Optimizing Clang with BOLT

Speaker: Amir Ayupov
Facebook

09/16/2021

Agenda

- BOLT overview
- Usage
- BOLT optimizations brief
- Clang optimization results
- Applying BOLT

Binary Optimization and Layout Tool

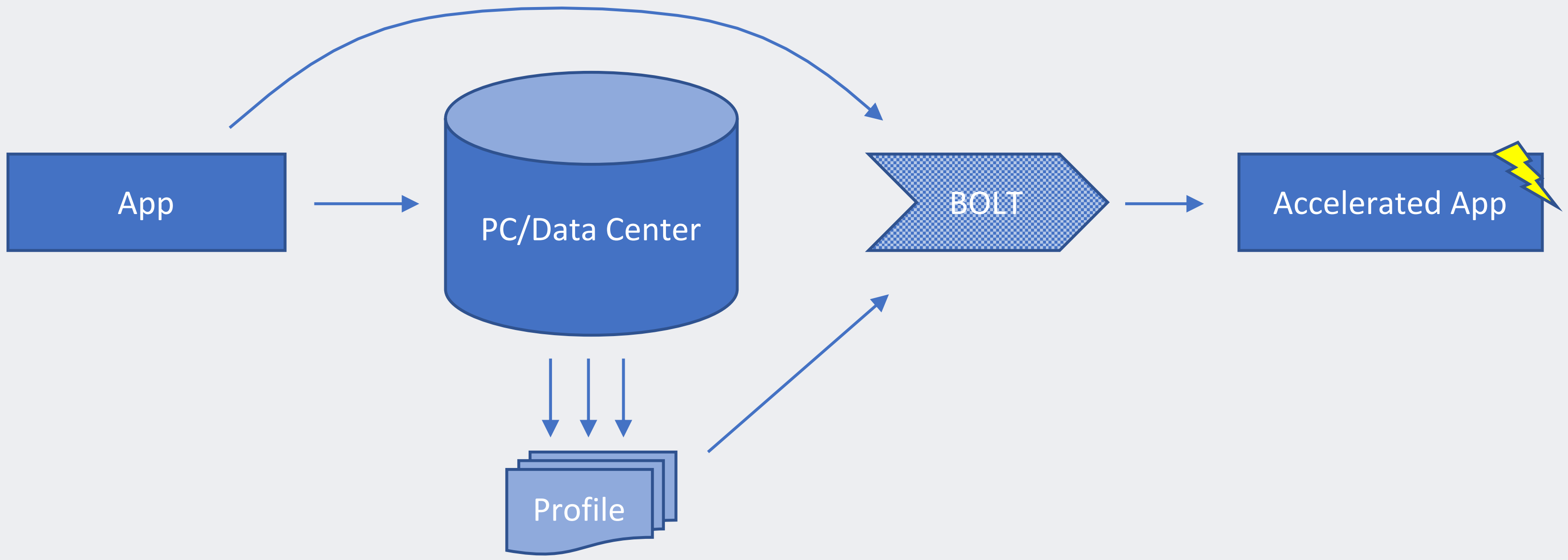
Overview

- Large application’s code does not fit in cache
- 90% of the time is spent in 10% of the code
- Use profile to identify and isolate/compact 10%
- Up to **52.1%** speedup on top of -O3
- Up to **20.4%** speedup on top of PGO and LTO
- Supports Linux ELF on X86-64 and AArch64
  - Experimental: X86-64 MachO

