CS610 Assignment 1

Khushi Gupta

August 2025

Problem 1

Given

The arrays have SIZE = $1 \ll 15 = 2^{15}$ elements and each float is 4 B, so each array occupies $2^{15} \cdot 4 = 2^{17}$ bytes. The cache is 128 KB = 2^{17} B, with line size $128 = 2^7$ B, 8-way associativity, and LRU replacement. The base addresses are $A =$ `0x12345678` and $B =$ `0xabcd5678`. The program repeats the inner traversal $N = 1000$ times and accesses $A[i]$ then $B[i]$ for the same indices.

A cache line is $2^7 = 128$ bytes, so the low 7 address bits are the *offset* within a line. The cache has $2^{17}/2^7 = 2^{10} = 1024$ lines and $1024/8 = 128 = 2^7$ sets, so the next 7 bits (bits [13:7]) are the *index* bits. The remaining high bits are the tag.

A (`0x12345678`):

$$\text{set} = A \gg 7 \;\&\; (2^7 - 1) = \texttt{0x2468AC} \;\&\; \texttt{0x7F} = 44 = 0x2C.$$

Because the offset is 120, the first cache line contains only the first two floats of A (indices 0 and 1). Subsequent lines are full (32 floats per full line).

Each array has 2^{15} floats $\times$ 4 B = 2^{17} B (128 KB). Since A starts at offset 120 B, it spans 1025 cache lines while the cache holds only 1024. B is the same, so together A and B exceed the cache capacity.

Stride = 1

The first line is partial (indices 0–1), and thereafter every time we cross a 32-float line boundary we fetch a new block. The misses look like 0, 2, 34, 66, 98, ... Block j of A maps to set $(44+j)$ mod 128, while the corresponding block of B also maps to the same set, so A and B contend directly. Each traversal touches all 1025 A blocks, and since A + B together exceed cache capacity, the misses repeat every traversal.

Total misses on A = $1025 \times 1000 = 1{,}025{,}000$

Stride = 16

Here, indices are 0, 16, 32, 48, ... Since stride 16 still ensures that each 32-float line is touched, every line of A is brought in once per traversal. The misses again look like 0, 16, 32, 48, ... As with stride 1, every traversal incurs misses for all 1025 A blocks, and conflicts with B cause capacity misses each time.

Total misses on A = $1025 \times 1000 = 1{,}025{,}000$

Stride = 32

Now the access skips within a line, so only the first element of each line is used. Thus exactly one miss per line occurs, covering all of A's 1024 full lines plus the partial first line. The misses look like 0, 32, 64, 96, ... Every traversal reloads all 1025 A blocks.

$$\text{Total misses on A} = 1024 \times 1000 = 1{,}024{,}000$$

Stride = 64

Each access jumps two lines at a time, so only every other line of A is touched. Thus per traversal, 512 A blocks are loaded. The misses look like 0, 64, 128, 192, ... Since A and B together still exceed cache, all of these 512 blocks miss in every traversal.

$$\text{Total misses on A} = 512 \times 1000 = 512{,}000$$

Stride = 2K

With stride $2^{11} = 2048$, each access jumps 64 lines at a time. Thus only 16 distinct A blocks are touched in a traversal. The misses look like 0, 2048, 4096, ... Out of these 8 blocks repeatedly map with B to the same sets, so every traversal reloads them.

$$\text{Total misses on A} = 16 \times 1000 = 16{,}000$$

Stride = 8K

With stride $2^{13} = 8192$, each access jumps 256 lines. Thus only 4 A blocks are touched in the whole traversal. The misses look like 0, 8192, 16384, ... Once loaded, they are reused across traversals without eviction.

$$\text{Total misses on A} = 4$$

Conclusion

Stride	Misses on A
1	1,025,000
16	1,025,000
32	1,024,000
64	512,000
2K	16,000
8K	4

Problem 2

1) `kij` — Direct mapped

	A	B	C
i	N	1	N
j	1	$\frac{N}{16}$	$\frac{N}{16}$
k	N	N	N
Total	N^2	$\frac{N^2}{16}$	$\frac{N^3}{16}$

Explanation

- A: Access pattern is $A[i][k]$ with i inner, columnwise for fixed k. In direct-mapped cache the 1024 different lines touched by the i loop do not all fit without conflicts, so the i loop gives N misses. This repeats for each k, hence the k factor N. j does not multiply misses (factor 1).
- B: For fixed k we need row $B[k][0..N-1]$. Each row costs $N/16$ line loads (the j-factor). This row remains same for each i, hence the i loop does not add misses (factor 1). New row is loaded for each k (the k factor $= N$).
- C: Inner j scans row $C[i][0..N-1]$ touching $N/16$ lines per inner loop (the j-factor); this occurs for each i and for each k, hence $i = N$ and $k = N$.

