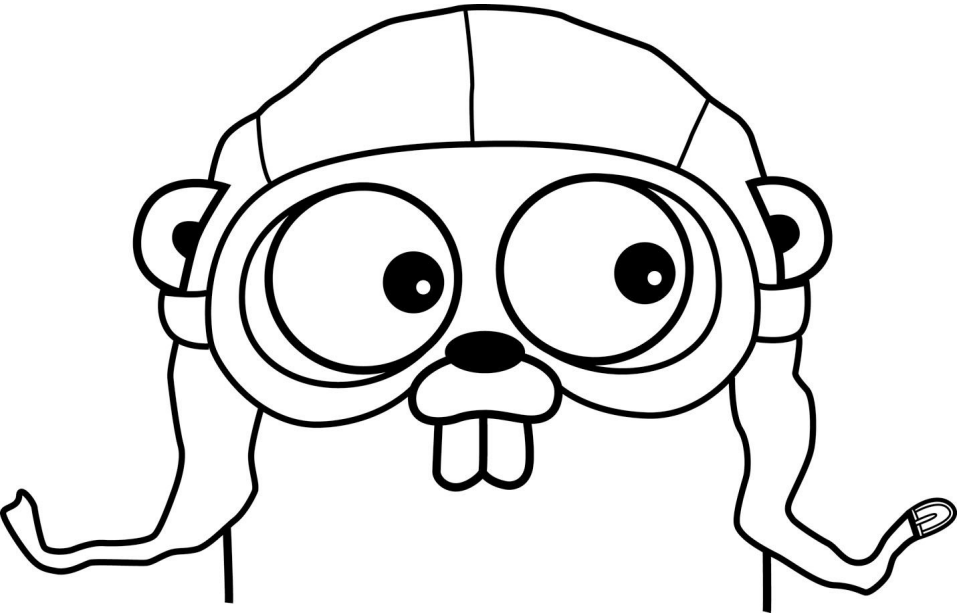
Learn Go by contributing

Часть #3



https://github.com/Quasilyte/kfu-go-2018

Введение в профилирование и оптимизацию

- Сбор метрик: бенчмарки, cpuprofile, распределение входных данных
- Анализ машинного кода
- Оценка влияния/поведения на другие платформы и архитектуры.

Простейший случай

- Нахождение проблемы
- Гипотезы для улучшения
- Проверка гипотезы на бенчмарках
  - Если бенчмарка нет, написать
- Если улучшения нет, пробовать другую гипотезу или искать новое “узкое место”
- Если улучшение есть, отправлять на ревью

Реальность: нужно проверять на регрессии

Относительно недавно в *Go* добавили генерацию CMOV инструкций. На многих бенчмарках было получено ускорение до 30%.

Но, что не было замечено: некоторые паттерны кода стали вплоть до 30% медленнее.

Бенчмарки в Go: код бенчмарка

```
package mypkg

import "testing"

func BenchmarkFoo(b *testing.B) {
   for i := 0; i < b.N; i++ {
      foo(16)
   }
}
```

Бенчмарки в Go: тестируемая функция

```
package mypkg

//go:noinline
func foo(length int) int {
   xs := make([]int, length)
   return len(xs)
}
```

Бенчмарки в Go: запуск

```
$ go test -bench=. -benchmem -cpuprofile=cpu

goos: linux
goarch: amd64
BenchmarkFoo-4    30000000        54.0 ns/op       128 B/op         1 allocs/op
PASS
ok    ~/go-sandbox/src/mypkg   1.676s
```

Бенчмарки в Go: pprof

```
$ go tool pprof cpu

(pprof) top 7

Showing nodes accounting for 970ms, 64.67% of 1500ms total
Showing top 7 nodes out of 106
      flat  flat%   sum%        cum   cum%
     270ms 18.00% 18.00%      990ms 66.00%  runtime.mallocgc
     210ms 14.00% 32.00%      210ms 14.00%  runtime.nextFreeFast (inline)
     190ms 12.67% 44.67%      190ms 12.67%  runtime.memclrNoHeapPointers
      70ms  4.67% 49.33%     1080ms 72.00%  runtime.makeslice
      60ms  4.00% 53.33%     1140ms 76.00% _/home/quasilyte/CODE/go/go_git/mypkg.foo
      40ms  2.67% 56.00%       40ms  2.67%  runtime.releasem (inline)
      40ms  2.67% 58.67%       50ms  3.33%  runtime.scanblock
```