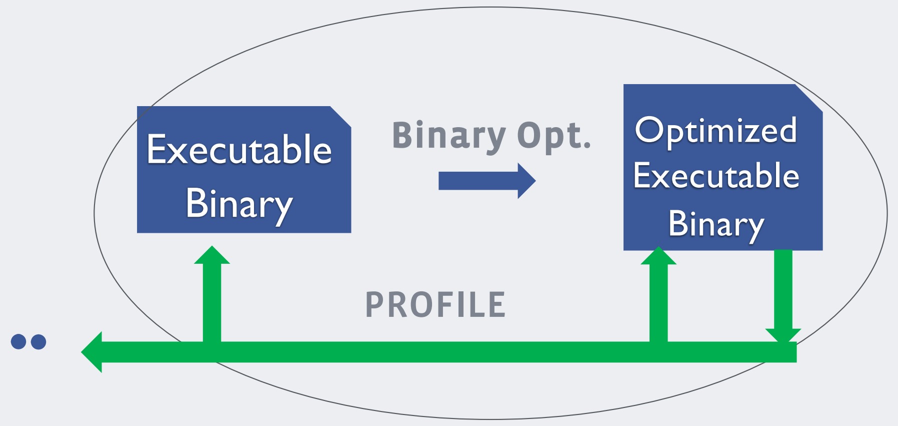
Usage

```
$ <re-link clang-12 with -Wl,--emit-relocs>
$ perf record -e cycles:u -j any,u -- clang-12 <input>
$ llvm-bolt clang-12 -data perf.data -o clang-12.bolt \
   -reorder-blocks=cache+ -reorder-functions=hfsort+ \
   -split-functions=3 -split-all-cold -icf=1 \
   -dyno-stats
$ perf stat ./clang-12 <input>
$ perf stat ./clang-12.bolt <input>
```

