CSCI 5451: Final Project

Fall 2025

This document describes the requirements, expectations, and grading scheme for the final project in CSCI 5451 (Parallel Algorithms). The project is worth a total of **24 points**, split across two major components:

- **Section 1: Project Planning Meeting** – 4 points
- **Section 2: Final Project & Report** – 20 points

1 Section 1: Project Planning Meeting (4 points)

Scheduling and Deadline

Each group must schedule a **one-on-one final project planning meeting** with James Mooney using the Google Calendar booking link:

Google Calendar Booking Link

This meeting must take place **on or before Wednesday, November 26, 2025**. Scheduling and attending this meeting is worth **4 points**.

To receive credit:

a) Use the booking link above to schedule a time.

b) Ensure the meeting occurs by the deadline.

c) The corresponding Canvas item for this requirement is available at:

Canvas: Project Planning Meeting Assignment

Meeting Preparation

You **must** come to the meeting with concrete ideas for your final project. At a minimum, you should:

- Identify which of the project categories in Section 2.1 you are most interested in.
- Bring at least one specific problem, dataset, algorithm, or codebase you might work with.
- Be prepared to discuss:
  - Why the problem is interesting or challenging from a *parallel algorithms* perspective.

  - What hardware (CPUs/GPUs, cluster nodes, etc.) and software stack (MPI, OpenMP, CUDA, etc.) you might use.
  - Any initial ideas you already have for parallelization, optimization, or experimentation.

The goal of this meeting is to make sure your project is ambitious, well-scoped, and feasible given the course timeline and available resources. You will be referred back to Section 2.1 during the meeting as needed.

2 Section 2: Final Project and Report (20 points)

This section constitutes the main body of your final project. You will:

- Choose a project direction from the categories below.
- Implement and evaluate one or more parallel solutions.
- Design and run a thorough experimental evaluation.
- Submit:
  (a) A written report.
  (b) A GitHub repository with your code and tests.

The entirety of Section 2 is graded out of **20 points**, broken down in the grading rubric at the end of this section.

2.1 Project Categories

Your project can fall broadly into *one* (or a well-justified combination) of the following categories (this list is not exhaustive - if you have an alternative idea, it should be thoroughly developed).

1) **Parallelizing a difficult problem from another course.**
   - Choose a computationally intensive or conceptually challenging problem from a different course (e.g., machine learning, numerical methods, graphics, vision, data mining).
   - Design and implement one or more parallel algorithms/implementations for this problem.

2) **Applying kernel fusion to a larger composition of kernels.**
   - Identify a workload composed of multiple kernels (e.g., a sequence of linear algebra operations, image-processing stages, or neural network components).
   - Apply kernel fusion techniques to improve performance.
   - For reference see:

Kernel Fusion Article

3) **Implementing an attention variant in CUDA.**
   - Implement a variant of attention (e.g., scaled dot-product attention, flash-style attention, sparse attention) in CUDA.
   - Explore and compare different kernel designs, memory layouts, tiling strategies, etc.

4) **In-depth performance analysis across hardware/software targets.**
   - Choose an algorithm or kernel (e.g., FFT, matrix multiplication, graph algorithm).
   - Profile and compare performance across multiple hardware and/or software environments:
     - Different CPU architectures, GPU models, or node types.
     - Different compiler flags, libraries, or parallel programming models.
   - Your emphasis is on deep analysis and explanation, not just raw benchmarks.

5) **Worklog: iterative performance tuning on a target problem.**
   - Start from a baseline implementation of a nontrivial problem.
   - Maintain a "worklog" documenting your optimization steps, design choices, and performance changes.
   - For an example of the style of worklog (not to copy, but to emulate the spirit), see:

     CUDA MMM Worklog Example

2.2 Testing Requirements

Regardless of which path you choose, your project must include:

- **At least 25 unit tests for correctness.**
  - These tests should validate that your implementation(s) produce correct or acceptable results across a variety of inputs.
  - Where appropriate, tests should cover edge cases, varying problem sizes, and stress tests.
  - Each test should have a reason for being chosen. Why is this helpful to test on?
- **At least 10 tests that demonstrate speedups.**
  - These tests should clearly compare performance against a baseline (e.g., sequential or naive parallel implementation).
  - You should report timing, speedup, and (where applicable) efficiency metrics.

