

Squeezing a Go function

Jesús Espino - Software Engineer @ Mattermost



What is optimizing?



A photograph showing a muddy, rutted path in a field. The path is deep and filled with brown mud, with several distinct ruts running down its length. The surrounding area is a mix of green grass and patches of snow. In the background, there is a wooden fence and several trees. The overall scene suggests a recent heavy rain or snowmelt.

Optimize at the right level

Optimize what you need when you need it



Do not guess



Measure everything



Benchmarks



Benchmarks

```
package main

import (
    "crypto/md5"
    "crypto/sha256"
    "testing"
)

const text = "text to get the hash"

func BenchmarkMD5Hash(b *testing.B) {
    for i := 0; i < b.N; i++ {
        _ = md5.Sum([]byte(text))
    }
}

func BenchmarkSHA256Hash(b *testing.B) {
    for i := 0; i < b.N; i++ {
        _ = sha256.Sum256([]byte(text))
    }
}
```



Benchmarks

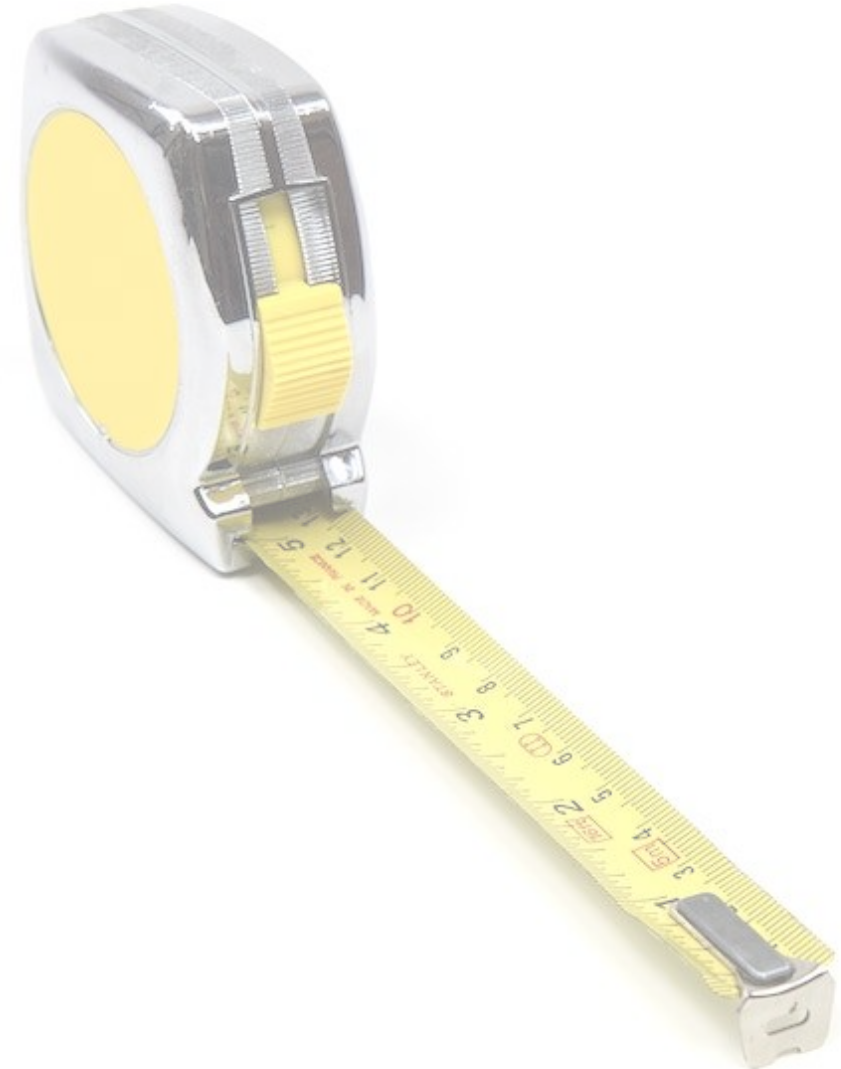
```
$ go test -bench .  
goos: linux  
goarch: amd64  
pkg: example  
cpu: Intel(R) Core(TM) i7-10750H CPU @ 2.60GHz  
BenchmarkMD5Hash-12      10615232      111.5 ns/op  
BenchmarkSHA256Hash-12  5326459      216.0 ns/op  
PASS  
ok      example      2.679s
```



