IRHash:
Efficient Multi-Language Compiler Caching by IR-Level Hashing

Tobias Landsberg
landsberg@sra.uni-hannover.de
Leibniz Universität Hannover

Johannes Grunenberg
Leibniz Universität Hannover

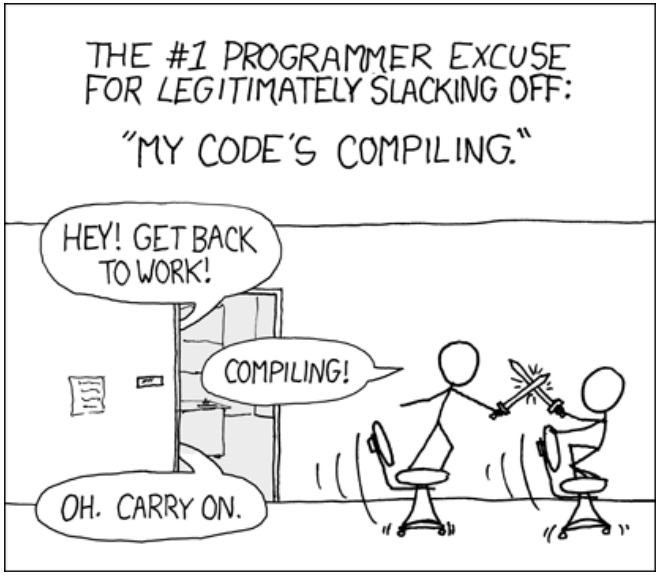
Christian Dietrich
Technische Universität Braunschweig

Daniel Lohmann
Leibniz Universität Hannover



USENIX ATC '25
July 2025

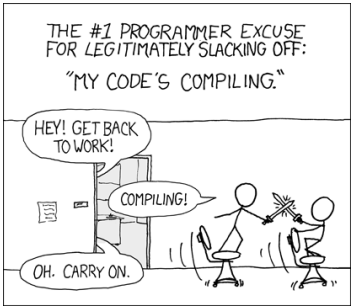
Funded by
DFG Deutsche Forschungsgemeinschaft
German Research Foundation



xkcd.com/303, CC BY-NC 2.5

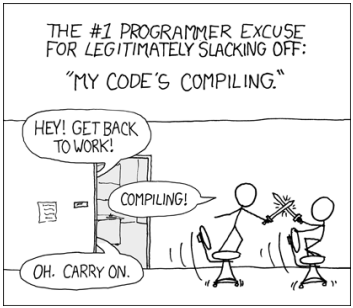
Motivation

- Expensive
  - Wastes energy
  - Consumes developer time
  - Costs money



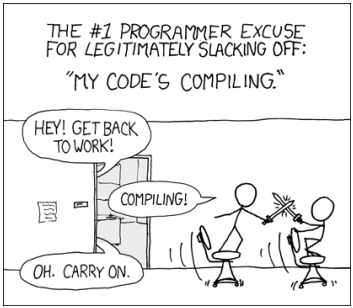
Motivation

- Expensive
  - Wastes energy
  - Consumes developer time
  - Costs money
- Omnipresent
  - In small project on a laptop
  - In large codebases built on servers



Motivation

- Expensive
  - Wastes energy
  - Consumes developer time
  - Costs money
- Omnipresent
  - In small project on a laptop
  - In large codebases built on servers
- Redundant
  - Often no new value
  - The same code was (probably) already compiled before



Ccache – The Prototypical Compilation Cache

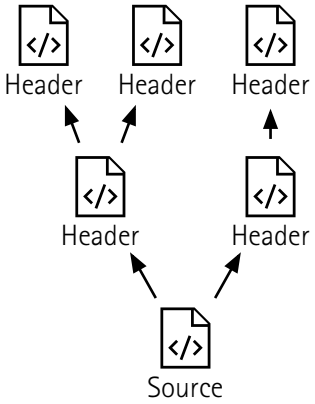
- Features
  - Hashes (preprocessed) source code
  - Caches compiler output by hash
  - About 10% speedup for incremental builds

Ccache – The Prototypical Compilation Cache

- Features
  - Hashes (preprocessed) source code
  - Caches compiler output by hash
  - About 10% speedup for incremental builds
- Limitations
  - Sensitive to formatting, whitespace etc.
  - Reacts to changes in unused code
  - Only works well with C/C++

