ECE 459: Programming for Performance
Lab 4 -- Profiling

Jeff Zarnett, Stephen Li
Updated by Bernie Roehl, December 2020

Due: April 7, 2021 at 11:59 PM Eastern Time

Profiling

One of keys to improving performance is identifying bottlenecks that are slowing a program down. The key to doing that is the use of profiling tools.

In this assignment, you are provided a simulation of a Hackathon. The program is slow, and you will use some profiling tools to analyze the code to find out where it's spending its time. Based on what you find, you can make changes to the program to speed it up.

Background

The Hackathon has three different kinds of threads: Students, Idea Generators, and Package Downloaders. The Idea Generator generates an idea the way many startup companies are pitched, as an equivalent of a known service for a new audience, such as "LinkedIn for Service Animals". Students work on ideas, and working on an idea requires downloading some number of software Packages. The Hackathon simulation runs until all ideas have been built.

Since we have multiple threads, the order of outputs may differ. However, we can still check for correctness by taking the xor (exclusive or) of the SHA256 hashes of every idea generated and every package downloaded.

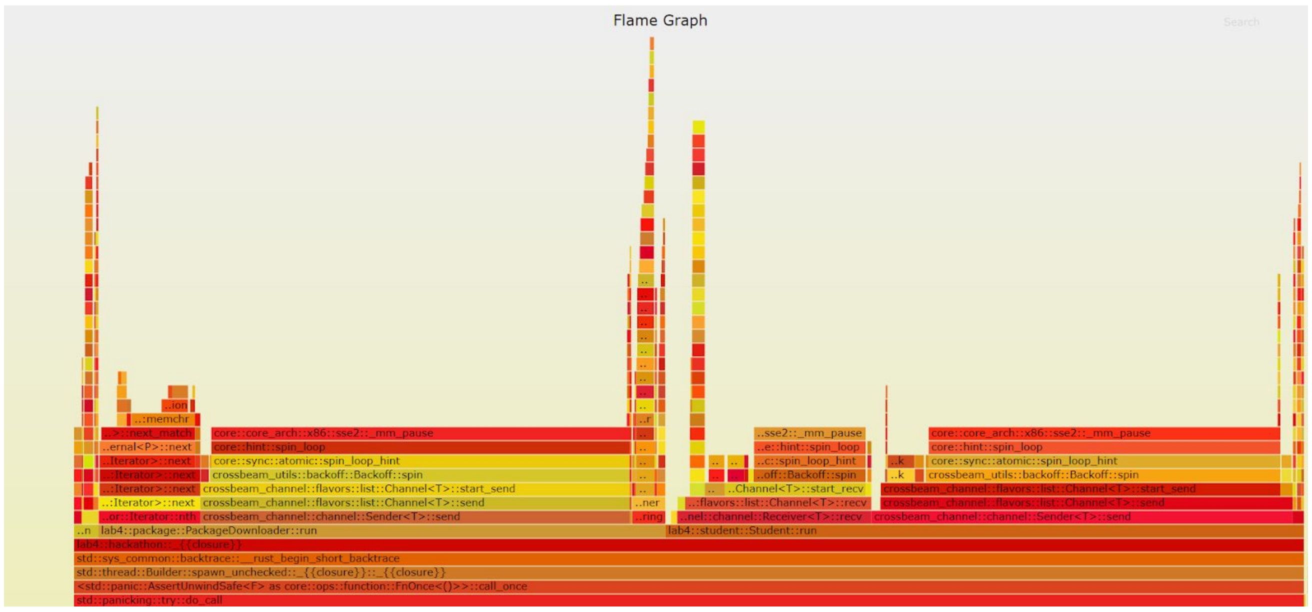
Analyzing and Optimizing the Program

Your goal is to speed up the program. You should therefore do some analysis using profiling tools. You may have some ideas about where to start immediately, but tools help you check your assumptions and find places for improvement. The basic workflow is to use profiling tools to identify what's slow, make changes, and re-evaluate to see how much (or how little) it improved the runtime of your program. If the program is sped up enough, you're all done (and can go on to writing your report).

You can use any analysis tools you like, whether or not they were presented in the lectures. However, for the purposes of your report you should use a flamegraph. A flamegraph is a visualization of profiling data, meant to show you where a program is spending most of its time. And they look cool, so there's that.

You probably need to know about how flamegraphs work. To read up on them, see http://www.brendangregg.com/flamegraphs.html. There are some useful YouTube videos linked there to give you guidance. In particular, the video at https://youtu.be/D53T1Ejig1 is useful. If you're in a rush, just watch from the 10 minute mark to about the 16 minute mark.

Here’s a flamegraph of the starter code:



For your convenience, a makefile target has been created that runs your program and generates the flamegraph. You can run it by typing “make”. This will generate an SVG (Scalable Vector Graphics) file called “flamegraph.svg” containing the graph, which can be viewed in any graphical viewing program or in your preferred web browser. For example, you can just type “firefox flamegraph.svg” to view it and interact with

In your report, explain how you used your profiling data to decide what to change. Be sure to include the flamegraph for the final version of the program and provide some explanation to the reader as to how your updated flamegraph demonstrates improvement over the starter code.

Restrictions. To prevent trivializing the problem, and to make it possible for us to compare your code against the starter code, here is a list of restrictions:

- main.rs cannot be modified.
- Cannot introduce bugs (memory leaks, deadlocks, etc.).
- The checksums printed out at the end of execution should be identical to the ones generated by the starter code.
- Cannot trivialize the threads e.g. moving everything to 1 thread to eliminate synchronization. You can rewrite how the threads communicate (different mutex/constructs usage, for example).
- All publicly observable behaviour/output for each thread must be preserved. For example, IdeaGenerator must add NEW_IDEA Events to the queue. It must also be responsible for inserting the poison pills for the Student threads.
- Any threads you create can only be terminated with the OutOfIdeas event (no passing "expected number of ideas").
- Do not hardcode any values that are read from files. We will be testing with different files of various sizes.
- No other changes that trivialize the simulation or execution (e.g. deleting functionality

or things that consume CPU time).

Rubric

Implementation (35 marks) Your code must preserve the original behaviour and can't violate any of the restrictions specified above. The changes you make should be supported by a profiling tool of some sort and not just be random changes.

Report (15 marks) 12 marks for explaining the changes that you've made, including your final flamegraph and some explanation for the reader. 3 marks for clarity of exposition.

Performance (50 marks) Your code must (of course) run faster than the starter code, the faster the better.