Challenges Of Enabling Golang Binaries Optimization By BOLT

Vasily Leonenko <vasily.leonenko@huawei.com>
Vladislav Khmelevsky <vladislav.khmelevskyi@huawei.com>

Advanced Software Technology Lab, Huawei

ADVANCED SOFTWARE TECHNOLOGY LAB

HUAWEI

Acknowledgement

- Major Contributor: Vladislav Khmelevsky

Contents

Golang Specifics

- Golang (aka Go) is a statically typed, compiled programming language
- The major toolchain implementation is a self-hosted Golang Compiler (github.com/golang/go), it doesn't use the LLVM framework for its implementation
- Supports a list of target operating systems, including Linux, Android, Windows, etc.
- Supports a list of target platforms, including AMD64, ARM64, MIPS64, etc.
- Uses it’s own runtime which operates with compiler/runtime version-specific metadata to implement language-specific functionality like Garbage Collector, Scheduler, etc.
- By default - project source code, all imported packages and the whole runtime library are built into a single statically linked executable

Why BOLT?

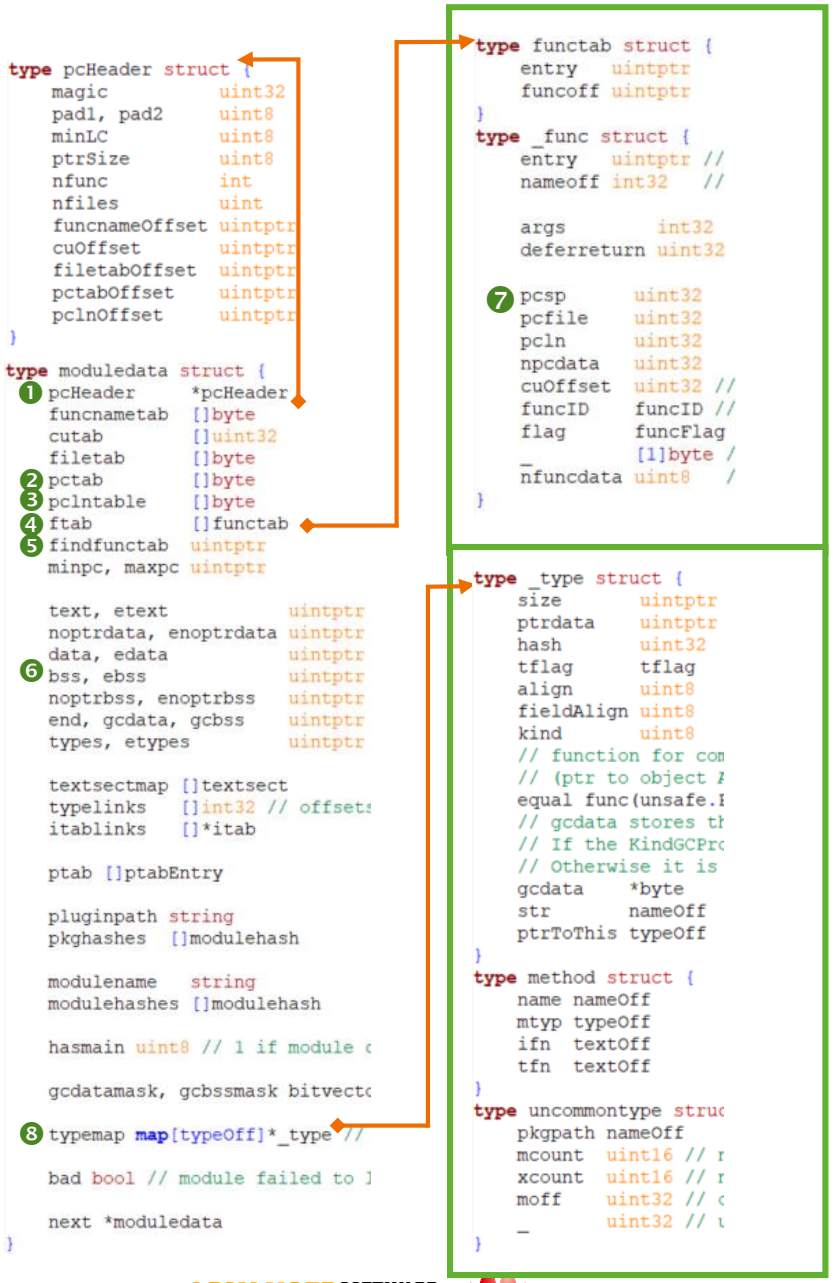
- Golang Compiler still doesn’t support profile-guided optimization
- Output binaries are huge (compared to C/C++ project executables) -> i-cache locality issues
  - E.g. K8s kubelet executable .text section size is ~50M