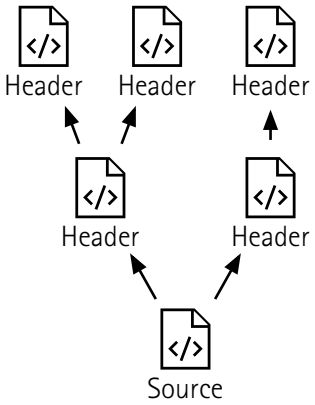
Ccache – The Prototypical Compilation Cache

- Features
  - Hashes (preprocessed) source code
  - Caches compiler output by hash
  - About 10% speedup for incremental builds
- Limitations
  - Sensitive to formatting, whitespace etc.
  - Reacts to changes in unused code
  - Only works well with C/C++

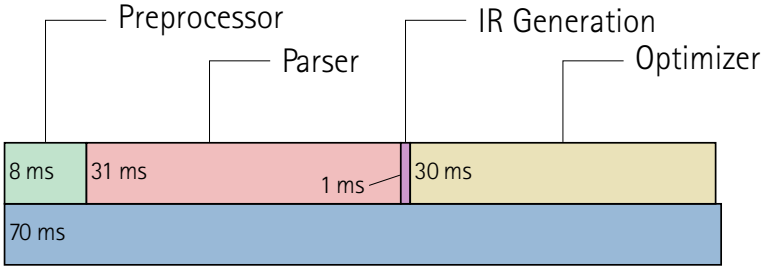


Ccache – The Prototypical Compilation Cache

- Features
  - Hashes (preprocessed) source code
  - Caches compiler output by hash
  - About 10% speedup for incremental builds
- Limitations
  - Sensitive to formatting, whitespace etc.
  - Reacts to changes in unused code
  - Only works well with C/C++
- cHash (USENIX ATC '17)
  - Hashes the abstract syntax tree (AST)
  - More precise than Ccache
  - About 16% speedup for incremental builds
  - Highly complex C-specific implementation

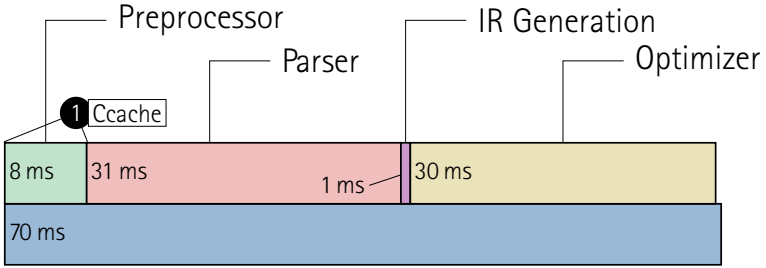


Build-Time Breakdown



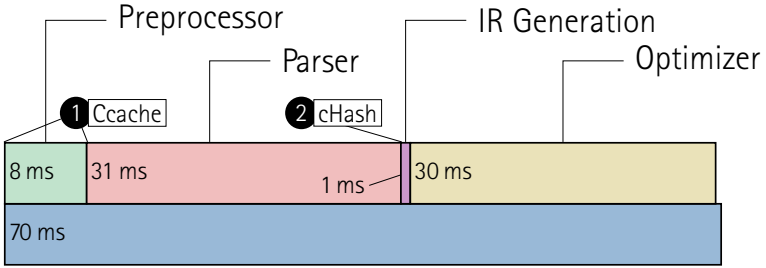
Build-time breakdown for the average translation unit when compiling OpenSSL using Clang

Build-Time Breakdown



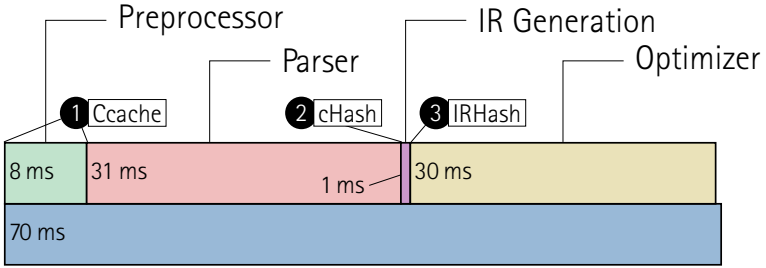
Build-time breakdown for the average translation unit when compiling OpenSSL using Clang

Build-Time Breakdown



Build-time breakdown for the average translation unit when compiling OpenSSL using Clang

Build-Time Breakdown



Build-time breakdown for the average translation unit when compiling OpenSSL using Clang

Why IR?

