Bolting the Linux kernel with profile instrumentation

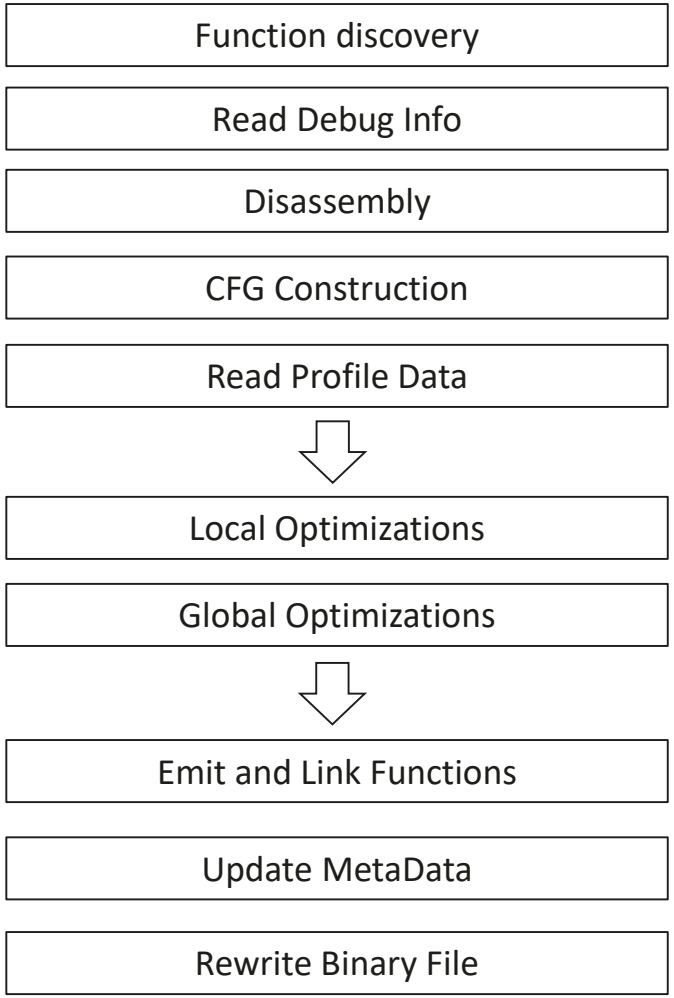
Wei Wei (weiwei64@huawei.com)
Fenglei Zhang(zhangfenglei@huawei.com)

HUAWEI

Introduction

- Binary Optimization and Layout Tool
- BOLT is a static post-link binary optimizer.
- Build on LLVM framework.
- Why a static binary optimizer?
  - ✓ the simplicity of the approach
  - ✓ the absence of runtime overheads
- Why use BOLT?
  - ✓ Speedup on top of LTO and PGO
  - ✓ Support both GCC and LLVM code
  - ✓ Perform some optimizations beyond than what a compiler can achieve

- Compilation pipeline



HUAWEI

Optimizing the Linux Kernel with Bolt

- Linux 5.12 kernel brings support for building the kernel with LTO + Clang compiler
- Linux 6.13 kernel introduces AutoFDO and Propeller optimization support
- Optimizing the linux kernel with BOLT :

  (Guide) https://github.com/llvm/llvm-project/blob/main/bolt/docs/OptimizingLinux.md
  2023 EuroLLVM - Optimizing the Linux Kernel with LLVM BOLT
  2024 Linux Plumbers Conference - BOLT - Binary Optimizer for Linux Kernel

- Limitations of bolting linux kernel:
  - ✓ **Only x86-64** Linux kernel
  - ✓ **Sample-based** profiling using LBR(Intel's last branch records)

How about other targets like AArch64 or RISCV?
Without LBR or BRBE? How to support instrumentation profiling?

HUAWEI

Bolting the Linux kernel with profile instrumentation

[BOLT][RFC] Enhance BOLT for Linux kernel

Subprojects BOLT

FLZ101 5 Jan 20

After adding support for AArch64, relocation mode and instrumentation, we managed to make BOLT for Linux kernel work on AArch64 and achieved an improvement of 8+% on our nginx benchmark.

We'd like to contribute our work to the community. Below is a basic plan:

1. Add support for AArch64. We need to refactor some code first to make it easier to handle different Linux kernel versions and different architectures.
2. Add support for relocation mode to enable more and better optimizations.
3. Add support for instrumentation to make BOLT for Linux kernel work on machines without LBR/BRBE.
4. Improve the documentation to make BOLT for Linux kernel easier to use.