- BOLT is known as an efficient tool to improve i-cache utilization and reduce branch miss-prediction
- BOLT optimization doesn’t require rebuilding application with a specific compiler

Golang Runtime Data Structures

Golang Runtime metadata includes the following most important structures (actual for Go 1.17):

- ❶ moduledata – The main structure in Golang executable. It records information about the layout of the binary file.
- ❷ pctab – Holds all deduplicated pcdata (used in ❹)
- ❸ pclntable – Header + array of pairs of **function address** and offset in ftab table for each function sorted by address
- ❹ ftab – Array of **function** descriptor structures with glued pcdata & funcdata table reference. Each function descriptor contains information about **address**, name, arguments, **size**, pcsp table offset, number of entries in pcdata & funcdata tables.
  - > pcdata – up to 3 varint-encoded pairs [Value, **PC**]. Types: UnsafePoint – used by scheduler, StackMapIndex – index for stack-related funcdata, InlTreeIndex – index for inline related funcdata.
  - > funcdata – up to 7 pointers to special structures. Types: ArgsPointerMaps, LocalPointerMaps, RegPointerMaps, StackObjects - connects **PC** with stack-related info, required for Garbage Collector and Scheduler work. InlTree type – array of offsets pointing start of inlined function. OpenCodedDeferInfo type – used to store max defers arguments size.

