CS 610 Semester 2025–2026-I: Assignment 1

11th August 2025

Due Your assignment is due by Aug 20, 2025, 11:59 PM IST.

General Policies

- You should do this assignment ALONE.
- Do not copy or turn in solutions from other sources. You will be PENALIZED if caught.

Submission

- Submission will be through Canvas.
- Submit a compressed file called "⟨`roll`⟩`-assign1.tar.gz`". The compressed file should have the following structure.

```
-- roll-assign1
-- -- roll-assign1.pdf
-- -- Makefile
-- -- <problem3-dir>
-- -- -- <roll.cpp>
-- -- -- <roll.h>
```

  The above outline assumes there are two programming problems. Name the `.cpp` and `.h` source files using your roll number.

  Use the `Makefile` for compilation. An example compilation command can be `make problem1`.

  The PDF file should contain solutions for the first three problems, and your description and results for the fourth problem.

- We encourage you to use the LaTeX typesetting system for generating the PDF file. You can use tools like Tikz, Inkscape, or Drawio for drawing figures if required. You can alternatively upload a scanned copy of a handwritten solution, but MAKE SURE the submission is legible.
- You will get up to TWO LATE days to submit your assignment, with a 25% penalty for each day.

Evaluation

- Write your programs such that the EXACT output format (if any) is respected.
- We will evaluate your implementations on recent Debian-based distributions, for example, KD first floor lab.
- We will evaluate the implementations with our inputs and test cases, so remember to test thoroughly.

Problem 1 **[20 marks]**

Consider the following loop.

```
#define N (1000)
#define SIZE (1<<15) // 32K
float s = 0.0f, A[SIZE], B[SIZE];
int i, it, stride;
for (it = 0; it < N; it++) {
    for (i = 0; i < SIZE; i += stride) {
        s += (A[i]+B[i]);
    }
}
```

Assume an 8-way set-associative 128 KB cache, line size of 128 B, and word size of 4 B (for `float`). The cache is empty before execution and uses an LRU replacement policy. Determine the total number of cache misses on `A` for the following access strides: 1, 16, 32, 64, 2K, and 8K. Consider all the three kinds of misses, and arrays `A` and `B` compete with each other for cache space. The addresses of `A` and `B` are `0x12345678` and `0xabcd5678`.

Problem 2 **[30 marks]**

Consider a cache of size 64K words and lines of size 16 words. The matrix dimensions are 1024×1024. Perform cache miss analysis for the `kij` and the `jki` 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++)
    r = A[i][k];
    for (j = 0; j < N; j++)
      C[i][j] += r * B[k][j];
```

Listing 2: `jki` form

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

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. Briefly explain the result for each array for each loop nest (e.g., array `C` for variant `jki`).

Problem 3 **[50 marks]**

Write a C++ program that takes the following arguments from the command line: the absolute path to an input file (`R`), the number of producer threads (`T`), the minimum number of lines each thread can read from the file ($\texttt{L}_{\texttt{min}}$), the maximum number of lines each thread can read from the file ($\texttt{L}_{\texttt{max}}$), the size of an intermediate shared buffer in lines (`M`), and the path to an output file (`W`).

Assume `R` contains `N` lines. The whole input file has to be read and can have more than $\texttt{T} \times \texttt{L}_{\texttt{max}}$ lines. AFTER ALL the `T` threads have been created, the threads will CONTEND for access to `R` to

repeatedly read `L` consecutive lines, where `L` is random and $\texttt{L}_{\texttt{min}} \leq \texttt{L} \leq \texttt{L}_{\texttt{max}}$. Then, each thread will write its share of `L` consecutive lines ATOMICALLY to a FIFO SHARED buffer. There will be `max(1,T/2)` consumer threads, where `T/2` is integer division. The consumer threads will KEEP READING from the shared buffer and write their contents atomically to `W`.

- `N` may not be a multiple of `L`. In such cases, the last reader thread will read whatever lines are left (possibly fewer than $\texttt{L}_{\texttt{min}}$).
- Blank lines are also counted in `L`.
- `L` can be greater than `M`. In such cases, the thread which gets access to the shared buffer would repeatedly write `M` lines to complete its `L` writes to the shared buffer before any other thread can start writing.
- The order of the lines in the output file can be jumbled. However, the output should list the `L` lines from each thread one after the other.
- The threads exit when they have no more work to do. There can be reader threads who do not get to read any lines.
- All threads should acquire locks only for the minimum required duration (e.g., producer thread reading `L` lines).
- Use conditional variables instead of busy waiting for synchronizing accesses to shared resources.
- You are not allowed to use concurrent data structures, but you can use other STL data structures like `std::vector` and `std::atomic`.
- The application should terminate properly after all threads are done.

The goal of this problem is to achieve correctness. Test your code thoroughly with different possible inputs.