BOLT User Model



BOLT approach: post-link



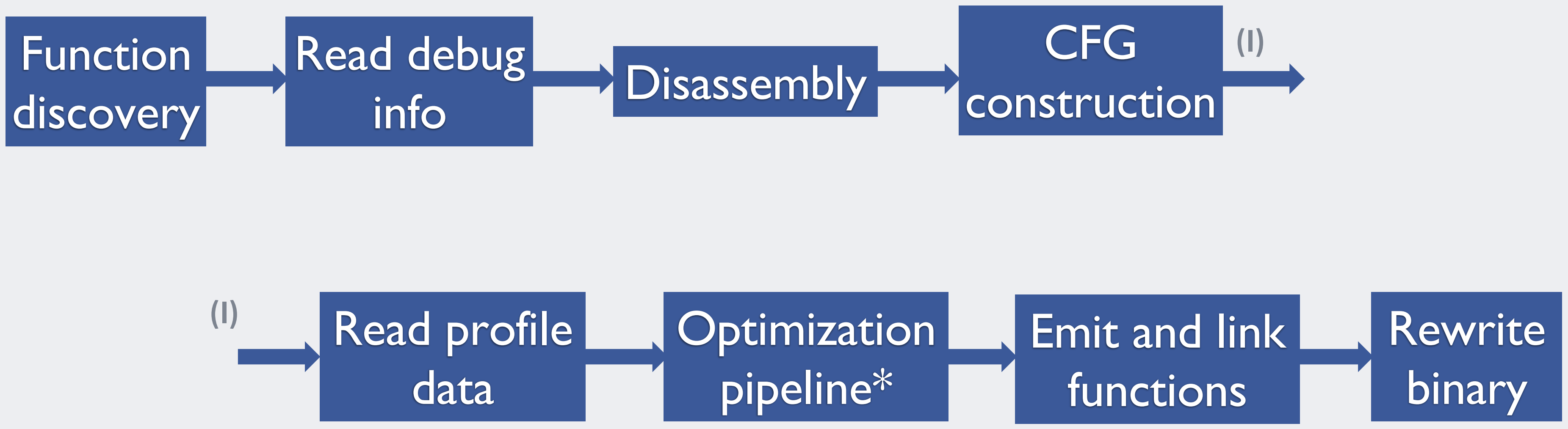
Focus at the end of the pipeline

Consumes profile with the highest level of accuracy

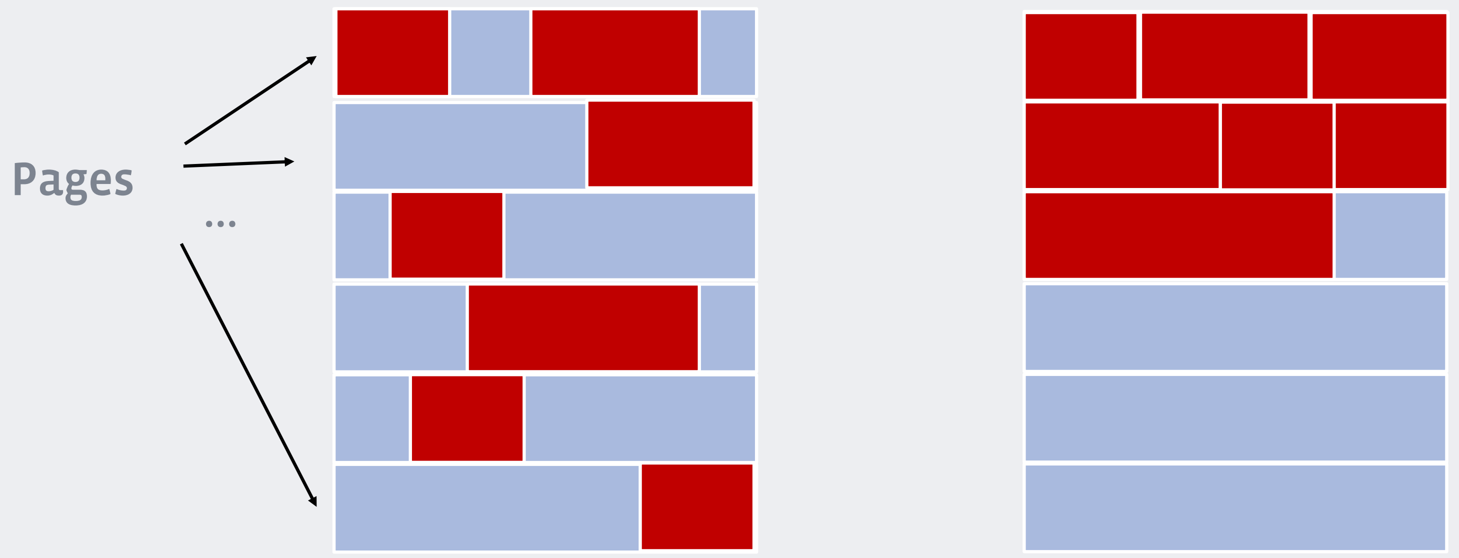
Compiler independent: Handles third-party libraries without source code

