Programming assignment 1

Getting started

The problems below should be solved using C++. I recommend that you use a recent version of either the GNU compiler (v13 or above) or LLVM/Clang (v18 or above). If you run Linux, just install the compiler via your package manager. If you use Windows, I recommend that you run the Windows Subsystem for Linux (WSL). It is possible to install the compiler on Windows, but it is easier to do so via WSL, and it works well with VSCode. If you use macOS, start with the Clang installed by the developer tools. The Apple version of Clang does, unfortunately, not support OpenMP. You can complete Problem 3 with threads, but if you want to try OpenMP, you must install it separately. You can use a package manager, such as HomeBrew, or see the instructions from the R-project.

You are allowed to use any sources you can find. The code you submit should be written by members in the team, but it can be heavily inspired by things you found online. Any team member must be able to explain all parts of the code, so make sure you understand everything you are inspired by. A good starting point is the “Goto-paper,” *Anatomy of High-Performance Matrix Multiplication*. Several blog posts and YouTube videos also discuss the problem, e.g., can you multiply a matrix? (noob lesson).

Problems

Problem 1

Implement the standard three-loop matrix multiplication. Use as simple an implementation as possible since this will serve as your baseline. Make sure it compiles and runs before you move on to Problem 2.

Problem 2

Your task is to optimize the matmul from Problem 1 using what you have learned so far. Your solution should be single-threaded. Focus on ideas from the first few lectures, such as improving memory layout and implementing blocking to improve cache hit/use. You should also try to avoid as many dependencies as possible to let the optimizer use as much ILP as possible. Your solution should at least use blocking.

Use the numpy implementation (or some other fast matmul) as a reference point. However, remember that it is generally multithreaded by default. You can disable this, but exactly how you do it depends on which platform you are running on, so use the Internet.

Problem 3

Once you are happy with the performance of your solution to Problem 2, it is time to make it multithreaded. Use either threads, OpenMP, or a combination of both to improve the performance. You should at least experiment with different values to the parameters from your solution to Problem 2 (e.g., blocking size) and the number of threads.

Submission guidelines

The submission is in two parts: demo and code/report. When you are done, book a demo session with Morgan (by email or Slack), where the group shows their solution. Note that you must be able to run it on a computer that you bring. You can benchmark on another machine, but it must be runnable during the demo. If you pass the demo stage, you should submit your code and report via Moodle.

You are allowed to work in groups of one to three persons. All the submitted code and text should be written by the group. Please provide a README.txt (or .md) that describes how to compile and run your code. Your solution should include a report that describes the optimizations and the results of your benchmarking. Try to be as detailed as possible when you describe the optimizations, benchmarks, and results. Try to explain why you think something worked or not. How well you optimized the code as well as how good your report is will determine your final grade. Note that only one person in the group should submit to Moodle. Ensure that all names are provided in the report.

Submit your solutions as a single zip-file via Moodle no later than end of day March 30, 2025 (cutoff 08:00 March 31). Remember that you must have completed the demo before you can submit, so try to book it at least a week before the deadline. It can be difficult to find a timeslot close to the deadline. You are allowed multiple demo sessions, so it makes sense to start early.