ROG

高性能 Go 实现

演讲人：刘翼飞

目录 | Contents

01 ROG 之缘起

眼红 Rust 性能

业务迁移到 Rust 的显著受益

- 业务 A（Proxy 类）：
  - CPU：−39.68%
  - MEM：−77.78%
  - P99：−76.67%
  - AVG：−70%
- 业务 B（有大量业务逻辑）：
  - CPU：−43.75%
  - MEM：−69%
  - P99：−43.22%
  - AVG：−47.81

写 Rust 太难了

让我们的业务方去写 Rust 并不是一件简单的事儿

1. Rust 生命周期太复杂了
2. 泛型系统太复杂了
3. 报错看不懂

那么有没有什么办法可以让我们的用户不写 Rust 仍然有性能提升呢?

所以 ROG 出现了

我们有了个大胆的想法：

1. 使用像 Rust 那样的编译技术生成性能更好的可执行文件
2. 使用 Rust 重写 Go 的 Runtime 和 GC
3. 提供几乎零开销的 FFI 方案来支持 Rust 和 Go 的互调

02 ROG 进展

一些纯计算场景

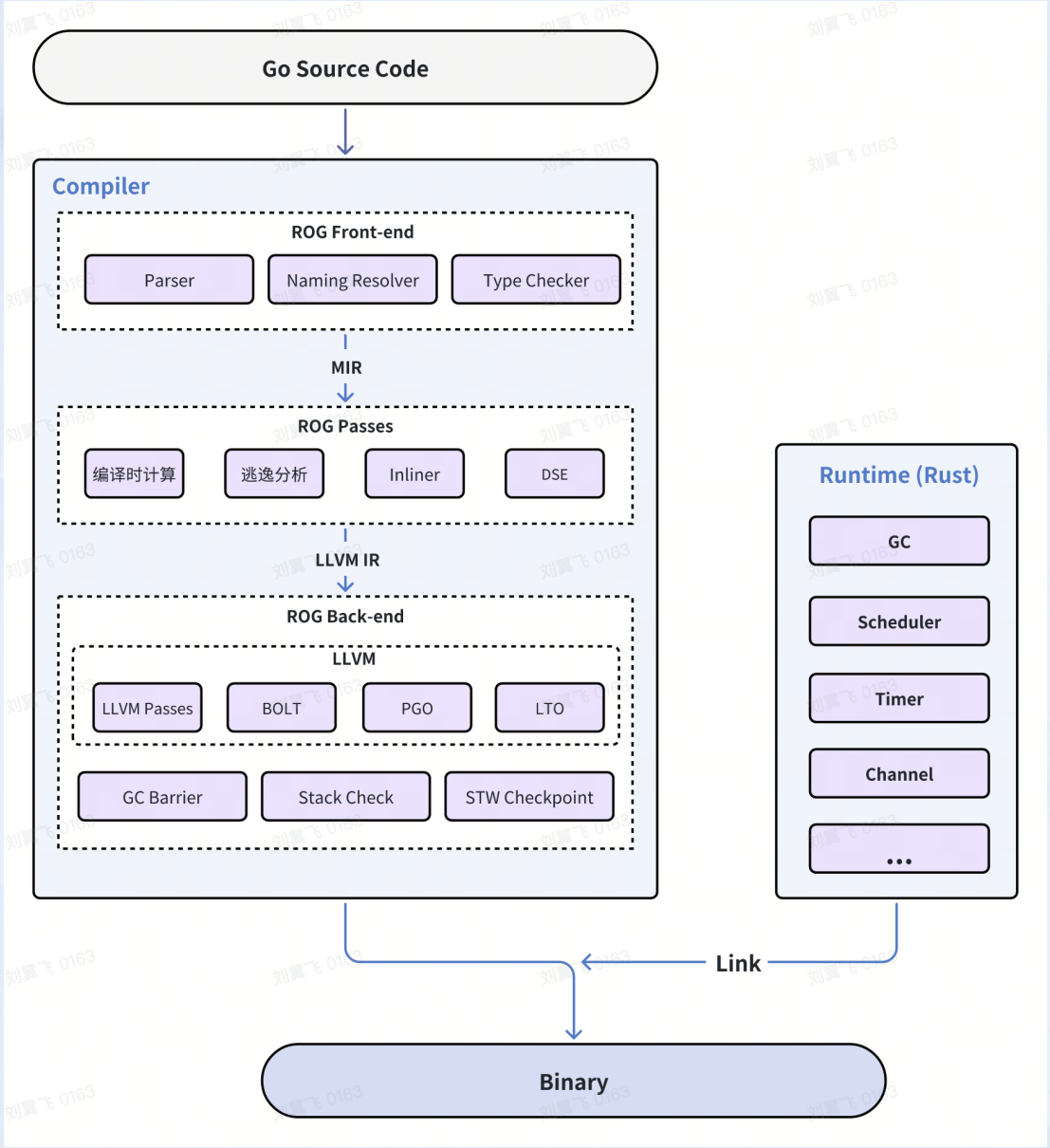
	快排 + 二分	Simple Lisp	Hashmap
Go	5.97s	8.17s	3.66s
ROG	4.12s (−30%)	7.09s (−14%)	1.43s(−60%)