- Early enough in the compilation
  - Directly after code generation
  - Ability to skip optimization phase

Why IR?

- Early enough in the compilation
  - Directly after code generation
  - Ability to skip optimization phase
- Close to the final compilation result
  - High Precision
  - Insensitive to formatting, unused code, and many syntactic variations

Why IR?

- Early enough in the compilation
  - Directly after code generation
  - Ability to skip optimization phase
- Close to the final compilation result
  - High Precision
  - Insensitive to formatting, unused code, and many syntactic variations
- Data structures designed for frequent traversal
  - High Performance
  - Plain implementation due to conceptual simplicity of IR

Why IR?

- Early enough in the compilation
  - Directly after code generation
  - Ability to skip optimization phase
- Close to the final compilation result
  - High Precision
  - Insensitive to formatting, unused code, and many syntactic variations
- Data structures designed for frequent traversal
  - High Performance
  - Plain implementation due to conceptual simplicity of IR
- Targeted by frontends of numerous languages
  - Example: C, C++, Fortran, Haskell, Rust, Swift etc.

IRHash – Implementation

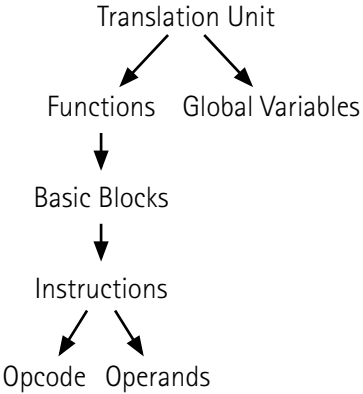
- Implemented as LLVM pass
  - Can be loaded by LLVM-based compilers
  - Less than 600 lines of code

IRHash – Implementation

- Implemented as LLVM pass
  - Can be loaded by LLVM-based compilers
  - Less than 600 lines of code
- Caching mechanism
  - On miss, compiler output is stored in a cache directory
  - On hit, cached output is linked to target location

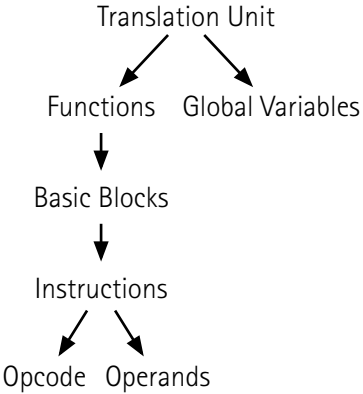
IRHash – Implementation

- Implemented as LLVM pass
  - Can be loaded by LLVM-based compilers
  - Less than 600 lines of code
- Caching mechanism
  - On miss, compiler output is stored in a cache directory
  - On hit, cached output is linked to target location



IRHash – Implementation

- Implemented as LLVM pass
  - Can be loaded by LLVM-based compilers
  - Less than 600 lines of code
- Caching mechanism
  - On miss, compiler output is stored in a cache directory
  - On hit, cached output is linked to target location



Successfully validated on 1.2 million translation units

Evaluation – Setup

- 16 open-source projects
  - C: Linux, OpenSSL, SQLite, Lua, PostgreSQL, CPython, Bash, Mbed TLS, SDL, and musl
  - C++: Bochs, Asio, Clazy, and GammaRay
  - Fortran: LAPACK
  - Haskell: ShellCheck