Three kinds of functions in Linux kernel source code need to be specially handled:

1. Functions whose address must be kept. Their address is usually used in add/sub/comparision. We could gather a list of them in the middle end.
2. Functions that can not be changed at all. Some functions in assembly code have additional semantics/enforcements. For example, `irq_entries_start` defined in `arch/x86/include/asm/idtentry.h` is actually an "array" but BOLT can never know that and views it as an ordinary function. If BOLT applies instrumentation, basic block reordering, etc to it, a run time error happens. As you can see, BOLT just can not handle functions defined in assembly code reliably. We could gather a list of functions defined in C code in the middle end and optimize them only.
3. Functions whose code can change at run time. Linux kernel could patch a function at run time. In relocation mode, BOLT could create a copy of such a function and Linux kernel can only patch the copy. To avoid undefined behaviors at run time, we fill patch points in the original function with undef instructions.

- https://github.com/llvm/llvm-project/pull/130948

  [BOLT][Linux] Add support for instrumentation #130948

  Pull Request #130948 6 contains the following `BOLT for Linux` features:

  - support for AArch64
  - support for relocation mode
  - support for instrumentation

  FLZ101/test-bolt-for-linux 2 contains test scripts. It should work for x86/aarch64 and Linux 5.x/6.x.

- https://gitee.com/openeuler/llvm-project/pulls/173

  !173 [BOLT] Add support for kernel instrumentation of aarch64

HUAWEI

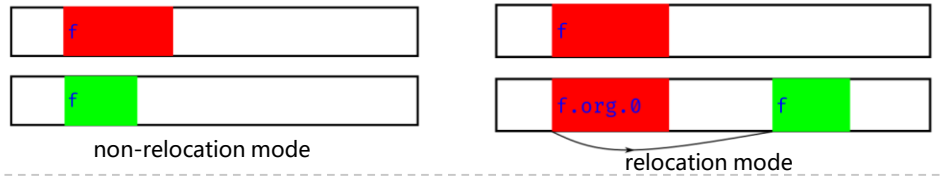
Current Status

Our Work:

- Enable kernel optimization for aarch64
- Enable relocation mode for kernel
- Enable bolt instrumentation for kernel
  - Support more features for linux kernel
  - Add indirect call instrumentation support

- Done
- In Progress

- **Non-relocation mode**
  - must keep function address
- **Relocation mode**
  - w/o opt: keep function address
  - w/ opt: create new function with new address
    - rename f to f.org.0
    - jump to f at the beginning of f.org.0



- **Instrumentation** (depend on relocation mode)
  - jump
    - leaf node
    - edge not in the spanning tree
  - call
    - before call
  - indirect call
    - before indirect call

- **Runtime**
  - indirect call runtime
  - emit profile

HUAWEI

Challenges

Code may be modified at boot time or runtime!

- **Inline assembly**

  E.g. a table for kernel, but function for bolt

```
Disassembly of section .text:

ffffffff822002a0 <irq_entries_start>:
ffffffff822002a0:       6a 20                   pushq  $0x20
ffffffff822002a2:       e9 d9 0b 00 00          jmpq   ffffffff82200e80 <asm_common_interrupt>
ffffffff822002a7:       cc                      int3
ffffffff822002a8:       6a 21                   pushq  $0x21
ffffffff822002aa:       e9 d1 0b 00 00          jmpq   ffffffff82200e80 <asm_common_interrupt>
ffffffff822002af:       cc                      int3
ffffffff822002b0:       6a 22                   pushq  $0x22
ffffffff822002b2:       e9 c9 0b 00 00          jmpq   ffffffff82200e80 <asm_common_interrupt>
ffffffff822002b7:       cc                      int3
ffffffff822002b8:       6a 23                   pushq  $0x23
ffffffff822002ba:       e9 c1 0b 00 00          jmpq   ffffffff82200e80 <asm_common_interrupt>
ffffffff822002bf:       cc                      int3
```

- **Linux features**
  a. Alternative instructions
  b. Exception table
  c. Bug table
  d. Static keys
  e. Static calls
  f. ...
- **Testing and Debugging**

Alternative instructions

```
1  struct alt_instr {
2      s32 instr_offset;
3      s32 repl_offset;
4      ...
5      u8  instrlen;
6      u8  replacementlen;
7      ...
8  } __packed;
```

structure definition

```
1  instr_offset:
2  BB0:
3    xxx
4  BB1:
5    xxx
6  BB2:
7    xxx
8
9  repl_offset:
10 BB0:
11   xxx
```