Бенчмарки в Go: оптимизируем вручную

```
package mypkg

//go:noinline
func fooOpt(length int) int {
   // => len(make([]int, length))
   // => length
   return length
}
```

Бенчмарки в Go: добавляем второй бенчмарк

```
// В том же файле, где и BenchmarkFoo.

func BenchmarkFooOpt(b *testing.B) {
   for i := 0; i < b.N; i++ {
      fooOpt(16)
   }
}
```

Бенчмарки в Go: запуск для сравнения

```
$ go test -bench=. -benchmem

goos: linux
goarch: amd64
BenchmarkFoo-4     30000000       53.2 ns/op       128 B/op         1 allocs/op
BenchmarkFooOpt-4  1000000000     2.62 ns/op       0 B/op           0 allocs/op
PASS
ok    _/home/quasilyte/CODE/go/go_git/mypkg    4.545sbox/src/mypkg   1.676s
```

https://godoc.org/golang.org/x/perf/cmd/benchstat

Benchstat tool

Бенчмарки в Go: данные для benchstat

```
$ go test -bench=. -benchmem -count=5 > stats
$ grep 'Foo-' stats > old
$ grep 'FooOpt-' stats > new
$ sed -i 's/FooOpt/Foo/g' new
```

Бенчмарки в Go: запуск для сравнения

```
$ benchstat old new

name   old time/op    new time/op  delta
Foo-4    53.8ns ± 1%  2.6ns ± 2%   -95.10%  (p=0.008 n=5+5)

name   old alloc/op   new alloc/op   delta
Foo-4    128B ± 0%        0B       -100.00%  (p=0.008 n=5+5)

name   old allocs/op  new allocs/op  delta
Foo-4    1.00 ± 0%    0.00     -100.00%  (p=0.008 n=5+5)
```

-gcflags: -S печатает машинный код

```
$ go test -bench=. -gcflags=-S

"".fooOpt STEXT nosplit size=11 args=0x10 locals=0x0
  00000 foo_test.go:12 TEXT "".fooOpt(SB), NOSPLIT, $0-16
  00000 foo_test.go:15 MOVQ "".length+8(SP), AX
  00005 foo_test.go:15 MOVQ AX, "".~r1+16(SP)
  00010 foo_test.go:15 RET
  0x0000 48 8b 44 24 08 48 89 44 24 10 c3              H.D$.H.D$..
```

...Плюс код для остальных функций

-gcflags: -m отладка компилятора

```
$ test -bench=. -benchmem -gcflags='-m=2'

./foo_test.go:6:6: cannot inline foo: marked go:noinline
./foo_test.go:12:6: cannot inline fooOpt: marked go:noinline
```

...Чем больше “-m”, тем больше отладочной информации будет напечатано.

-gcflags: -m отладка компилятора

```
$ test -bench=. -benchmem -gcflags='-m=2'

./foo_test.go:5:6: can inline foo as: func(int) int { … }
./foo_test.go:10:6: can inline fooOpt as: func(int) int { return length }
```

Теперь компилятор встраивает тела обеих функций.

Удаляем директиву “noinline”

```
//go:noinline
func fooOpt(length int) int {
   return length
}
```

Бенчмарки в Go: запуск для сравнения

```
$ go test -bench=. -benchmem

goos: linux
goarch: amd64
BenchmarkFoo-4          30000000         52.0 ns/op
BenchmarkFooOpt-4       2000000000       0.38 ns/op
PASS
ok    _/home/quasilyte/CODE/go/go_git/mypkg    2.431s
```

Для fooOpt ускорение значительное.