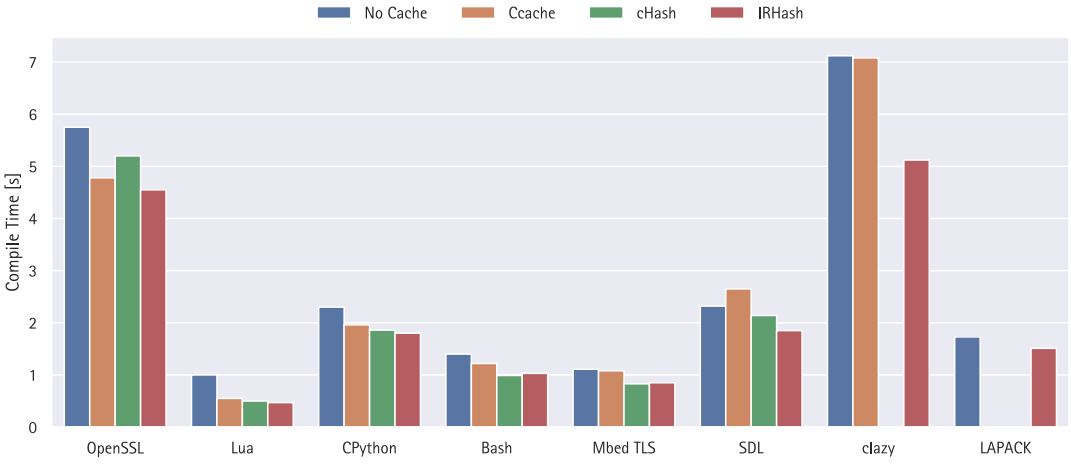
Evaluation – Setup

- 16 open-source projects
  - C: Linux, OpenSSL, SQLite, Lua, PostgreSQL, CPython, Bash, Mbed TLS, SDL, and musl
  - C++: Bochs, Asio, Clazy, and GammaRay
  - Fortran: LAPACK
  - Haskell: ShellCheck
- Development history of 100 commits

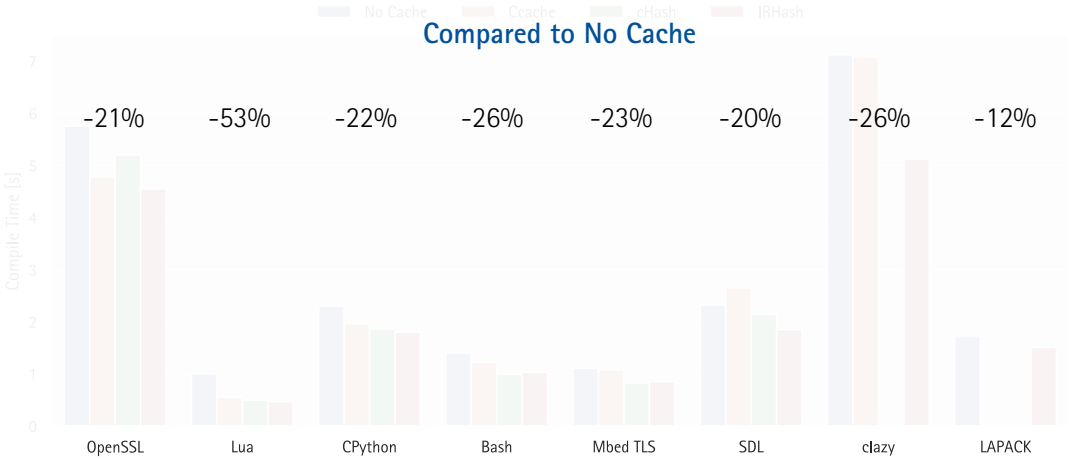
Evaluation – Setup

- 16 open-source projects
  - C: Linux, OpenSSL, SQLite, Lua, PostgreSQL, CPython, Bash, Mbed TLS, SDL, and musl
  - C++: Bochs, Asio, Clazy, and GammaRay
  - Fortran: LAPACK
  - Haskell: ShellCheck
- Development history of 100 commits
- Hardware
  - Intel Core i7-12700 (12 cores)
  - 32GB Memory
  - Turbo Boost disabled