Your tests should be executable from the GitHub repository with clear instructions in the `README`. We should be able to reproduce at least a subset of your reported results using these tests.

2.3 Expected Deliverables

You must submit the following:

1. Written Report

A report that follows and addresses the grading scheme in Section 2.5. At a minimum, your report should include:

a) **Introduction and Problem Description**

- Describe the problem you chose in detail.
- Explain why it is interesting or challenging from a parallel perspective.

b) **Methods / Parallelization Strategies**

- Explain what strategies you used for parallelization and *why* you chose them.
- Discuss relevant theoretical considerations (e.g., work, communication, memory hierarchy).
- Describe different implementations you tried (at least 5 variants; see grading rubric).

c) **Experimental Setup**

- Describe the hardware (CPUs/GPUs, number of cores, memory, etc.) and software (compilers, libraries, frameworks) used.
- Explain how tests were run, input sizes, and any important configuration details.

d) **Results and Analysis**

- Present detailed results, including:
  - Speedup and efficiency metrics for at least one implementation across your test set.
  - Comparisons of at least 5 separate implementations and their speedups on some subset of tests.
- Use tables and/or plots to clearly present data.
- Discuss what worked, what did not, and how performance changed as you tuned the implementation.

e) **Discussion and Conclusions**

- Summarize the main takeaways.
- Reflect on what you would try next with more time (further optimizations, different architectures, larger problem sizes, etc.).

2. GitHub Repository

You must provide a link to a public or appropriately accessible **GitHub repository** containing:

- All source code for your implementations.
- Your test suite (a single test can be in both the categories below if you test a large problem both for correctness and speedups):
  - At least 25 correctness tests.
  - At least 10 performance/speedup tests.
- A clear `README` that includes:
  - How to build and run your code.
  - How to run correctness tests.
  - How to run performance tests and reproduce (at least some of) the reported results.
  - Any required dependencies or environment setup instructions.

We should be able to clone the repository, follow your instructions, and observe speedups similar to those reported in your paper for a subset of your experiments.

2.4 Submission and Deadline

All materials for Section 2 (report and GitHub link) must be submitted via the Canvas Final Project link by:

December 18, 2025

Please see the course Canvas site for the submission link and any additional logistics.

2.5 Grading Breakdown for Section 2 (20 points)

Section 2 is graded according to the following rubric:

(1) **Reproducibility and Replication of Results (8 points)**

- We should be able to clone your GitHub repository, follow your `README`, and run a subset of your tests.
- For that subset, we should see results that are consistent with those reported in your paper, including:
  - Correctness on the tested inputs.
  - Observable speedups relative to your baseline(s).

- Points are awarded based on:
  - Clarity and completeness of instructions.
  - Ease of running tests and scripts.
  - Consistency between reported and reproduced results.

(2) **Problem Description and Parallelization Strategy (4 points)**

- Your report should include a clear and detailed description of:
  - The problem you chose (including necessary background).
  - Why it is meaningful in the context of parallel algorithms.
- You should thoroughly describe what strategies you chose for parallelization and *why*:
  - Parallel decomposition, communication patterns, synchronization, etc.
  - Any relevant trade-offs you considered.

(3) **Results, Implementations, and Performance Analysis (8 points)**

- Your results section should be **detailed and exhaustive** within the scope of your project.
- At a minimum, it must include:
  - Speedup and efficiency metrics from the test suite for **at least one** implementation.
  - A comparison of **at least 5 separate implementations** and their speedups on some set of tests.
- You should carefully explain:
  - What exactly was done in each implementation (e.g., different tilings, memory layouts, synchronization strategies, kernel fusion variants, etc.).
  - What other things you tried (even if they did not work as well) and what results they produced.
  - How you tuned your implementation(s) to obtain the best possible performance.
- Strong projects will:
  - Provide insightful analysis rather than just tables of numbers.
  - Connect empirical results back to theoretical expectations.
  - Clearly identify bottlenecks and explain improvements across implementations.

Total: 4 points (Section 1) + 20 points (Section 2) = **24 points**.