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



Acknowledgement

Major Contributor: Vladislav Khmelevsky



Contents

- 1. Golang Specifics
- 2. Why BOLT?
- 3. Enabling Support in Bolt
- 4. Status
- 5. Performance Impact
- 6. Known Limitations
- 7. Future Plans

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):

- • moduledata The main structure in Golang executable. It records information about the layout of the binary file.
- 2 pctab Holds all deduplicated pcdata (used in 4)
- S pcIntable Header + array of pairs of **function address** and offset in ftab table for each function sorted by address
- Itab 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, InITreeIndex – 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. InITree type – array of offsets pointing start of inlined function. OpenCodedDeferInfo type – used to store max defers arguments size.

	type functab struct {				
pcHeader struct	entry uintptr				
magic uint32	funcoff wintertr				
nadi nadi uinte	luncori urneper				
minIC wint?	time func struct /				
ntrSize uint9	cype _func struct {				
ptiblize unito	entry untiper //				
nfunc int	nameoli intisz //				
nilles uint					
runchameoffset uintptr	args int32				
cuOffset uintptr	deferreturn uint32				
filetabOffset uintptr					
pctabOffset uintptr	7 pcsp uint32				
pclnOffset uintptr	pcfile uint32				
	pcln uint32				
moduledata struct (npcdata uint32				
ncHeader *ncHeader	cuOffset uint32 //				
funchametab []byte	funcID funcID //				
cutab []wint32	flag funcFlag				
filetah []hute	[1]byte /				
nctah []byte	nfuncdata uint8 /				
polatable []byte	1				
ftab []functab	15 C				
findfunctab wintett					
minne maxne wintett					
minpe, maxpe mineper	type type struct {				
text stext wintets	size uintptr				
nontrdata anontrdata wintetr	ptrdata uintptr				
data edata wintett	hash uint32				
hee abee winter	tflag tflag				
poptrbes epoptrbes wintptr	align uint8				
and acdata aches wintett	fieldAlign uint8				
times atimes wintetr	kind uint8				
cibes, colbes arucher	// function for com				
tavteactman []tavteact	// (ptr to object #				
typelinks [lint32 // offsets	equal func(unsafe.E				
itablinks []titab	<pre>// gcdata stores th</pre>				
ICADITIKS [] ICAD	// If the KindGCPro				
ntah []ntahEntry	// Otherwise it is				
peap []peapenery	gcdata *byte				
plugippath string	str nameOff				
prographen String	ptrToThis typeOff				
pronasnes []modulenasn	}				
modulename string	type method struct {				
modulehame String	name_nameOff				
modulenasnes []modulenasn	mtyp typeOff				
harmain wint? // 1 if module /	ifn_textOff				
nasmain uinco // I II module (tfn_textOff				
and the set of the bit of the	LIN CONCOLL				
gcdatamask, gcbssmask bitvectc	time uncommonture struc				
	pkmath nameOff				
cypemap map[cypeorI]*_type'//	propact nameOII				
had been 17 meaning printing are t	mcount wint16 // r				
bad boot // module lailed to]	xcount uintio // I				
	moli uint32 // (
next moduledata	uint32 // t				

type

6

8

Golang Runtime Data Structures

- Indfunctab Service table used to speedup search of a function in ftab table by PC value
- 6 Pointers to file sections (text, data, bss, etc.)
- pcsp Program Counter to Stack Pointer table offset. It's used for stacktrace resolving.
- Image: Image: 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

	time functab struct /
me nelleader struct	entry wintett
magic wint32	funcoff wintertr
magic uintsz	tuncori urneper
minIC wint?	tune func struct (
minic units	cype _func struct (
ptrsize uinto	entry untper //
nfunc Int	nameoII 1nt32 //
nfiles unt	Antonio and a second second
funchameOffset uintptr	args int32
cuOffset uintptr	deferreturn uint32
filetabOffset uintptr	
pctabOffset uintptr	7 pcsp uint32
pclnOffset uintptr	pcfile uint32
	pcln uint32
me moduledata struct (npcdata uint32
newarden thewarden	cuOffset uint32 //
functional function	funcID funcID //
runchametab []byte	flag funcFlag
filatel (lbuta	[1]byte /
2 petab []byte	Infuncuada uffico /
for the second s	1
4 Itab []Iunctab	
5 findfunctab uintptr	
minpe, maxpe unitper	type type struct {
	size uintptr
text, etext uintptr	ptrdata uintptr
noptroata, enoptroata uintptr	hash uint32
data, edata uintptr	tflag tflag
Obss, ebss uintptr	align uint8
noptross, enoptross uintptr	fieldAlign uint8
end, gcdata, gcbss uintptr	kind uint8
types, etypes uintptr	// function for com
	// (ptr to object #
textsectmap []textsect	equal func (unsafe.)
typelinks []int32 // offsets	// gcdata stores th
Itablinks []*Itab	// If the KindGCPro
	// Otherwise it is
ptab []ptabEntry	acdata *byte
Service and the service of the servi	str nameOff
pluginpath string	ntrToThis typeOff
pkghashes []modulehash)
	time method struct (
modulename string	cype method Struct (
modulenasnes []modulenasn	mtum tumoOff
A REPORT OF A REPORT OF A REPORT OF A	if to to to to to
hasmain uint8 // 1 if module (iin textoii
	tin textoii
gcdatamask, gcbssmask bitvecto	1
	type uncommontype struc
<pre>typemap map[typeOff]*_type*//</pre>	prgpath nameOff
	mcount uint16 // r
bad bool // module failed to]	xcount uint16 // r
	moff uint32 // c
next *moduledata	uint32 // 1
	}
A DVA NCCDSOFTWARF	

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

Function discovery		
Read Debug Info		
Disassembly		
CFG Construction		
Read Profile Data		
Run Optimization Passes		
GolangPrePass		
GolangPostPass		
GolangPass		
Emit and Link Functions		
Update Debug Info		



- GolangPrePass: Preprocessing stage
- Runs right after the binary file was disassembled and no changes applied yet
 - > For every function from pcIntable 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)

Function discovery

Read Debug Info

Disassembly

CFG Construction

Read Profile Data

Run Optimization Passes		
GolangPrePass		
GolangPostPass		
GolangPass		







- 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.

Function discovery Read Debug Info Disassembly **CFG** Construction **Read Profile Data Run Optimization Passes GolangPrePass** . . . **GolangPostPass** . . . GolangPass Emit and Link Functions

Update Debug Info



- 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 **pcIntable** and **ftab** tables
 - > Restores **pcdata** & **funcdata** tables: inline funcdata, deferreturn call, **pcsp** table
 - > Creates a new findfunctab table
 - > Fixes pointers in firstmoduledata structure

Function discovery

Read Debug Info

Disassembly

CFG Construction

Read Profile Data

Run Optimization Passes GolangPrePass ... GolangPostPass ... GolangPass

Emit and Link Functions

Update Debug Info



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: <u>https://github.com/twharmon/goweb.git</u>
 - > 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%
- benchmark of graphql frameworks
 - > Repo: <u>https://github.com/appleboy/golang-graphql-benchmark.git</u>
 - > 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	
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%	

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) <u>https://go-review.googlesource.com/c/go/+/343150</u> (<u>https://github.com/yota9/golang_aarch64_mapping_symbols</u>)
- 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.

