CS 610 Semester 2023–2024-I: Assignment 2

31st August 2023

Due Your assignment is due by Sep 13, 2023, 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 PDF file with name "<roll-no>.pdf". We encourage you to use the LaTeX typesetting system for generating the PDF file.
- Briefly explain your implementations, results, and other issues of interest (if any). Include the exact compilation instructions for each programming problem.
- Name each source file "<roll-no-probno>.cpp" where "probno" stands for the problem number (e.g., "23111545-prob2.cpp").
- You will get up to TWO LATE days to submit your assignment, with a 25% penalty for each day.

Evaluation

- Your solutions should only use concepts that have been discussed in class.
- Write your code such that the EXACT output format is respected.
- We will evaluate the implementations with our OWN inputs and test cases, so remember to test thoroughly.
- We may deduct marks if you disregard the listed rules.

Problem 1 **[50 marks]**

You are given a set of N producer and M consumer processes. There are N files and the objective is to use the producer-consumer setting to create a histogram of the total count of all unique words across files.

Input The input to your program will be a path to a file, say input.

Producers The file input lists the full paths of N source files that are to be processed by the N producers. Read the contents of the file input into a shared data structure X. Each producer thread will then compete with other producer threads and pick *one* file at random from the shared data structure to analyze. A producer reads its file one line at a time and places the line into a shared queue Y of size 10 elements.

Consumers One consumer thread reads one line from the shared queue and splits the line into words based on whitespace. The words in a line are separated by a single whitespace, ignore punctuation marks such as semicolon or period. The consumer processes update the word counts in another shared data structure `Z` such that when all lines in all the files are processed, `Z` contains each word along with its total frequency across all files.

Output Print the content of `Z` line by line in the format `WORD single_whitespace WORD\_COUNT` in *alphabetical* order of the words.

Example

```
File 1:
ABC, EFG HIJK.
LMNOP QRST.

File 2:
ABC EF HI LMNOPQ
RST UV

Output:
ABC 2
EF 1
EFG 1
HI 1
HIJK 1
LMNOP 1
LMNOPQ 1
QRST 1
RST 1
UV 1
```

Notes You need to correctly coordinate accesses to the three shared data structures `X`, `Y`, and `Z`. You can use any data structure to implement them.

Do not explicitly control the schedule of threads, it is fine to have non-determinism because of synchronization.

Problem 2 **[40 marks]**

Reimplement the previous problem using publicly available libraries that implement concurrent data structures (e.g., hashmap, queues, and stacks). Here are links to a few libraries in no particular order.

- Facebook Folly
- oneAPI Containers
- moodycamel ConcurrentQueue
- Boost Lockfree

You are free to mix multiple libraries.

Problem 3

[5+5+5+5+10 marks]

Consider the following code:

```
int i, j, t, k;
for (t = 0; t < 1024; t++) {
 for (i = t; i < 1024; i++) {
  for (j = t; j < i; j++) {
   for (k = 1; k < j; k++) {
    S(t, i, j, k);
   }
  }
 }
}
```

The data dependences for the loop are given to be (1,0,-1,1), (1,-1,0,1), and (0,1,0,-1).

(a) What are the valid permutations of the loop? Why?

(b) Which loops, if any, are valid to unroll and jam? Why?

(c) What tiling is valid, if any? Why?

(d) Which loops, if any, are parallel?

(e) Show code for the *tikj* form of the code. For this part, ignore the above dependences and assume *tikj* permutation is allowed.

Problem 4

[50+30 marks]

Part (i) Perform loop transformations to improve the performance of the attached C code (`prob4-v0.c`) for sequential execution on one core (i.e., no multithreading). You may use any transformation we have studied, e.g., loop permutation and loop tiling, but no array padding or layout transformation of arrays is allowed.

Explain the reason for the poor performance of the provided reference code, your proposed optimizations (with justifications) to improve performance, and the improvements achieved. Some transformations will lead to speedup, some will not. Include all the transformations that you tried, even if they did not work. Finally, summarize the set of transformations (e.g., xx times unrolling + yy permutation) that gave you the best performance. Report your speedup using the GNU gcc compiler. You can compile the reference code with `gcc -O3 -std=c17 -D_POSIX_C_SOURCE=199309L prob4-v0.c -o prob4-v0.out`.

Use a workstation in the KD lab for your performance evaluation. A good start is to check the processor architecture and cache parameters for your workstation to better understand the impact on performance. Include the system description (e.g., levels of private caches, L1/L2 cache size, processor frequency) in your report. Also include the name of the workstation in your report. The TAs will reuse the same system to reproduce the performance results.

Part (ii) This part is open ended. You are free to apply any valid code optimization trick (e.g., LICM, function inlining, and changing function prototypes) on the version from Part (i) for improved performance.

You should submit two files for this problem: `roll-no-prob4-v1.c` and `roll-no-prob4-v2.c`, corresponding to the two parts. You are allowed to switch to C++ for your solutions.