Link-time Call Graph Analysis to facilitate user-guided program instrumentation

TECHNISCHE UNIVERSITÄT DARMSTADT

Tim Heldmann[1] and Sebastian Kreutzer[2], Scientific Computing, TU Darmstadt

[1]tim.heldmann@tu-darmstadt.de [2]sebastian.kreutzer@tu-darmstadt.de

Abstract

Code instrumentation is the primary means for extracting fine-grained performance data from programs. However, special care has to be taken to with regard to overhead management, as a full instrumentation can increase the runtime by orders of magnitude. Careful selection of the instrumentation configuration (IC), typically via filter lists, is therefore crucial to retain the performance characteristics of the original application. In order to give the user better control of what is measured, we have developed CaPI [1], an open-source tool for the creation of low-overhead, user-defined ICs. CaPI relies on a statically constructed whole-program call-graph as its central data structure, enabling the user to select functions based on the context they are called in, in addition to function-level metrics. Currently, this call-graph is generated externally by tools running on the source level [4]. This can be cumbersome, especially when targeting large-scale scientific software [2]. To mitigate this issue, we are developing an approach that runs the analysis on the LLVM intermediate representation during link-time optimization [3]. Running during link-time also allows us to embed the result into the binary, improving the workflow and usability of CaPI. In this talk we will discuss the advantages and shortcomings of link time generated call graphs compared to source level generated call graphs, and show how statically generated information can be augmented dynamically at runtime.

Content outline

1. Workflow of the CaPI instrumentation selection tool
2. Usability issues occurring with large-scale target applications
3. Link-Time idiosyncrasies
    a) Information availability at source vs link time
    b) How to regenerate missing information
4. Embedding knowledge into the binary
5. Requesting knowledge from binaries

References

[1] https://github.com/tudasc/CaPI

[2] Sebastian Kreutzer, Christian Iwainsky, Jan-Patrick Lehr, Christian Bischof, "Compiler-assisted Instrumentation Selection for Large-scale C++ Codes", C3PO'22 Workshop - Third Workshop on Compiler-Assisted Correctness Checking and Performance Optimization for HPC, Jun 2022, Hamburg, Germany (to appear)

[3] Sebastian Kreutzer, "Using Link-Time Call Graph Embedding to Streamline Compiler-Assisted Instrumentation Selection", Talk at European LLVM Developers' Meeting 2022, London, UK, https://llvm.org/devmtg/2022-05/

[4] Jan-Patrick Lehr, Alexander Hück, Yannic Fischler, and Christian Bischof. 2020. MetaCG: annotated call-graphs to facilitate whole-program analysis. In Proceedings of the 11th ACM SIGPLAN International Workshop on Tools for Automatic Program Analysis (TAPAS 2020). Association for Computing Machinery, New York, NY, USA, 3–9. https://doi.org/10.1145/3427764.3428320