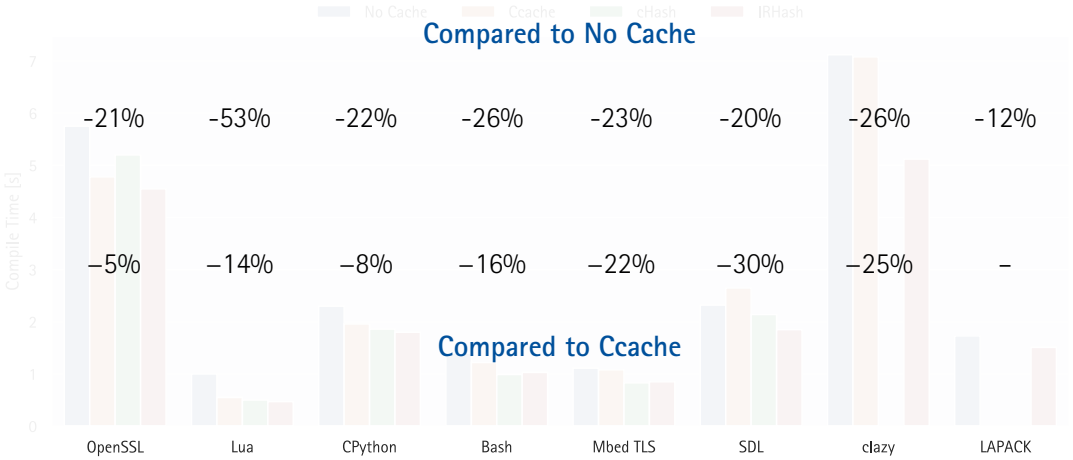
Evaluation – End-to-End Performance



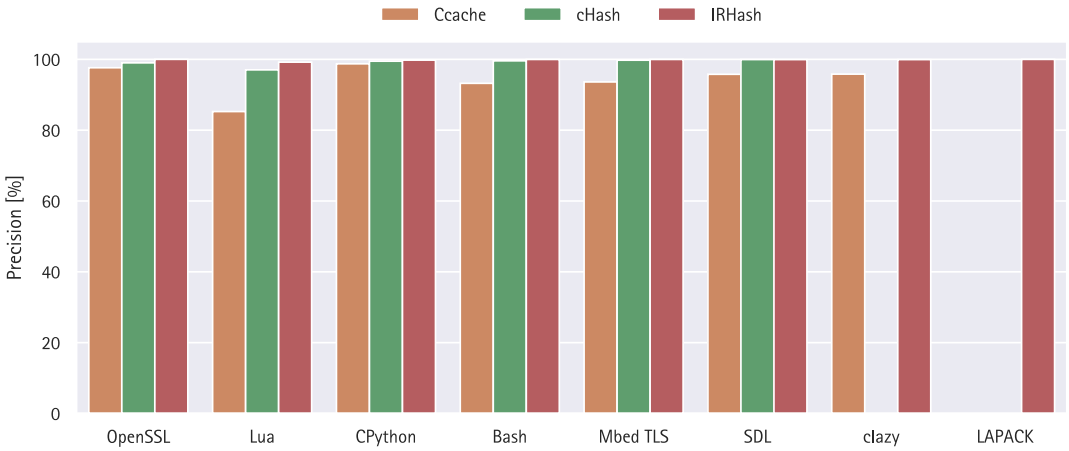
Evaluation – End-to-End Performance



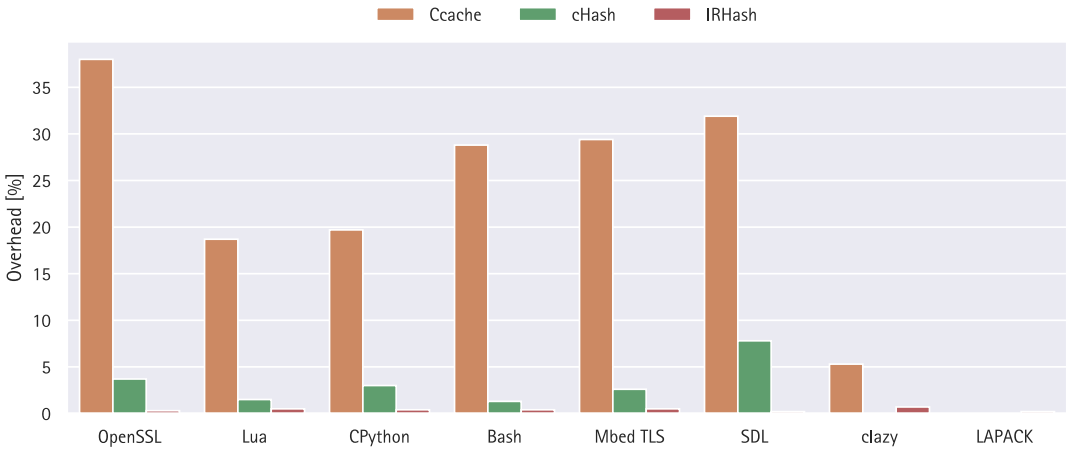
Evaluation – End-to-End Performance



Evaluation – Precision



Evaluation – Overhead



Summary

- We presented IRHash,
  a generalizable, maintainable, and efficient compiler cache
- Supports numerous languages
  - Evaluated on C, C++, Fortran, and Haskell
  - Expected to work on all LLVM IR
- Maintainable
  - Less than 600 lines of code
- Efficient
  - Precision of up to 100%
  - Two orders of magnitude less overhead compared to Ccache
  - Up to 30% faster than Ccache



zenodo.org/
records/15574403

landsberg@sra.uni-hannover.de