Fed by LBRs or instrumentation profile

BOLT pipeline



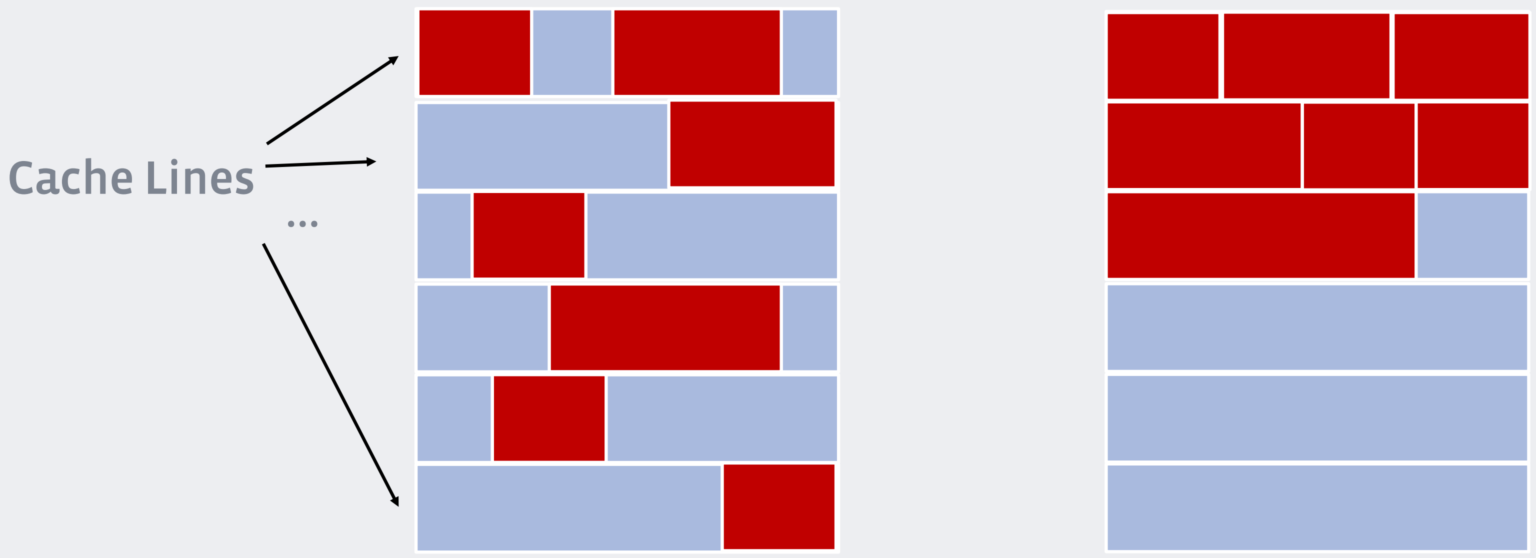
Improved Function Layout



🟥 Hot Function

🟦 Cold Function

Improved Basic Block Layout



🟥 Hot BB

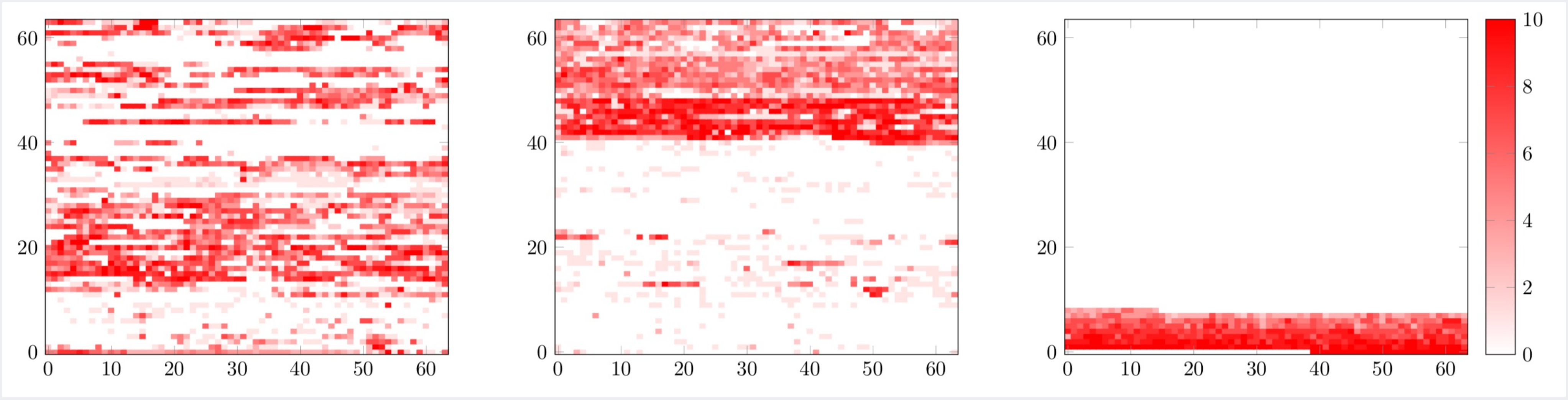
🟦 Cold BB

Code Layout

- Given in CGO’19 paper (Maksim Panchenko, Rafael Auler, Bill Nell, Guilherme Ottoni)
- Integrated HFSort+ (based on C^3[1]) for function ordering
- Pettis-and-Hansen [2] variation for basic block ordering
- Function Splitting
- Metadata updates

[1] Ottoni, G. and Maher, B. Optimizing function placement for large-scale data-center applications. CGO 2017.
[2] Pettis K. and Hansen R. Profile guided code positioning, PLDI 1990.

