C++ Optimization

Hayden Cook

This lab focused on optimizing C++ used to simulate a hotplate. The code given to us was highly unoptimized, blatantly contradicting common c++ programming practices such as accessing elements in row major order and passing variables into functions by reference. Thus it was fairly easy to find ways to speed up the code.

Figure 1 summarizes the optimizations that were applied to the program as well as their runtimes.The code was compiled and ran on my personal desktop running Ubuntu 22.04 on a Ryzen 5900x processor. To help reduce jitter caused by system load, temperature, caches, etc, the program's runtime was measured three times and the average time was recorded. The program's (user) runtime was measured with Linux's `time` command. Each optimization in Figure 1 includes all of the optimizations to the left of it.

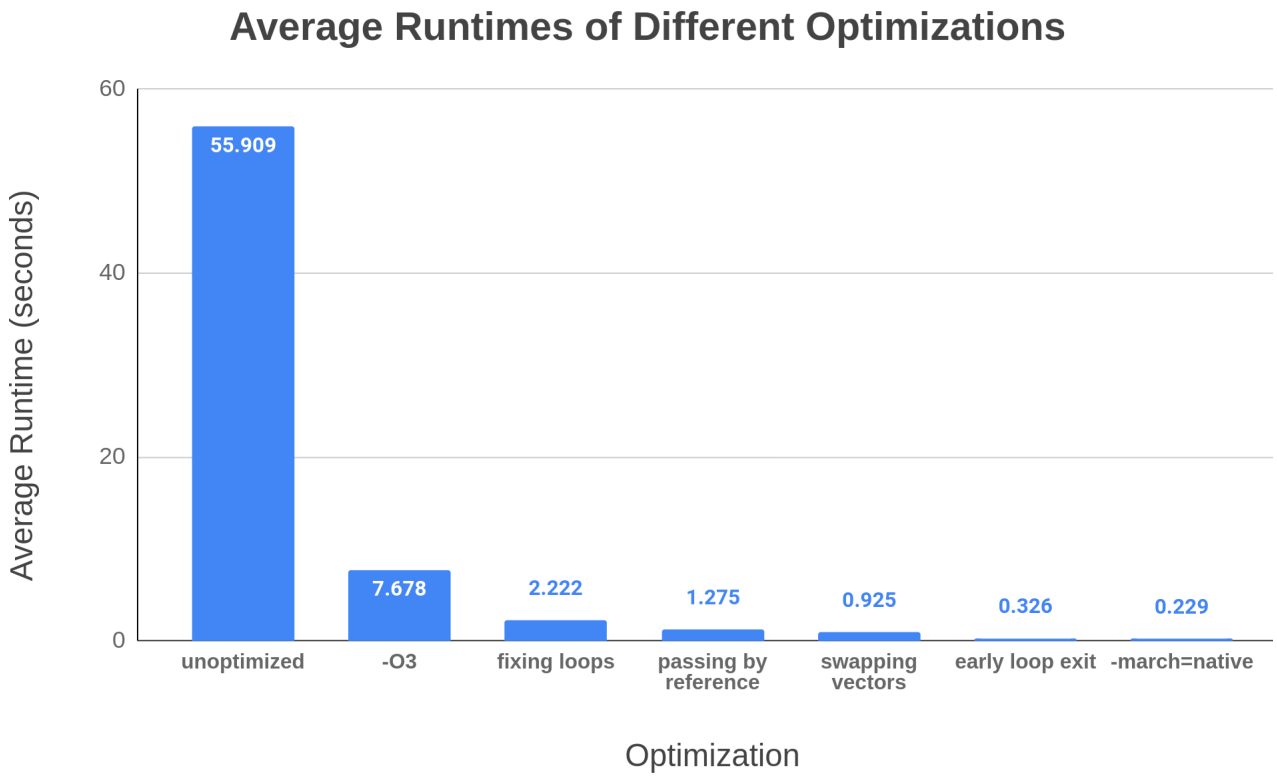

Figure 1. The optimizations performed and their associated runtimes.

Without optimizations, the program took 55.91 seconds to run. The first optimization I did was simply add the `-O3` flag to the compilation. This reduced the runtime by a factor of 7.28, reducing it down to 7.678 seconds. This is a huge speedup that was achieved without even touching the code.

Next, I swapped the inner and outer loops in both the `update()` and `maxabsdiff()` functions. This allowed the memory accesses in the loops to access the vectors in the same order it is laid out in memory, allowing for much more efficient memory accesses. This reduced the runtime to 2.22 seconds, a factor of 3.45, making this optimization have the largest speedup of all the manual optimizations performed.

Afterwards I changed all of the functions to pass parameters by reference instead of value, this allows the functions to use the parameters without spending time to copy them over. This reduced the runtime to 1.28 seconds, a factor of 1.74.

Next, instead of making a new `next` vector each iteration of the main loop, I create one before the main loop, pass it into the `update()` function by reference, and use `std::swap()` to swap the `next` and `current` vectors on each iteration of the main loop. This reduced the runtime to 0.93 seconds, a factor of 1.38.

The last manual optimization I did was change `maxabsdiff()` to `isStable()`, which returns a boolean and exits as soon as it finds an element that is bigger than epsilon. This reduced the runtime to 0.33, a factor of 2.84.

The final optimization I did was a compiler flag. -march=native allows the compiler to add instructions specific to the microarchitecture of the processor. This reduced the runtime to 0.23, a factor of 1.42.

The total speed up due to code changes alone was 23.55x. The total speedup with all optimizations (manual and compiler flags) was 244.14x. This shows how much time can be saved optimizing a program that will run often, or for a long time.