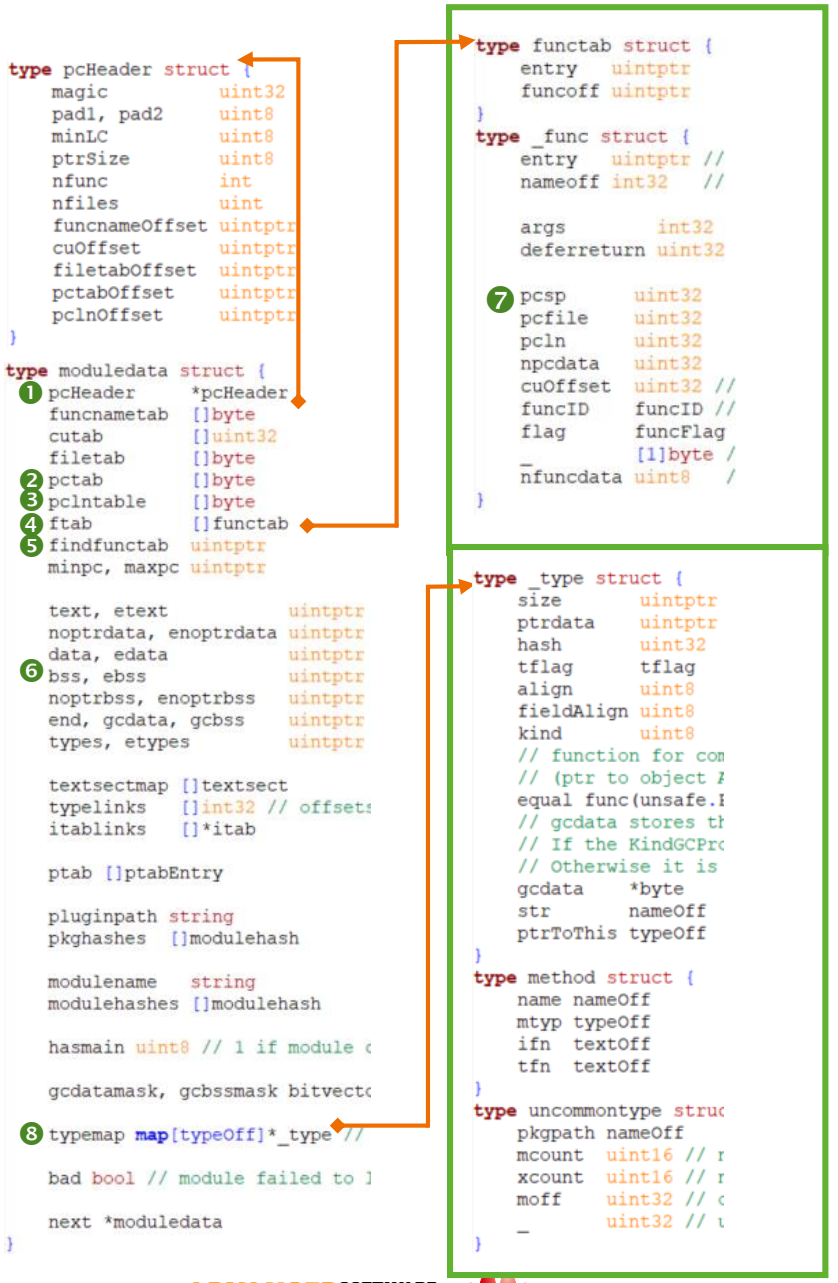


ADVANCED SOFTWARE TECHNOLOGY LAB HUAWEI

Golang Runtime Data Structures

- ❺ findfunctab – Service table used to speedup search of a function in ftab table by **PC** value
- ❻ Pointers to file sections (text, data, bss, etc.)
- ❼ pcsp – **Program Counter** to Stack Pointer table offset. It's used for stacktrace resolving.
- ❽ type descriptors – set of glued structures which may include an array of functions/methods referenced using **offsets** from Golang text start to function entry point

- Data structures mentioned above will be broken after BOLT will finish execution of optimization passes, so offsets and addresses of these data structures in output binary should be updated

