CS 610 Semester 2024–2025-I: Assignment 3

5th October 2024

Due Your assignment is due by Oct 23, 2024, 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.
- You can tweak the compilation commands in the `Makefile` to suit your compiler version and the target architecture.

Submission

- Submission will be through Canvas.
- Submit a compressed file called "⟨`roll`⟩`-assign3.tar.gz`". The compressed file should have the following structure.

```
-- roll
-- -- roll-assign3.pdf
-- -- <problem1-dir>
-- -- -- <source-files>
-- -- <problem2-dir>
-- -- -- <source-files>
-- -- <problem3-dir>
-- -- -- <source-files>
-- -- <problem4-dir>
-- -- -- <source-files>
```

  The PDF file should contain descriptions for the first two problems, and your solution for the last problem.

- We encourage you to use the LaTeX typesetting system for generating the PDF file. You can use tools like Tikz, Inkscape, or Draw.io 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 a Unix-like system, for example, recent Debian-based distributions installed on KD first floor labs.
- We will evaluate the implementations with our OWN inputs and test cases, so remember to test thoroughly.

Problem 1 **[30+30 marks]**

Implement matrix-matrix multiplication using (i) SSE4 and (ii) AVX2 intrinsics. Implement an aligned variant and an unaligned variant for each SIMD extension (if available).

Ensure correctness of your result and compare performance across versions. You can assume the data type of a matrix element is single-precision floating-point and the dimensions are a multiple of four.

You can optionally implement optimizations like blocking as an additional variant.

Refer to https://www.intel.com/content/www/us/en/docs/intrinsics-guide/index.html# for documentation on Intel Intrinsics.

Problem 2 **[40 marks]**

Implement an *inclusive* prefix sum function using AVX2 instrinsics. Compare the performance with the sequential, OpenMP, and SSE4 versions (discussed in class).

Problem 3 **[40 marks]**

Implement Problem 2 from Assignment 2 using OpenMP (e.g., do not use `pthread_create()`). Use only OpenMP constructs for synchronization between producers and consumers and also for maintaining the shared buffer. You can continue to make use of Pthread APIs (e.g., condvars) ONLY for cases which cannot covered by OpenMP.

I suggest you do the following before attempting to solve the problem: (i) Fix your Pthreads implementation, if required, and (ii) go through the rest of the OpenMP slides for possible ideas.

Problem 4 **[30+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 loop transformation we have studied, e.g., loop permutation and loop tiling, but no array padding or layout transformation of arrays is allowed. You are free to apply any code optimization trick (e.g., LICM, function inlining, and changing function prototypes) on the attached version for improved performance.

Summarize the set of transformations (e.g., xx times unrolling + yy permutation) that gave you the best performance. Explain your optimizations and the improvements achieved. Some transformations you try may not improve the performance. You can still include a description in your report.

Part (ii) Parallelize the attached C code (`problem4-v0.c`) using OpenMP. Compare the performance of the OpenMP program with the sequential version. You may use any combination of valid OpenMP directives and clauses that you think will help.

Compile the reference code with `gcc -O3 -std=c17 -D_POSIX_C_SOURCE=199309L prob4-v0.c -o prob4-v0.out`. You are allowed to switch to C++ for your solutions. Submit two files for this problem: `roll-no-prob4-v1.c[pp]` and `roll-no-prob4-v2.c[pp]`, corresponding to the two parts.

Use a workstation in the KD lab for your performance evaluation, and include the name of the workstation in your report. The TAs will reuse the same system to reproduce the performance results.