2) `kij` — Fully associative

	A	B	C
i	N	1	N
j	1	$\frac{N}{16}$	$\frac{N}{16}$
k	$\frac{N}{16}$	N	N
Total	$\frac{N^2}{16}$	$\frac{N^2}{16}$	$\frac{N^3}{16}$

Explanation:

- **A:** For fixed k we scan down column $A[i][k]$; with full associativity a column's working set (N lines) fits in cache for reuse across BL k iterations, so $i = N$, $j = 1$, $k = \frac{N}{16}$.
- **B:** For fixed k we stream the row $B[k][0..N-1]$, costing $N/16$ line loads per k (the j-factor); this repeats for all k (the k-factor N), while i contributes no extra misses (1).
- **C:** Inner j scans a row of $C[i][*]$ costing $N/16$ lines per loop; this occurs for each i and each k, hence $j = \frac{N}{16}$, $i = N$, $k = N$.

3) `jki` — Direct mapped

	A	B	C
i	N	1	N
j	N	N	N
k	N	N	N
Total	N^3	N^2	N^3

Explanation:

- **A:** Inner-i walks down column $A[i][k]$. In a direct-mapped cache the column touches N distinct lines per inner loop and this repeats over k and j, so factors $i = N,\ k = N,\ j = N$.
- **B:** For fixed j the middle loop runs k and accesses $B[k][j]$ (column), direct-mapped conflicts make the column-access miss each time across k and j, giving $k = N,\ j = N,\ i = 1$.
- **C:** Inner-i updates $C[i][j]$ down a column for fixed j, touching N lines per i loop, direct-mapped conflicts make the column-access miss each time across k and j, giving $k = N,\ j = N,\ i = N$.

4) `jki` — Fully associative

	A	B	C
i	N	1	N
j	N	$\frac{N}{16}$	$\frac{N}{16}$
k	$\frac{N}{16}$	N	1
Total	$\frac{N^3}{16}$	$\frac{N^2}{16}$	$\frac{N^2}{16}$

Explanation:

- **A:** Inner-i walks down column $A[i][k]$ so $i = N$, with full associativity a column can be resued across BL iteration of k, giving $k = N/16$; the outer loop repeats this for each j, hence $j = N$.
- **B:** k loop walks down column $B[k][j]$ so $k = N$, with full associativity a column can be resued across BL iteration of j, giving $j = N/16$, i loop does not add anything so $i = 1$.
- **C:** Inner i updates column $C[i][j]$ so $i = N$, full associativity prevents repeated evictions across k (factor $k = 1$), and a column can be resued across BL iteration of j giving $j = N/16$.

Problem 3

The program implements a multi-producer / multi-consumer pipeline that reads an input file R and writes to an output file W using a bounded FIFO buffer. Command-line arguments are: $R,\ T,\ L_{\min},\ L_{\max},\ M,\ W$. There are T producer threads and $C = \max(1, \lfloor T/2 \rfloor)$ consumer threads.

Producer behavior. Each producer repeatedly samples L uniformly at random with $L_{\min} \leq L \leq L_{\max}$, atomically reads up to L consecutive lines from the shared input stream (serialized by

`file_mutex`), and writes that entire L-line block atomically into the shared FIFO buffer. Producers block when the buffer has insufficient free capacity and notify consumers after inserting lines. When a producer finds no more lines it decrements `active_producers`. The last producer sets `producers_done=true` and notifies waiting consumers.

Consumer behavior. Each consumer waits for buffer non-emptiness, then atomically removes the currently available lines from the buffer and writes them to the output file W. If the buffer is empty and `producers_done` is set, the consumer exits.

Synchronization primitives.

- `file_mutex`: serializes reads from the input file so each producer obtains distinct consecutive lines.
- `producer_mutex`: ensures a producer's L-line insertion into the FIFO is atomic w.r.t. other producers.
- `buffer_mutex` + `not_full_cv`/`not_empty_cv`: implement the bounded FIFO semantics (producers wait when full; consumers wait when empty).