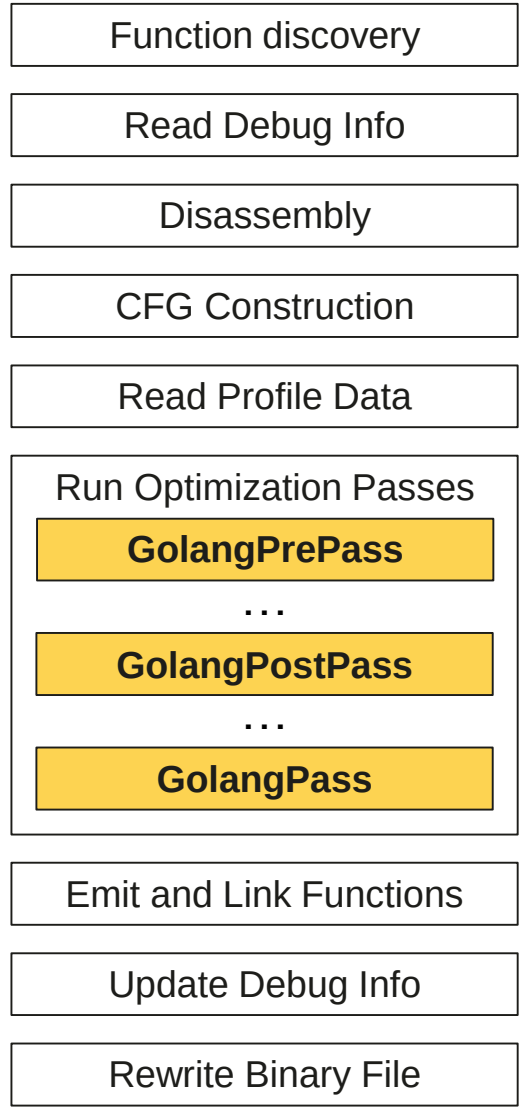


ADVANCED SOFTWARE TECHNOLOGY LAB HUAWEI

Enabling Support in BOLT

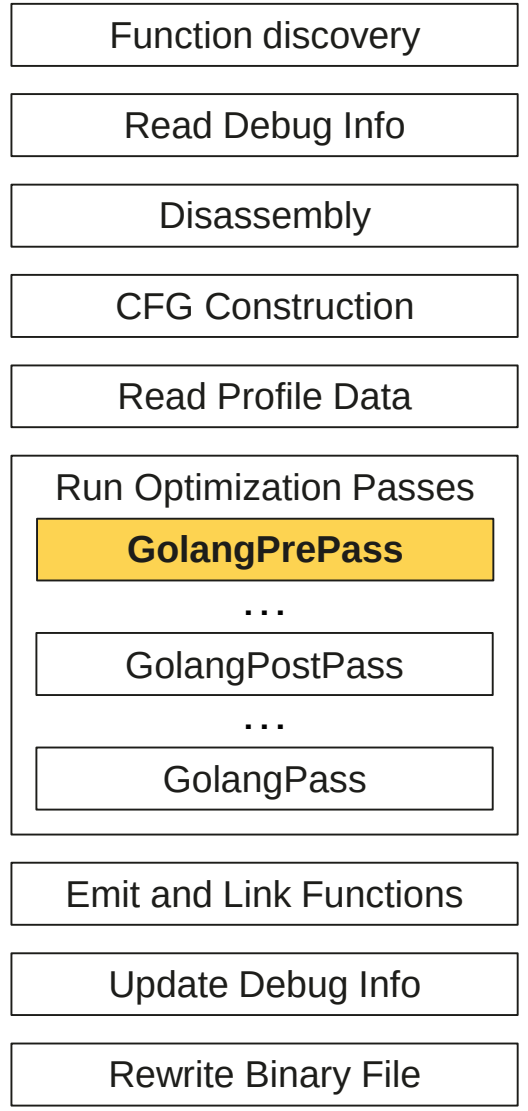
We added three Golang passes to Optimization phase to handle Golang specifics:

- GolangPrePass: Preprocessing stage, runs right after the binary file was disassembled and no changes applied yet
- GolangPostPass: Postprocessing stage, must be the latest pass that changes text
- GolangPass: The very last pass, fixes data section and does not change text



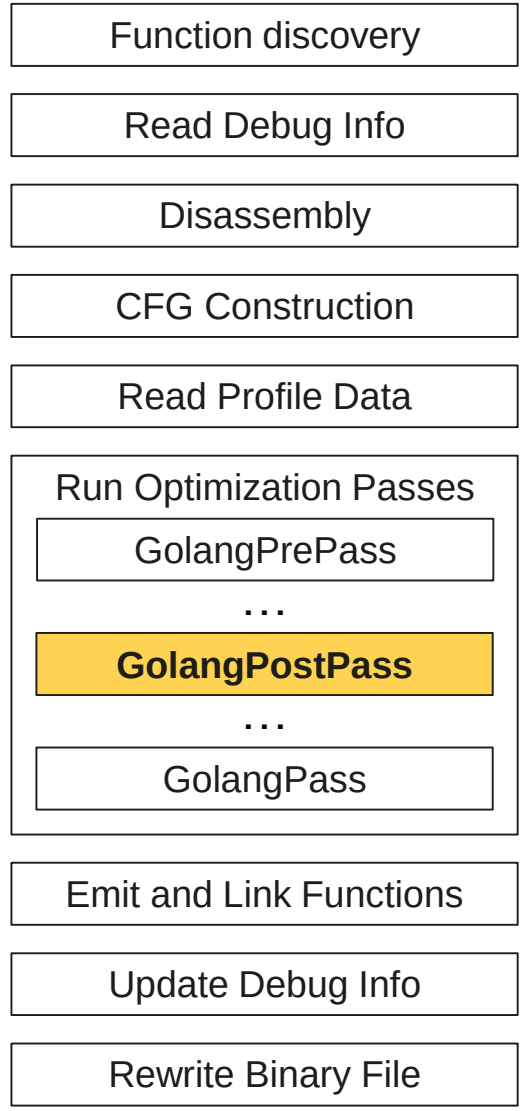
Enabling Support in BOLT

- GolangPrePass: Preprocessing stage
- Runs right after the binary file was disassembled and no changes applied yet
  - > For every function from **pclntable** - save offset in **ftab** for each golang function to extra field of BinaryFunction
  - > For every BinaryFunction:
    - Mark as non-simple if the function has non-standard ID or from the exclusion list (special asm-written functions, that are dangerous to change)
    - Save values of **pcdata** tables in corresponding MCInst (using MCAnnotation)
    - For StackMapIndex **pcdata** additionally save the next instruction to restore table properly
    - Mark deferreturn call instructions (using MCAnnotation with IsDeffer name)
    - Store **pcsp** table conditionally (using MCAnnotation)
    - For every InlTree **funcdata** - store inline index to the first inline caller instruction for each of the inlined functions (using MCAnnotation with FUNCDATA* names)



