accesses will be hits. When k changes, there will be a cold miss for the new row, 1 miss for each new row, a total of N misses for k. Therefore, the total number of misses is $N^2/16$.
- **Array C[i][j]:** In the innermost loop over j, C is accessed row-wise, resulting in $N/16$ misses. In the immediate outer loop over i, there will be a miss for each row, giving N misses. In the outermost loop over k, whenever k changes, there will again be N misses. Therefore, the total number of misses is $N^3/16$.

Table 2: `kij` with Direct Mapped Cache

	A	**B**	**C**
k	N	N	N
i	N	1	N
j	1	$\frac{N}{16}$	$\frac{N}{16}$
Total Misses (Expression)	N^2	$\frac{N^2}{16}$	$\frac{N^3}{16}$
Total Misses (Evaluated Value)	2^{20}	2^{16}	2^{26}

1.2. Fully Associative Cache

- **Array A[i][k]:** Access to A is independent of j. For each i, as the array is accessed column-wise, there will be N distinct cache line accesses. In the fully associative cache, these blocks persist because the total fetched size is $N \times (\text{line size}) = 2^{10} \times 2^4 = 2^{14}$ bytes $< 2^{16}$ bytes (cache size). Thus, when $k = 0$, the first block of each row is loaded with a miss, and for $k = 1$ to 15, all accesses hit since the columns remain in cache. Therefore, for each k, there is one miss for every 16 accesses, giving $\frac{N}{16}$ misses per row and a total of $\frac{N^2}{16}$ misses.
- **Array B[k][j]:** In the innermost loop over j, accesses to B are row-wise, giving $\frac{N}{16}$ misses. In the immediate outer loop over i, k is fixed, so all accesses hit after the first. When k changes in the outermost loop, a new row of B is required, resulting in N misses. Thus, the total number of misses is $\frac{N^2}{16}$.
- **Array C[i][j]:** Same miss pattern as in the direct-mapped case for `kij`, resulting in a total of $\frac{N^2}{16}$ misses.

Table 3: `kij` with Fully Associative Cache

	A	**B**	**C**
k	$\frac{N}{16}$	N	N
i	N	1	N
j	1	$\frac{N}{16}$	$\frac{N}{16}$
Total Misses (Expression)	$\frac{N^2}{16}$	$\frac{N^2}{16}$	$\frac{N^3}{16}$
Total Misses (Evaluated Value)	2^{16}	2^{16}	2^{26}

2. Analysis of `jki` Form

2.1. Direct Mapped Cache

- **Array A[i][k]:** In the innermost loop, access is column-wise, giving N misses for i. When k changes, previous blocks are evicted in the direct-mapped cache, adding another N misses. For j, the whole array is accessed, giving N misses. Total misses $= N^3$.
- **Array B[k][j]:** Same access pattern as $A[i][k]$ in the kij form direct mapped cache, so total misses $= N^2$.
- **Array C[i][j]:** For i, there are N misses; for k, previously loaded blocks are evicted causing N misses; for j, another N misses occur. Total misses $= N^3$.

Table 4: `jki` with Direct Mapped Cache

	A	**B**	**C**
j	N	N	N
k	N	N	N
i	N	1	N
Total Misses (Expression)	N^3	N^2	N^3
Total Misses (Evaluated Value)	2^{30}	2^{20}	2^{30}

2.2. Fully Associative Cache

- **Array A[i][k]:** In innermost loop with i, A is accessed column-wise, so N misses. For $k = 0$, the first line of each row is loaded; $k = 1$ to 16 are hits, giving $N/16$ misses. For j, there are N misses. Total misses = $N^3/16$.
- **Array B[k][j]:** Same access pattern as A in kij form for fully associative cache. Total misses = $N^2/16$
- **Array C[i][j]:** In the innermost loop, j is fixed and C is accessed column-wise, causing N misses the first time a column is accessed. Due to spatial locality, each cache line holds 16 consecutive columns for the same row, so the next 15 columns incur hits. Thus, misses occur only when j crosses a multiple of 16. There are $N/16$ such j values, each causing N misses, giving a total misses = $N^2/16$.

Table 5: `jki` with Fully Associative Cache

	A	**B**	**C**
j	N	$\frac{N}{16}$	$\frac{N}{16}$
k	$\frac{N}{16}$	N	1
i	N	1	N
Total Misses (Expression)	$\frac{N^3}{16}$	$\frac{N^2}{16}$	$\frac{N^2}{16}$
Total Misses (Evaluated Value)	2^{26}	2^{16}	2^{16}

Problem 3

3.1. Compilation and Execution

Compilation Command:

```
With Makefile    ->   make problem3
Without Makefile ->   g++ -std=c++17 251110069.cpp -o 251110069.out -pthread
```

Execution Command:

```
The program is executed from the command line with six arguments:
the absolute path to the input file (R),
the number of producer threads (T),
the min lines to read (Lmin),
the max lines to read (Lmax),
the buffer size (M),
the path for the output file (W).

Example -> ./251110069.out /absolute/path/input.txt 4 5 10 20 /absolute/path/output.txt
```

3.2. Synchronization

- **Shared Buffer:**
  - `buffer_mutex`: Ensures that only one thread, either a producer or a consumer thread, can write or read the shared buffer at any given time.
  - `buffer_not_full_condition`: Producers wait on this if the buffer is full. They are awakened by consumers who have removed data from the buffer.
  - `buffer_not_empty_condition`: Consumers wait on this if the buffer is empty. They are awakened by producers who have added data.
- **Input File:**
  - `input_mutex`: Ensures that only one producer can read from the input file at a time. A thread locks it, reads its L lines, and then unlocks it, ensuring atomicity of the read operation.
- **Output File:**
  - `output_mutex`: Ensures that the block of lines retrieved by the consumer from the buffer is written to the output file contiguously without interruption from other consumer threads.
- **Producer Thread L-lines Atomicity:**
  - `producer_thread_turn_mutex`: Ensures that a producer writes its entire block of L lines to the buffer atomically. This prevents other producers from interleaving writes, especially when $L > M$ and the thread must write in multiple chunks.
- **Atomic Variable:**
  - `producers_status`: This ensures that consumer threads only terminate when all producer threads have finished their work (reading all input and writing to the buffer is complete), and the buffer is empty.