Assignment 1

Name: Rohit Raj
Roll No.: 210874

Problem 1

As per the question, cache is 128 KB, with line size 128 B, and word size 4 B (float). Hence, the cache can store $\frac{128\times1024}{4} = 32K$ words. Each cache line can store $\frac{128}{4} = 32$ words. The array size is also $32K$ words (= 128 KB), so array A and B together compete for cache space. We analyze the cache misses for array A for different access strides.

1. Stride = 1

We are accessing consecutive elements of A. Each cache line contains 32 elements, so after one miss, the next 31 accesses hit. Thus, in one inner pass through A, every 32nd access is a miss. Since array A has 32K elements, the number of cache lines accessed is 1025 (due to base misalignment). Therefore, the total number of compulsory misses is 1025.

But, due to A and B together exceeding the cache capacity (256 KB total vs 128 KB cache), each new iteration of the outer loop incurs capacity misses again. Hence the total misses on A are:

$$\#\text{miss}_{\text{stride=1}} = 1025 \times 1000 = 1{,}025{,}000$$

2. Stride = 16

Now every 16th element of A is accessed. Still, each line of 32 elements is eventually touched (since stride divides the line size). So again 1025 distinct lines are touched in one pass. The same capacity effect occurs as in stride 1.

$$\#\text{miss}_{\text{stride=16}} = 1{,}025{,}000$$

3. Stride = 32

Now, every access is to the first element of a cache line (one per line). So only one word from each line is used, but all lines (1024, since alignment works out) are touched. Thus, the misses are:

$$\#\text{miss}_{\text{stride=32}} = 1024 \times 1000 = 1{,}024{,}000$$

4. Stride = 64

Here, only half the lines of A are accessed (every 64th element skips one line). Thus, 512 distinct lines are touched per pass. In a fully associative cache these would be only compulsory misses. But due to limited 8-way associativity and exact overlap with array B, severe conflict misses occur. Hence:

$$\#\text{miss}_{\text{stride=64}} = 512 \times 1000 = 512{,}000$$

5. Stride = 2K

Here, only $\frac{32K}{2K} = 16$ elements of A are accessed, each from a distinct cache line. So 16 distinct lines are touched per pass. Again, due to B mapping to the same sets, conflict misses cause repeated evictions. Thus:

$$\#\text{miss}_{\text{stride=2K}} = 16 \times 1000 = 16{,}000$$

6. Stride = 8K

Now, only $\frac{32K}{8K} = 4$ elements of A are accessed, corresponding to 4 cache lines. These lines remain in cache across iterations. So only 4 compulsory misses occur in total:

$$\#\text{miss}_{\text{stride=8K}} = 4$$

Final Table of Results

Stride	Distinct Lines of A	#Misses (Total)
1	1025	1,025,000
16	1025	1,025,000
32	1024	1,024,000
64	512	512,000
$2K$	16	16,000
$8K$	4	4

Problem 2

In this problem we have 64K words cache and each cache line is 16(BL = 16). Also, each matrix has 1024K words meaning it can't be stored in cache all at once. For further analysis $N = 1024$

2.1 kij form

Table 1: Fully Associative Cache

	A	**B**	**C**
k	N	N	N
i	1	N	1
j	1	$\frac{N}{16}$	$\frac{N}{16}$

Analysis

Table 2: Directly Mapped Cache

	A	**B**	**C**
k	N	N	N
i	1	N	1
j	1	$\frac{N}{16}$	$\frac{N}{16}$

Analysis

label=: In the inner jth loop, there will miss only once since after that we are accessing the same value for whole loop. Now, for middle ith loop we accessing values in a column major form hence every access will be a miss giving a miss factor of 10. Now for outer kth loop, A[i][0] for all i will be already in the cache, hence next 7 values will be hit (row-major access). This means this will result in a factor of $\frac{N}{16}$.

lbbel=: Similar to **A**, the inner jth loop is a simple row-major access resulting in a factor of $\frac{N}{16}$. The middle ith loop results in same row being again and again so factor will 1. The outer kth loop results in accessing new row each time with total of N rows, the factor will be N.

lcbel=: Similar to **A** & **B**, the inner jth loop has factor of $\frac{N}{16}$. The middle ith loop being column major access, the factor is N. The outer kth loop runs all of these N times so factor is also N.

label=: In the inner jth loop and the middle ith loop is very similar to that of **fully associative cache**, the difference is that all the A[i][0]s will not be retained this time due to conflict misses. Hence, the outer kth loop will result in a factor of $\frac{N}{16}$.

lcbel=: All the loops is similar to that of **fully associative cache** hence we will have similar factors for this as well.

Similar to **B**, all the loops is similar to that of **fully associative cache** hence we will have similar factors for this as well.

2.2 jki form

Table 3: Fully Associative Cache

	A	**B**	**C**
j	1	N	N
k	N	N	1
i	N	1	$\frac{N}{16}$

Analysis

Table 4: Directly Mapped Cache

	A	**B**	**C**
j	1	N	N
k	N	N	N
i	N	1	$\frac{N}{16}$

Analysis

label=: The access to A, ‘A[i][k]‘, is independent of the outer ‘j‘ loop, resulting in a factor of 1. The middle ‘k‘ loop and inner ‘i‘ loop both traverse the matrix in a column-major fashion against its row-major storage, resulting in poor spatial locality and a miss factor of N for both loops.

lbbel=: The outer ‘j‘ loop selects a new column of B, and the middle ‘k‘ loop iterates through its rows; both are inefficient and have a factor of N. The inner ‘i‘ loop reuses the exact same element ‘B[k][j]‘ N times, giving it a factor of 1.

lcbel=: The outer ‘j‘ loop processes a new column of C each time (factor N). The entire column is accessed in the inner ‘i‘ loop (factor $\frac{N}{16}$). Crucially, this column is small enough to stay in the cache and be reused for all iterations of the middle ‘k‘ loop, giving it a factor of 1.

label=: The analysis for **A** is very similar to that of the **fully associative cache**. The column-major access pattern already results in maximum misses, so conflict misses do not significantly worsen the performance.

lbbel=: The analysis for **B** is also similar to the **fully associative cache**. The access patterns in the outer loops are already inefficient, and the reuse in the inner loop is unaffected by conflicts.

lcbel=: The key difference from the fully associative case is for **C**. The temporal reuse of the column ‘C[:][j]‘ across the ‘k‘ loop is destroyed. The scattered memory accesses to matrix **A** during the ‘k‘ loop thrash the cache, evicting the lines holding **C**. Therefore, the ‘i‘ loop causes $\frac{N}{16}$ misses on *every* iteration of the ‘k‘ loop, changing its factor from 1 to N.

Problem 3

Compilation and Execution

The program can be compiled using the provided Makefile with the following command:

```
make problem3
```

An example invocation of the program is:

```
./problem3 <R> <T> <Lmin> <Lmax> <M> <W>
```

where

- `R`: Absolute path of the input file.
- `T`: Number of producer threads.
- `Lmin`, `Lmax`: Minimum and maximum number of lines read by a thread.
- `M`: Size of the shared buffer in lines.
- `W`: Path of the output file.

Synchronization Approach

We use the classic **producer-consumer model**:

- Producers read L consecutive lines from the input file, where L is chosen randomly between $L_{\min}$ and $L_{\max}$. These lines are inserted atomically into the FIFO shared buffer.
- Consumers dequeue data from the buffer and write atomically to the output file.
- A mutex lock protects shared resources (input file, buffer, and output file).
- `std::condition_variable` is used to synchronize access between producers and consumers, avoiding busy-waiting.