CS 610 Semester 2025–2026-I: Assignment 3

13th October 2025

Due Your assignment is due by Oct 24, 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.
- 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>
-- -- -- <rollno-prob1.cpp
-- -- -- <other-source-files>
-- -- <problem2-dir>
-- -- -- <rollno-prob2.cpp
-- -- -- <other-source-files>
-- -- <problem3-dir>
-- -- -- <rollno-prob3.cpp
-- -- -- <other-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 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.

Suggestions

- Refer to Intel Intrinsics Guide for documentation on Intel Intrinsics.
- 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.

Problem 1 **[30 marks]**

Apply valid loop transformations to the stencil computation given in the template file (`problem1.cpp`) to improve its performance. You can apply compound transformations to improve the kernel's memory access pattern and can use OpenMP to exploit parallelism. You should not write explicit SIMD code.

Summarize the set of transformations (e.g., xx times unrolling + yy permutation) that gives you the best performance. Write different kernels to incrementally show the benefit of your transformations. Ensure correctness of your transformed kernels and compare performance across versions.

Problem 2 **[20 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]**

Vectorize the scalar 3D gradient kernel given the template file (`problem3.cpp`) using SSE4 and AVX2 intrinsics. Compare the performance among the three versions, and report the speedups.

Problem 4 **[20+15 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 (e.g., 2D→1D) 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.

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.