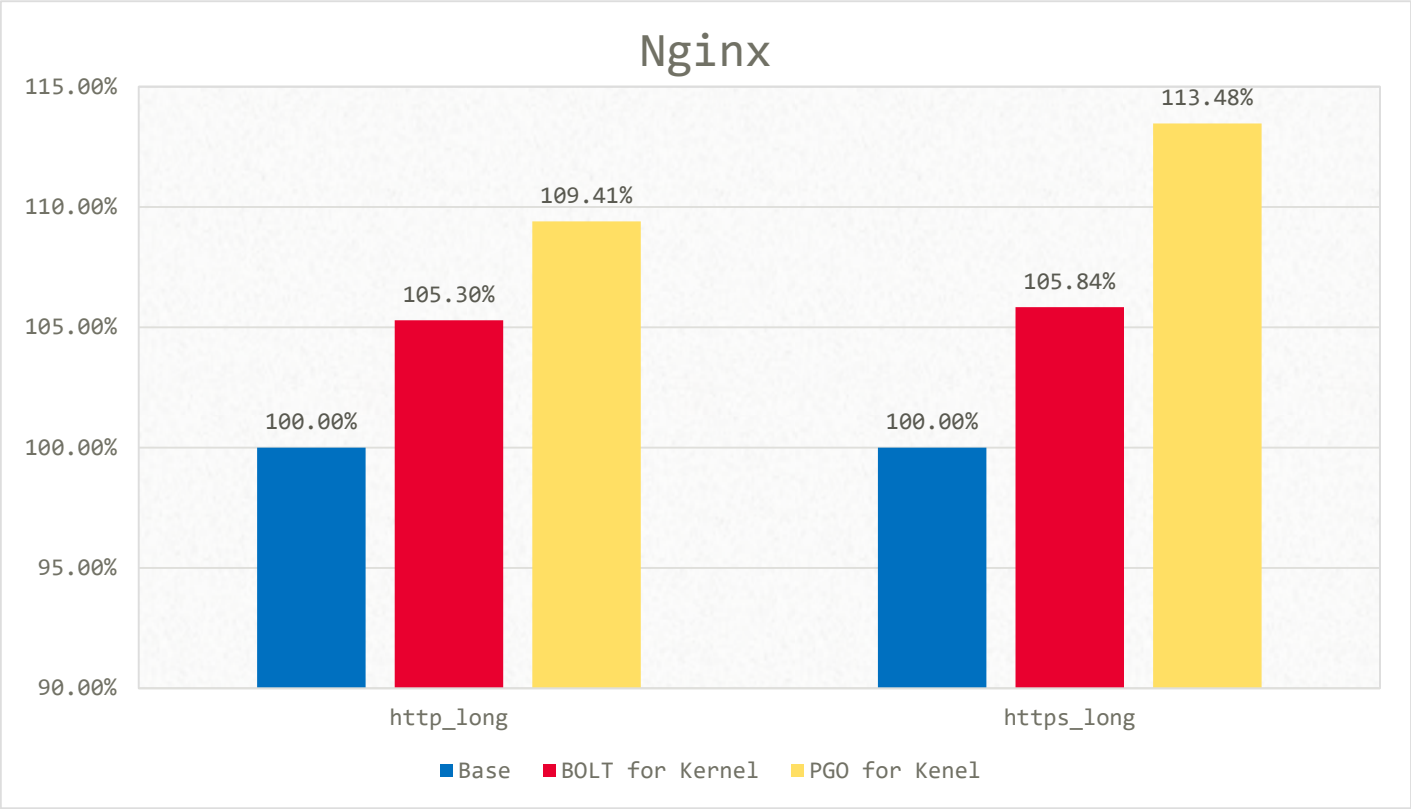
example

- **What are alternative instructions?**

  E.g.
  - Replace BB0,BB1 of instr_offset with BB0 of repl_offset at kernel runtime.
- **Problem:**
  - Bolt reorder/split may change instruction length, so alternative instructions may fail.
- **Solution:**
  - Record alternative instructions info:

    Assembly {offset, length} <-> BinaryBasicBlock {address, length}
  - Trace and update offset and length of alternative instructions.

Results



- **CPU:** Kunpeng 920 (ARMv8.2)
- **OS:** openEuler 24.03 LTS (Kernel 6.6 version)
- **Compiler:** BiSheng Compiler (based on LLVM17.0.6)
- **Nginx Version:** 1.21.5
- **Benchmark Tool:** wrk-4.2

HUAWEI

Plans

- Test more benchmarks or workloads
  - > Unixbench/Sysbench/speccpu… …
  - > MySQL/Redis/Nginx/RocksDB… …
- Compatible with and collaborate with other tools
  - > PGO/AutoFDO/Propeller… …
- Support other archs: RISC-V/x86-64/…
- Support more kernel versions: 5.x/6.x…

Thank you.

把数字世界带入每个人、每个家庭、
每个组织，构建万物互联的智能世界。

Bring digital to every person, home and organization for a fully connected, intelligent world.

HUAWEI