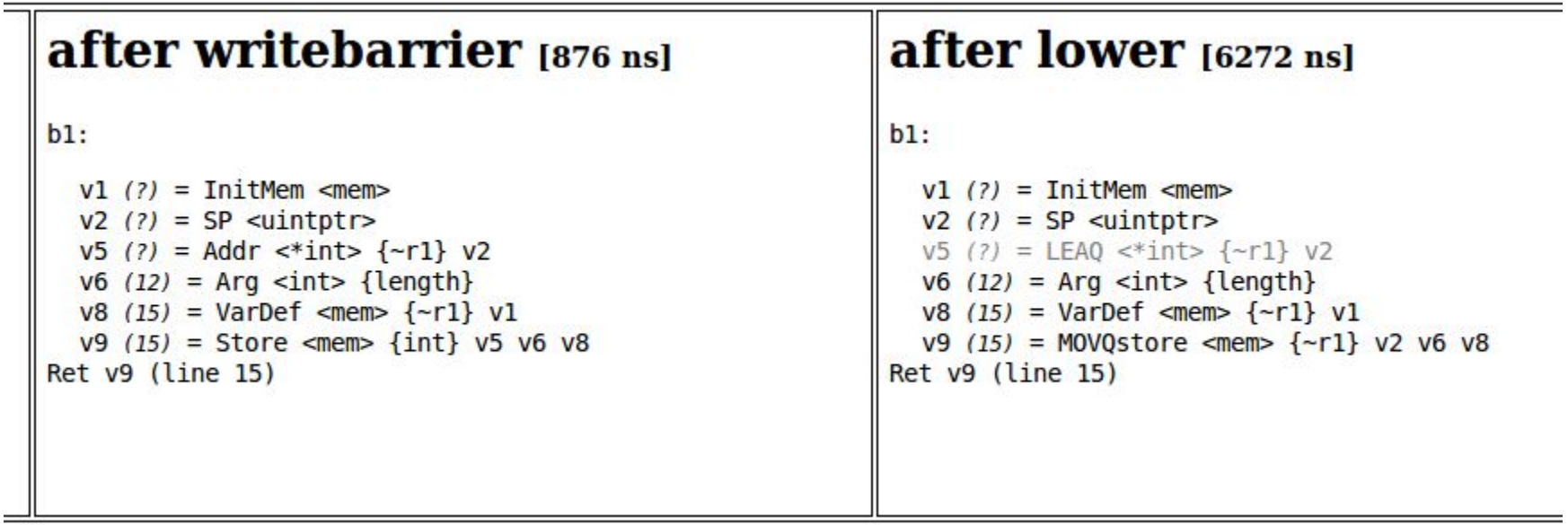
До встраивания было 2.6 ns/op.

GOSSAFUNC: печатает SSA представление

```
$ GOSSAFUNC=fooOpt go test -bench=.

    pass trim begin
    pass trim end [640 ns]
fooOpt func(int) int
  B1:
    v1 = InitMem <mem>
    v8 = VarDef <mem> {~r1} v1
    v2 = SP <uintptr> : SP
    v6 = Arg <int> {length} : length[int]
    v5 = LoadReg <int> v6 : AX
    v9 = MOVQstore <mem> {~r1} v2 v5 v8
    Ret v9
```

ssa.html после GOSSAFUNC



after writebarrier [876 ns]

```
b1:

  v1 (?) = InitMem <mem>
  v2 (?) = SP <uintptr>
  v5 (?) = Addr <*int> {~r1} v2
  v6 (12) = Arg <int> {length}
  v8 (15) = VarDef <mem> {~r1} v1
  v9 (15) = Store <mem> {int} v5 v6 v8
Ret v9 (line 15)
```

after lower [6272 ns]

```
b1:

  v1 (?) = InitMem <mem>
  v2 (?) = SP <uintptr>
  v5 (?) = LEAQ <*int> {~r1} v2
  v6 (12) = Arg <int> {length}
  v8 (15) = VarDef <mem> {~r1} v1
  v9 (15) = MOVQstore <mem> {~r1} v2 v6 v8
Ret v9 (line 15)
```

Файл ssa.html создаётся в директории, где была запущена go команда вместе с GOSSAFUNC.