CS598APE	Due: **Feb 21, 2025**
: Project Proposal	
Name: Don't forget to add your name	*Collaborators: List all sources used*

Goal

The goal of this project is to give you experience with designing and implementing performance engineering techniques to optimize an application of your choice, preferably with a novel goal that goes beyond current literature.

A secondary goal is to gain exposure to topics of current research. We encourage project proposals of the following kind(although you are free to submit an alternative proposal if you want to):

- Optimizing an open source application or tool
- Your current research project
- Implementing and optimizing an existing algorithm or reimplementing a paper.
- Any other novel idea related to performance engineering.

Although a novel contribution is optional, we highly encourage all of you to consider doing something that counts as a research contribution; we are happy to advise you in developing such components in your project.

You must work in **teams of atmost 3 members** for this project. All members of a team get the same grade, *without exception*. Because these are team projects spanning about fourteen weeks, these projects will be expected to meet a high standard for completeness, testing, and documentation. *Correctness, scalability to realistic programs, modularity, and code quality are more important than how much functionality you implement, so start simple and add features and optimizations incrementally, testing thoroughly at each step.*

The following sections describe the deliverables for this project with their due dates, some issues in working as a team, and finally a list of suggested projects. You are also welcome to identify other problems of interest and discuss them with me.

Testing and Evaluation

We recommend that whatever code you turn in should be as thoroughly tested as possible, first on a large number of small unit tests, and then on medium-to-large real-world programs up to 100K lines of code. In particular, we are a bit *handwavy in this handout*, but the idea is to inspire you to think about how would you go about implementing testing your project.

Testing performance in general is a task that requires care and creativity(because it often depends on what you are trying to optimize), and cannot be taught in lectures: you must do it to learn it!

Deliverables

Friday, Feb. 21, 11:59pm

Short project proposal, as ascii text via e-mail (2 pages max). Prefix your email subject with **[CS598APE Proposal]**. This proposal should include:

- a short description of the problem to be solved;
- a list of references;
- a short summary of the state of the art, and how you want to improve on this.
- a break-down of the work into a list of concrete tasks, *credible* (but tentative) completion dates and how the tasks will be divided between the two team members.

Your list of tasks should include two explicit tasks for testing: one for creating and testing with small unit tests, and another for setting up and testing with larger programs. Don't underestimate these tasks: they can each take one person a week or more, just to do a minimally adequate job. Moreover, you will be submitting your test cases as part of your reports, and the quality of the test cases will impact your project grade. *Note that each of these two tasks should be shared by all team members*, i.e., both write unit tests and both set up and run real programs.

Thursday, March 27, 11:59pm

Interim progress report, as ASCII text in the body of an e-mail message (3 pages max). Prefix your email subject with **[CS598APE Midterm]**.

By this stage you should have *partial working code for a subset of proposed functionality* that has been tested on both unit tests and medium-to-large input programs. The functionality can be small, but it should be solid. Describe what you have accomplished, including any relevant preliminary results for programs that work.

Friday, April 28, 11:59pm **P**ass / Fail

The final report should be emailed to us(with the subject prefix being **[CS598APE Final Report]**), and should include the following components:

- Modular, well-documented, working code (we recommend the use of a git repo)
- Test suites
- PDF file for project report, as outlined below

The final report should reflect the correct status of your project, and how complete/incomplete it is wrt testing. Any code you write in C/C++ should be "*valgrind-clean*": no errors or warnings from valgrind for *any* of your input tests. *The report should mention the results of running valgrind in such cases.*

The report should explain the layout of the files in the your artifact, unless you have a README.md in your artifact explaining how to compile and run your code. Don't forget to specify hardware requirements.

Final Report

Your final project report should be submitted to me via email in PDF form. It should be 5 pages max, using the ACM Reference Format(excluding the References and Appendix). Page estimates below are approximate.

The report should include:

1. Introduction, including the problem statement and motivation (1/3 page).
2. Brief summary of existing work in the literature, with citations (1/2 page).

3. High-level overview of your technical approach (e.g., algorithm, design, etc.) (1 page).

4. Implementation details (1-1.5 pages):

   (a) Prior analyses or transformations required by your code.

   (b) Major code components (passes, data structures, and functions).

   (c) Testing strategy and status: unit tests, small tests, larger tests

   (d) Implementation status describing what functionality works and what doesn't work, for various tests.

   (e) A description of where to find your source code, executable, and input programs, and how to run the executable for example inputs.

5. Experimental results: (1-2 pages): Choose results appropriate for your project, again for complete programs.

6. References.

7. Appendix: An Extended Example (as long as necessary): Use part of one of the benchmarks (or design your example based on one of the benchmarks) to illustrate what your code does. Choose the example carefully to highlight the major technical capabilities and limitations. You can use more than one example if needed, but no more than absolutely necessary.