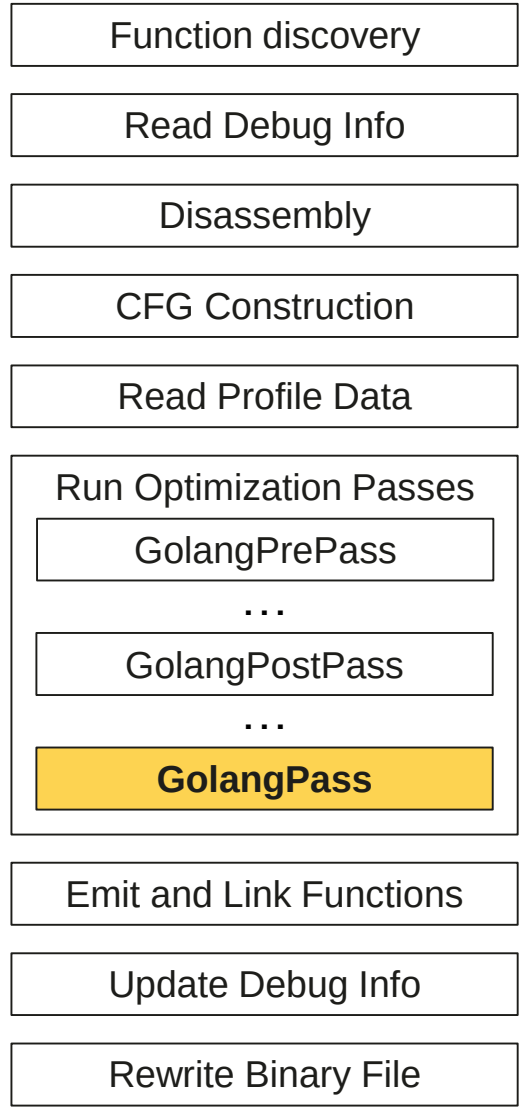
Enabling Support in BOLT

- GolangPostPass: Postprocessing stage
- Must be the latest pass that changes text.
- Also, it used for instrumentation support enabling.
  - > Inserts instrumentation dump() call in runtime.exit function
  - > Restores NOPs padding for some special runtime functions (runtime.skipPleaseUseCallersFrames)
  - > Fixes **pcdata** tables:
    - UnsafePoint table: handles extra instrumentation snippet instructions
    - StackMapIndex table: During preprocessing stage we saved pcdata value to the next instruction as MCAnnotation. If “next” instruction was modified by preceding BOLT passes - we need to insert NOP instruction with added MCAnnotation with correct pcdata to restore it correctly on next stage.



Enabling Support in BOLT

- GolangPass: Final stage
- The very last pass. Fixes data section and does not change text
  - > Fixes offsets of functions/methods of **type descriptors**
  - > Creates a new **pclntable** and **ftab** tables
  - > Restores **pcdata** & **funcdata** tables: inline funcdata, deferreturn call, **pcsp** table
  - > Creates a new **findfunctab** table
  - > Fixes pointers in **firstmoduledata** structure



Status

- Supports Go Compiler versions 1.14, 1.16, 1.17, passes 100% Golang Runtime functional tests
- Supports x86_64 & ARM64 binaries
- Supports Instrumentation for two platforms: x86_64 and ARM64
- Minor changes required for Golang support were merged to BOLT
- Published RFC: https://reviews.llvm.org/D124347
  - This patch is quite big and requires splitting into a series of patches