Clang Code Layout Heatmap



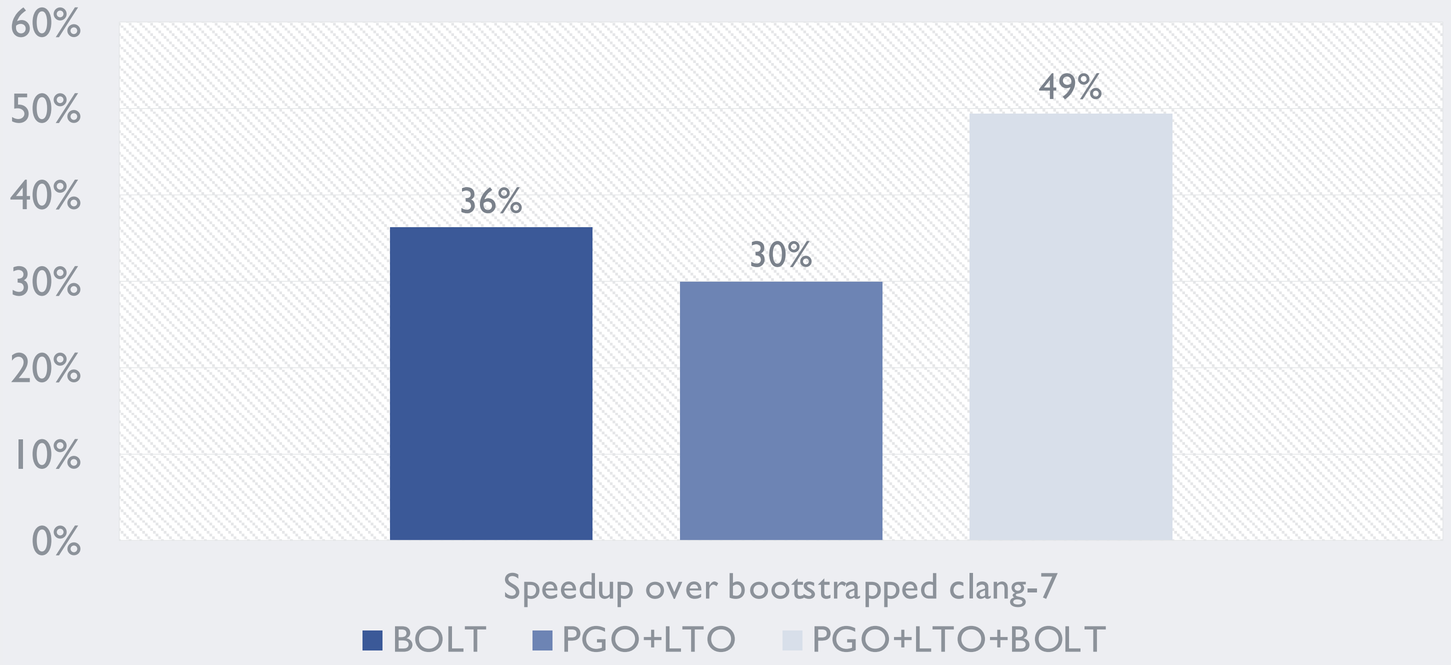
baseline PGO+LTO PGO+LTO+BOLT

Optimizations Beyond Code Layout

- Indirect-call Promotion/De-virtualization
- Identical-code folding
- *.rodata* optimizations
- PLT call optimization
- Macro-Fusion assistance
- Frame Optimizations

(see CGO’19 paper for optimizations breakdown)

Clang7 speedup over bootstrapped Clang clang build, Intel Ivy Bridge



Applying BOLT

- Manually:
  - Re-link application with `–Wl,--emit-relocs`
  - Collect LBR profile using perf on *representative* workload
    - If LBR is not available, use BOLT instrumentation
  - Run BOLT and replace the application
- After upstreaming BOLT to LLVM (around LLVM 14):
  - As part of Clang bootstrap/stage3, or test-release.sh

https://github.com/facebookincubator/BOLT