复杂的微服务场景 Kitex Benchmark

https://github.com/cloudwego/kitex-benchmark

	QPS	Latency(p99)
Go	27W	0.76ms
ROG	28W	0.52ms(−32%)

ROG 架构


Go Source Code
Compiler
ROG Front-end
Parser
Naming Resolver
Type Checker
MIR
ROG Passes
编译时计算
逃逸分析
Inliner
DSE
LLVM IR
ROG Back-end
LLVM
LLVM Passes
BOLT
PGO
LTO
GC Barrier
Stack Check
STW Checkpoint
Runtime (Rust)
GC
Scheduler
Timer
Channel
...
Link
Binary

收益来源

	ROG	Go
编译优化	利用了 LLVM 积累多年的编译优化算法，能够生成性能更好的代码	完全自研，为了编译速度作出一些牺牲
FFI + 跨语言 LTO	可以几乎零开销的调用 Rust 提供的方法，因此在要求更高性能的场景可以使用 Rust 开发，由 ROG 进行编译运行	使用CGO，并且存在一些 overhead
Runtime & GC	使用 Rust 重写，再通过 FFI 提供调用来保证性能	使用 Go 原生实现

03 面临的挑战

TinyGo 现状

1. 手动通过 runtime.Gosched 协作调度
2. 不支持多线程
3. 缺少 channel timer reflect 等 lib 的支持

因为本身不是面向高性能场景，所以以上痛点都不是大问题

ROG 的方式

	ROG	TinyGo
调度方式	编译器插入代码完成协作式调度	手动通过runtime.Goshed 协作调度
多线程	支持	不支持
基础 lib 支持	已经全部完成适配	并发相关不支持，reflect 残缺

Safety FFI

```
func rog_test(a *int32)

func main() {

    var a int32 = 123

    rog_test(&a)
}
```

```
#[no_mangle]
extern "C" fn rog_test(a: *const i32) -> *const (*const i32, i32) {
    let rust_tup: (*const i32, i32) = (a, 12312);

    Box::leak(Box::new(rust_tup))
}
```

绝知此事要躬行

Runtime 调度	插入指令协作式调度
GC	插入 Write Barrier，STW Point
众多不兼容的 pkg	重写 或者 修改标准库自行维护
性能优化	常量传播，inlined defer，跨语言LTO，修改 LLVM

04 | Roadmap

CGO

```go
package main

/*
#include <stdio.h>

void printint(int v) {
    printf("printint: %d\n", v);
}
*/
regenerate cgo definitions
import "C"

func main() {
    v := 42
    C.printint(C.int(v))
}
```

```go
package main

//rog:linkname printint
func printint(a int)

func main() {
    v := 42
    printint(v)
}
```

CGO = FFI + Bindgen

编译期生成代码

Go 和 Rust JSON 序列化方案

Go 标准库	Go Sonic	Rust Serde
使用反射，存在非常的反射开销	使用 JIT 过于黑魔法	在编译时生成序列化和反序列化代码

编译期生成代码

如果 ROG 开放编译时生成代码的能力

	ROG	Go
序列化	不需要反射，编译时生成对应的代码	使用反射或者 JIT 黑魔法
idl 生成 server 或者 client 代码	编译时自动生成，告别手动命令执行	手动调用命令生成

Plan 9

```
11
12  // func addVV(z, x, y []Word) (c Word)
13  TEXT ·addVV(SB),NOSPLIT,$0
14      ADD.S   $0, R0      // clear carry flag
15      MOVW    z+0(FP), R1
16      MOVW    z_len+4(FP), R4
17      MOVW    x+12(FP), R2
18      MOVW    y+24(FP), R3
19      ADD R4<<2, R1, R4
20      B E1
21  L1:
22      MOVW.P  4(R2), R5
23      MOVW.P  4(R3), R6
24      ADC.S   R6, R5
25      MOVW.P  R5, 4(R1)
26  E1:
27      TEQ R1, R4
28      BNE L1
29
30      MOVW    $0, R0
31      MOVW.CS $1, R0
32      MOVW    R0, c+36(FP)
33      RET
34
```

常见于 Go 标准库以及高性能开源库

开源

如果一切顺利的话，最早可能需要

2025 Q2

THANKS