Performance Impact

- Up to **19%** of relative performance improvement on internal applications
- goweb “Light weight web framework based on net/http”
  - Repo: https://github.com/twharmon/goweb.git
  - Profile collected using BOLT instrumentation
  - Go 1.17
  - .text size ~3.5M
  - Performance Improvement (Xeon Gold 6230N): +**8.13%**
  - Performance Improvement (Kunpeng 920): **+11.74%**

```
name                 old time/op  new time/op  delta
GowebPlaintext-8     2.12µs ± 0%  1.89µs ± 0%  -10.54%  (p=0.008 n=5+5)
GinPlaintext-8       1.40µs ± 1%  1.31µs ± 2%   -6.36%  (p=0.008 n=5+5)
GorillaPlaintext-8   3.19µs ± 0%  3.09µs ± 0%   -3.25%  (p=0.008 n=5+5)
EchoPlaintext-8      1.39µs ± 0%  1.29µs ± 0%   -7.39%  (p=0.008 n=5+5)
MartiniPlaintext-8   17.9µs ± 0%  15.9µs ± 0%  -11.04%  (p=0.008 n=5+5)
GowebJSON-8           119µs ± 0%    99µs ± 0%  -16.81%  (p=0.008 n=5+5)
GinJSON-8             130µs ± 0%   110µs ± 0%  -15.61%  (p=0.008 n=5+5)
GorillaJSON-8         121µs ± 0%   101µs ± 0%  -16.26%  (p=0.008 n=5+5)
EchoJSON-8            117µs ± 0%    98µs ± 0%  -16.11%  (p=0.008 n=5+5)
MartiniJSON-8         169µs ± 0%   143µs ± 0%  -15.23%  (p=0.008 n=5+5)
GowebPathParams-8    5.97µs ± 0%  5.26µs ± 0%  -11.84%  (p=0.008 n=5+5)
GinPathParams-8      4.07µs ± 0%  3.67µs ± 0%   -9.81%  (p=0.008 n=5+5)
GorillaPathParams-8  7.18µs ± 0%  6.40µs ± 0%  -10.77%  (p=0.008 n=5+5)
EchoPathParams-8     4.24µs ± 0%  3.74µs ± 0%  -11.76%  (p=0.008 n=5+5)
MartiniPathParams-8  20.3µs ± 0%  17.9µs ± 0%  -12.09%  (p=0.008 n=5+5)
[Geo mean]           13.8µs       12.2µs       -11.74%
```

- benchmark of graphql frameworks
  - Repo: https://github.com/appleboy/golang-graphql-benchmark.git
  - Profile collected using BOLT instrumentation
  - Go 1.17
  - .text size ~6M
  - Performance Improvement (Xeon Gold 6230N): +**11.36%**
  - Performance Improvement (Kunpeng 920): +**8.98%**

```
name                      old time/op    new time/op    delta
GinHttpRoute-8              1.92µs ± 0%    1.70µs ± 0%  -11.50%  (p=0.008 n=5+5)
GinGQLGenRoute-8            1.95µs ± 0%    1.76µs ± 0%   -9.84%  (p=0.008 n=5+5)
GinGoGraphQLRoute-8         22.5µs ± 0%    19.3µs ± 0%  -14.09%  (p=0.008 n=5+5)
GinGopherGraphQLRoute-8      754ns ± 0%     686ns ± 1%   -9.01%  (p=0.008 n=5+5)
GinThunderGraphQLRoute-8    1.30µs ± 0%    1.16µs ± 1%  -10.48%  (p=0.008 n=5+5)
GoGraphQLMaster-8           59.4µs ± 0%    51.9µs ± 0%  -12.67%  (p=0.008 n=5+5)
PlaylyfeGraphQLMaster-8     5.18µs ± 0%    4.70µs ± 0%   -9.20%  (p=0.008 n=5+5)
GophersGraphQLMaster-8      4.08µs ± 0%    3.56µs ± 0%  -12.77%  (p=0.008 n=5+5)
ThunderGraphQLMaster-8      2.31µs ± 0%    2.02µs ± 1%  -12.57%  (p=0.008 n=5+5)
[Geo mean]                  3.96µs         3.51µs       -11.36%
```

Known Limitations

- Golang Compiler Linker doesn’t support emitting static relocations (emit-relocs option)
  - Resolved by usage of an external linker
- Golang Compiler doesn’t fully follow ARM64 ELF Specification in context of mapping symbols generation
  - Fixed with patches in Golang Compiler (not yet merged) https://go-review.googlesource.com/c/go/+/343150 (https://github.com/yota9/golang_aarch64_mapping_symbols)

- High memory consumption (we observed up to 80GB memory usage for processing of large binaries)
- Some BOLT optimizations are disabled: Inlining, frame optimizations, hot/cold functions splitting, lite mode, updating debug information

Future Plans

- Continue working on RFC, split it into a series of patches and gradually upstream
- Continue upstreaming of ARM64 ELF Symbols support in Golang Compiler
- Add support of newer Golang Compiler versions

Thank you.