Benchmarks

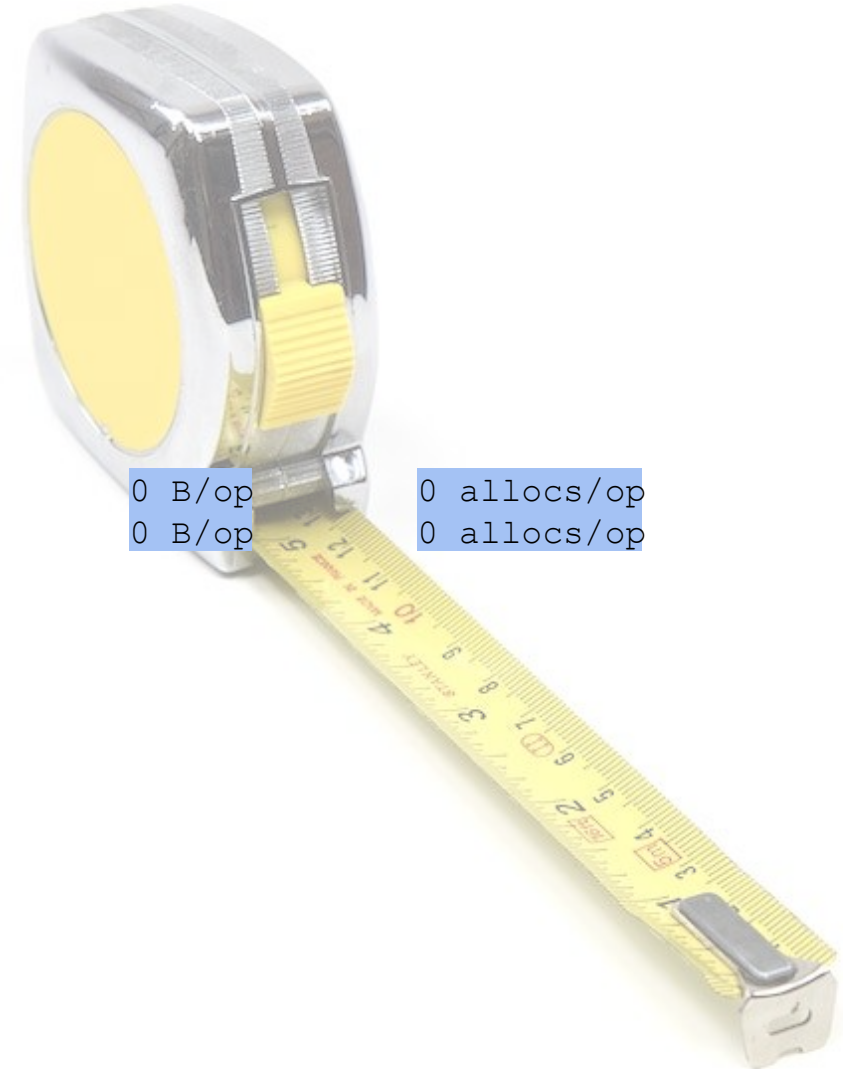
```
func BenchmarkMD5Hash(b *testing.B) {  
    b.ReportAllocs()  
    for i := 0; i < b.N; i++ {  
        _ = md5.Sum([]byte(text))  
    }  
}
```

```
func BenchmarkSHA256Hash(b *testing.B) {  
    b.ReportAllocs()  
    for i := 0; i < b.N; i++ {  
        _ = sha256.Sum256([]byte(text))  
    }  
}
```



Benchmarks

```
$ go test -bench .  
goos: linux  
goarch: amd64  
pkg: example  
cpu: Intel(R) Core(TM) i7-10750H CPU @ 2.60GHz  
BenchmarkMD5Hash-12      10687831      111.5 ns/op      0 B/op      0 allocs/op  
BenchmarkSHA256Hash-12  5717868      210.4 ns/op      0 B/op      0 allocs/op  
PASS  
ok      example      2.725s
```



