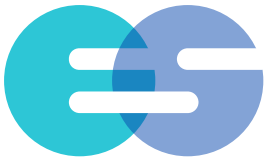
TUHH
Hamburg
University of
Technology



Debugging - CPU profiling

Seminar Report

submitted by
Lennart Brandin

Debugging – Methodology instead of Confusion
Hamburg University of Technology

May 2025

Abstract

Performance profiling is a method of analyzing where execution time is spent. Using profiling tools, it is possible to locate performance hotspots which are “sections of code that, if optimized, would yield the best overall speed-up” [1] (performance hotspot) .
To achieve this, the written user code is executed and recorded as a call graph, i. e. the relationship between a function and those it is calling. Such a recording can provide information about the number of calls and time spent in each function. This report aims to describe different techniques of profiling, including their benefits, limitations and possible inaccuracies. Additionally a conceptual overview of a predictive profiling tool, as proposed in [2], is given and compared to the usage of “traditional” profiling tools.

Contents

Abstract **1**

1 Introduction **1**
1.1 Motivation 1
1.2 Usage of electronic tools 1
1.2.1 Typst 2
1.2.2 VSCode 2

2 Preliminaries **3**

3 Conclusion **4**

Bibliography **5**

Declaration **6**

Glosarry / Acronyms Index

IDE **– Integrated Development Environment**
performance hotspot **– "sections of code that, if optimized, would yield the best overall speed-up" [1]**
TUHH **– Hamburg University of Technology**

1 Introduction

Improving the perfomance of a computational program is consistent goal in software development, while underlying platforms and abstractions are becoming more optimized, the computational tasks are growing increasingly complex.

Provided with the task of improving runtime, a software designer might take an educated guess or create specific tests to narrow down the hypothesis. Both lack a qualitative assurance, do the tests reflect accurately the performance issue or describe an entirely different performance issue? [1] CPU profiling offers quantitative measurements of runtime associated with specific user symbols, the profiling techniques vary in degree of intrusion, detail and accuracy of measurement.

This report will describe an overview different profiling tools and their uses. Those presented base on an iterative code-improvement cycle:

- Execution using a profiler
- Analysing results for performance hotspots
- Inspecting hotspot code for optimization

The concept of predictive profiling, i. e. predicting the profiling results without compiling the source code, as part of an Integrated Development Environment (IDE) , will be summarized. [2]

1.1 Motivation

In the process of iterative code-improvement, badly designed code may be visible early on, for example a long running function with little computational complexity. Manually finding performance hotspots in low level abstractions, for example, data structures or wrapping functions may be less obvious or simply infeasible due to the size and spread of function calls in complex programs. While it is possible to manually measure function runtime by printing time stamps, it is tedious, error prone, and does not provide information about the summed or average execution time of all collective calls to a function. Profilers aim to ease spotting these issues by collecting statistics during the execution and providing a profile used for visualising the performance distribution over the program. Different visualisation offer specialized insights, *flame graphs* show a linear overview of functions calls and their hierarchy, which give a general clue what the program is doing as the program execution progesses. A table, listing the functions and their runtime statistics, provides condensed information about how to spend optimization efforts.

Profilers do not require any prior implementation so they can applied on any program. Since the profile represents a statistical measure of the programs runtime, profiles can be accumulated and compared to indentify changes inbetween runs. This could be used to spot performance regression over multiple versions of a program. When runs are performed with different inputs, the profiles could be used to predict the application runtime growth.

Monitoring these information provide a way of identifying and solving a broad spectrum of performance issues before they occur in a critical manner. [1]

1.2 Usage of electronic tools

The following tools were used to design, research, write this report in the described extent.

1.2.1 Typst

Typst is a typsetting language, used here to simplify the design process by providing an underlying system of using templates, styles and citations. This report was designed using the TUHH typst ies-report template

1.2.2 VSCode

Visual Studio Code is a general purpose editor, mainly used here to write the report and provide macros and debugging features for Typst.

- Google Scholar - Source discovery
- Generative AI - ChatGPT acquiring topic outline, source discovery
- Generative AI - Github Copilot code autocomplete, not used for writing passges

2 Preliminaries

3 Conclusion

Using an abbreviation like Hamburg University of Technology (TUHH) is a good idea.

Bibliography

[1] R. Bernecky, "Profiling, performance, and perfection (tutorial session)," in *Proceedings of the ACM/SIGAPL conference on APL as a tool of thought (session tutorials)* -, New York, New York, United States: ACM Press, 1989, pp. 31–52. doi: 10.1145/328877.328879.

[2] X. Hu *et al.*, "Towards On-The-Fly Code Performance Profiling," *ACM Transactions on Software Engineering and Methodology*, p. 3725212, Mar. 2025, doi: 10.1145/3725212.

[3] S. L. Graham, P. B. Kessler, and M. K. Mckusick, "Gprof: A call graph execution profiler," *ACM SIGPLAN Notices*, vol. 17, no. 6, pp. 120–126, Jun. 1982, doi: 10.1145/872726.806987.

[4] P. Wood, "A Survey of Performance Analysis Tools."

[5] H. Xu, Q. Wang, S. Song, L. K. John, and X. Liu, "Can we trust profiling results?: understanding and fixing the inaccuracy in modern profilers," in *Proceedings of the ACM International Conference on Supercomputing*, Phoenix Arizona: ACM, Jun. 2019, pp. 284–295. doi: 10.1145/3330345.3330371.

[6] B. Gregg, "The flame graph," *Communications of the ACM*, vol. 59, no. 6, pp. 48–57, May 2016, doi: 10.1145/2909476.

Declaration

Hereby, I declare that I produced the present work myself only with the help of the indicated aids and sources. The thesis in its current or a similar version has not been submitted to an auditing institution before.

Hamburg, May 11, 2025
Place, Date

Lennart Brandin