Midterm Project

In the midterm project, each student should implement one of the three algorithms taught in the lectures: matrix multiplication, samplesort, or semisort.

The goal of this project is not to get the fastest implementations for these problems in the world---these are some of the mostly-studied algorithms in computing for decades. The goal is to provide a chance for you to start from a simple algorithm and experience how you can optimize the performance based on the theory and practice mentioned in the class. Namely, in your report, you should provide all versions of your implementation and emphasize where the improvements are from and by how much. Of course, the running time for the final version also matters since basically that shows how much you attempt to engineer the performance. The midterm project will give you an intuition about what you can do in a few weeks so it will help you to pick a final project with more realistic goals.

The correctness of your implementation is crucial. Yihan has shown the joke in the class, and it makes no sense to optimize performance when the algorithm is wrong. You need to design strategies to guarantee that all versions of your implementation give correct outputs. **In your report, you need to specify what you do for this.**

You should not only optimize the performance but also reason the performance. Similar to assignment 0, you should show self-speedup, scalability, and any other measures that you think are helpful.

You should not use and copy any code in highly optimized parallel libraries. For example, you should not use MKL for matrix multiply and anything other than the scheduler in PBBS. If you do want to use some of them, talk to the instructor before. However, you can use sequential libraries, such as STL.

Matrix Multiplication

Algorithms and implementations for matrix multiplication are discussed in Lectures 2 and 5. For experiments, you should use double-precision square matrices with size 4096. Since the pseudocode and directions are already given, the maximum score you can get for MM is 90%, but you can pick the bonus as mentioned.

[Bonus for MM 1, 10%] Apply the implementations to non-square versions. Design a reasonable experiment setup for testing performance.

[Bonus for MM 2, TBD] Any further optimizations beyond Version 9. The bonus score depends on how much you do.

Samplesort

Implement and engineer the performance of samplesort discussed in Lecture 6. You should start with a sequential baseline without the optimizations in Slide 26. Then you should seek parallelism and better I/O-efficiency gradually. You need to provide running time for 100 million uniformly random 32-bit integers for all versions. Remember to design unit tests for your sorting algorithm and discuss your approach in the report.

[Bonus for Samplesort, 10%] Apply the I/O-efficient matrix transpose and distribution (second point in Slide 26).

Semisort

Implement and engineer the performance of semisort discussed in Lecture 7 based on the content in Lecture 6. You should also start with a sequential baseline without the optimizations in Slide 26 in Lecture 6. Then you should seek parallelism and better I/O-efficiency gradually. You need to provide running time for 100 million random 32-bit integer-integer pairs for all versions, on inputs generated from uniform, Zipfian, and exponential distributions. Remember to design unit tests for your semisort algorithm and discuss your approach.

[Bonus for Semisort, 10%] Apply the I/O-efficient matrix transpose and distribution (second point in Slide 26).

[Additional Bonus: AWS, 10%] For all three problems, you can test the performance of your code on Amazon EC2 (https://aws.amazon.com/ec2/), which is a cloud service that provides you parallel machines with different specs. The main difficulty is to set up the environment to run your code. You are welcome to discuss with classmates about this step. Once you register using your UCR email account, you will have $50 credit, which should be sufficient to rent middle-end machines to finish this project.

Submission

You should submit your report and your code. You should also specify how to run your code, especially the final version, for the standard input mentioned above. For semisort, you should specify how to run on uniformly random, Zipfain- and expeonetial-distribution input instances.

Grading

The grading is based on the following three criteria:

1. The design of your approach (30%). For example, the correctness guarantee, whether the versions of your implementation are reasonable in illustrating and explaining the optimizations, etc.
2. Absolute performance (30%). For students working on the same problem, you should provide the running time of the required input size. **Your submitted code should be runnable on the course server and give running time matching the time in your report.** The student with the correct and fastest running time receives full points, and other students get scores relative to the best one. The instructor will verify the running time of some of the submitted code, so it is crucial to make your code more user-friendly for easy testing. **Please include in your report what parameters in the command line I should use to get the best performance for your code.**

3. Experiment (40%). Please show abundant experiments in studying the performance of your implementation. Examples include scalability, self-relative speedup, and anything else you think helpful to explain your points. You can also use software like Valgrind or cilkprof.

While you need to submit your code together with the report, you should not expect the instructor to run all the experiments on the course server that you have run. Hence, **writing a good report to clearly illustrate how you meet the previous criteria is crucial and the quality of your report can largely affect your score.** You should expect at least two days in writing the report.

Collaboration Policy

You are welcome to discuss the implementation with your classmates verbally, but **you must not read or use any code from others**. All of your written work and code should be your own.