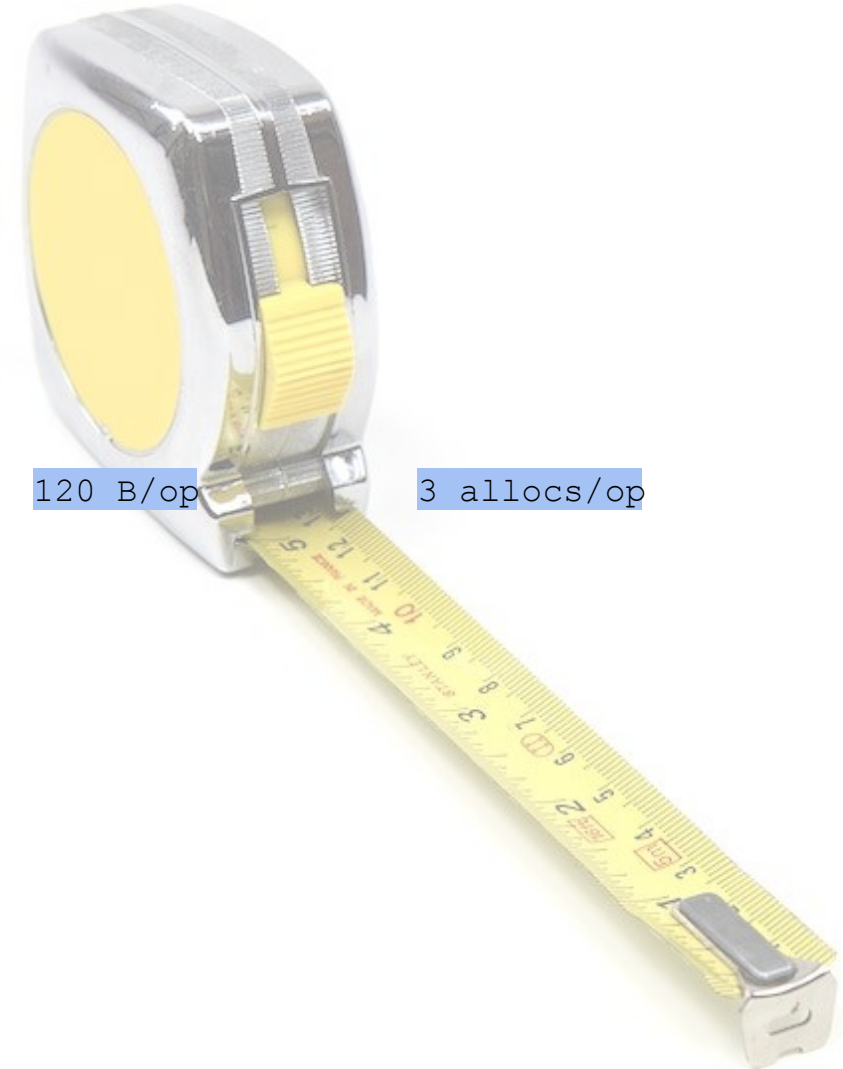
Benchmarks

```
func BenchmarkOsOpen(b *testing.B) {  
    b.ReportAllocs()  
    for i := 0; i < b.N; i++ {  
        _, _ = os.Open("/proc/cpuinfo")  
    }  
}
```



Benchmarks

```
$ go test -bench .  
goos: linux  
goarch: amd64  
pkg: example  
cpu: Intel(R) Core(TM) i7-10750H CPU @ 2.60GHz  
BenchmarkOsOpen-12      387180      3205 ns/op  
PASS  
ok      example    2.479s
```

