Optimizing Linux Kernel with BOLT

Maksim Panchenko
Facebook
Overview

- What is BOLT
- How it works
- Linux Kernel Challenges
What is BOLT?

Binary Optimization and Layout Tool

• Binary
  • Compiled and linked executable or dynamic library
  • Suffers from CPU front-end stalls

• Optimization
  • Double digit speedup on top of PGO+LTO

• Layout
  • Code Layout is the main optimization
BOLT

Code Layout Optimizations

Un-optimized  Better  Best
Clang and GCC Speedup
How to use BOLT?

- Collect profile
  - Sample run
  - Production environment
- Methods
  - Sampling (Linux perf tool)
  - BOLT instrumentation
- Link with `--emit-relocs`
- Apply optimizations by running BOLT
  - `$ llvm-bolt a.out -data perf.data -o a.out.fast <...>`
Compiler

- Uses profile for optimizations – PGO/FDO/AutoFDO
- Layout is the last step in optimization pipeline
- Profile no longer accurate

Compiler Solution

- Use two profiles
- Compile twice
- (*) Context-sensitive profiling
BOLT Pipeline

- Find functions and data in code
- Identify instructions inside functions
- Analyze instructions and build CFG
- Read profile and attribute to CFG edges
- Execute local and global optimization passes
- Generate code and process relocations
- Write code to file and update ELF
Linux Kernel Challenges for BOLT

- Self-modifying and self-patching code
- Custom unwinding and exception handling
- Re-writing the binary
Linux Kernel Challenges for BOLT

- Alternative instruction sequences
  - .altinstructions, etc.
- Exceptions
  - __ex_table, .pci_fixup
  - __bug_table
- ORC
- Possible approach: selective optimizations
Linux Kernel Challenges for BOLT

- Rewriting the binary
- Regular ELF
  - Allocate new segments(s) for code and data
- Kernel
  - Pre-allocate at link time
  - Expand existing segments
Thank you

http://github.com/facebookincubator/BOLT