Optimizing after the compiler: FDO and LTO

Nicolo' Valigi

Jun 15 2019

Goal
Make code run faster

Plan

- The C/C++ compilation model: compiler and linker
- Optimizing modules, not functions
- Link Time Optimization (LTO) and ThinLTO
- Feedback-Guided optimization
- Bonus: reordering binaries (BOLT)

About me

- Sofware Engineer, mainly working on Robotics (drones, self-driving cars).
- Personal blog at nicolovaligi.com.

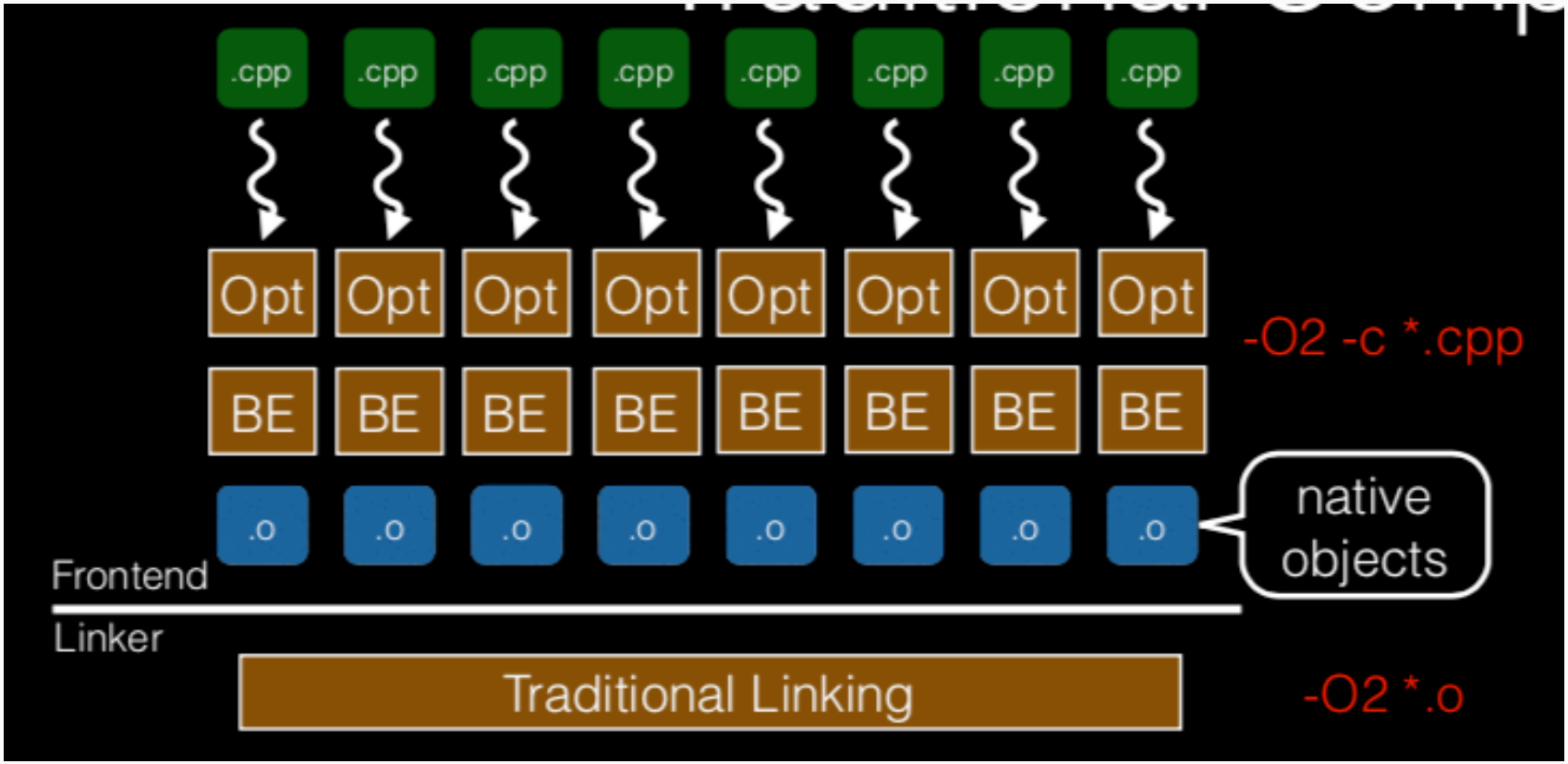
I work on the build infrastructure and toolchain at Cruise Automation.

Will tend to talk more about Clang than gcc, since that's what

Background

The compilation model

There are three main actors in C++ compilation: *preprocessor*, *compiler*, and *linker*. For this talk, we're going to focus on the compiler and linker.



- what is a compiler: produces object files; parsing, code

Thanks

IPA: Inter-Procedural analysis

TODO: - make examples of some optimizations, and how LTO can improve them. - run some numbers on clang? - -fwhole-program and the importance of linker visibility. - comparison between WHOPR and ThinLTO from the ThinLTO slides: the idea is the same (small serial + big parallel step), but the clang implementation is better.

For example, a conditional branch may turn out to be always true or always false at this particular call site. This in turn may enable dead code elimination, loop-invariant code motion, or induction variable elimination.

LIPO represents a major paradigm shift in program