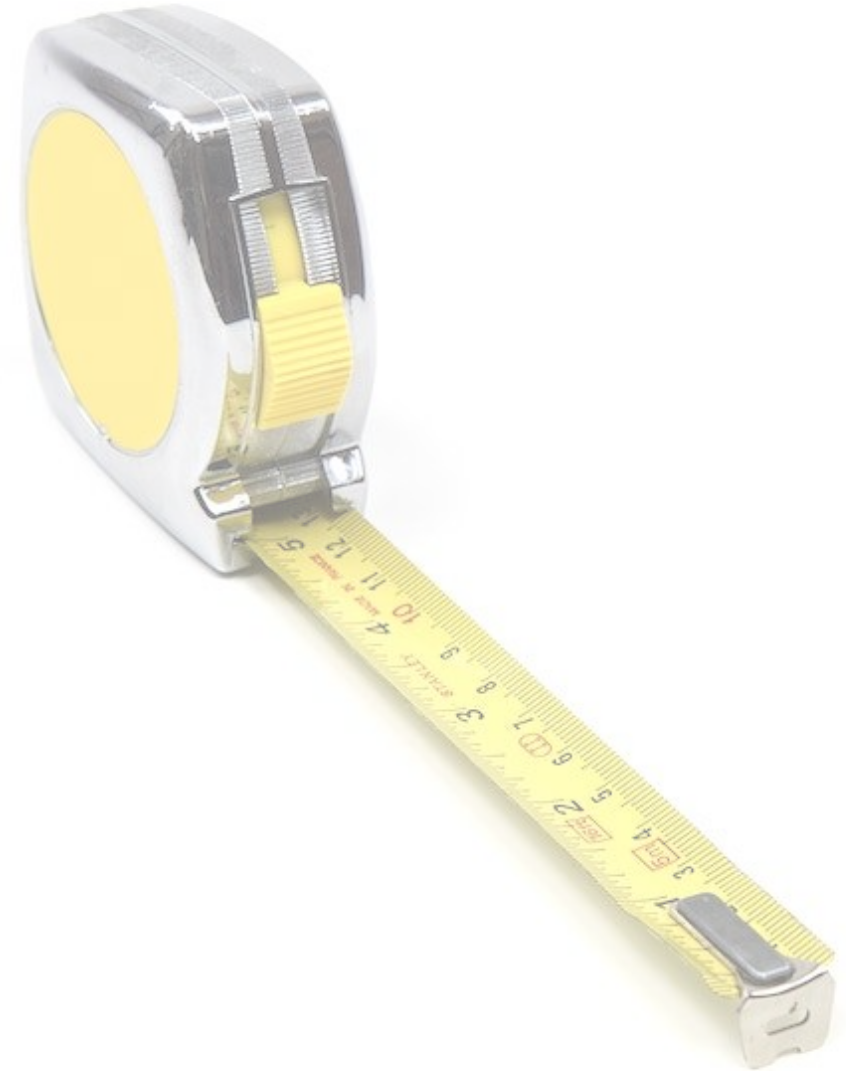


Profiling



Profiling

```
func BenchmarkOsOpen(b *testing.B) {  
    b.ReportAllocs()  
    for i := 0; i < b.N; i++ {  
        _, _ = os.Open("/proc/cpuinfo")  
    }  
}
```



Profiling

```
$ go test -bench . -memprofile memprofile.out
```

```
$ go tool pprof -text memprofile.out
```

```
File: example.test
```

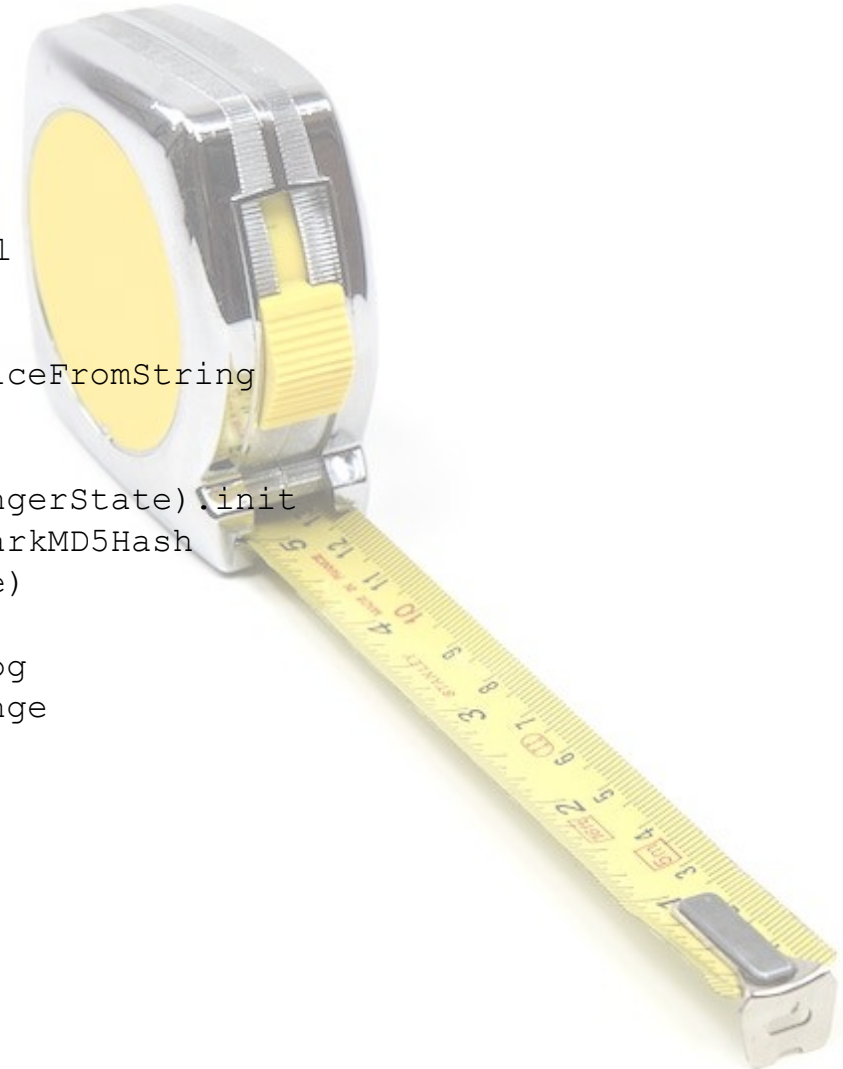
```
Type: alloc_space
```

```
Time: Feb 2, 2023 at 10:20am (CET)
```

```
Showing nodes accounting for 88.01MB, 100% of 88.01MB total
```

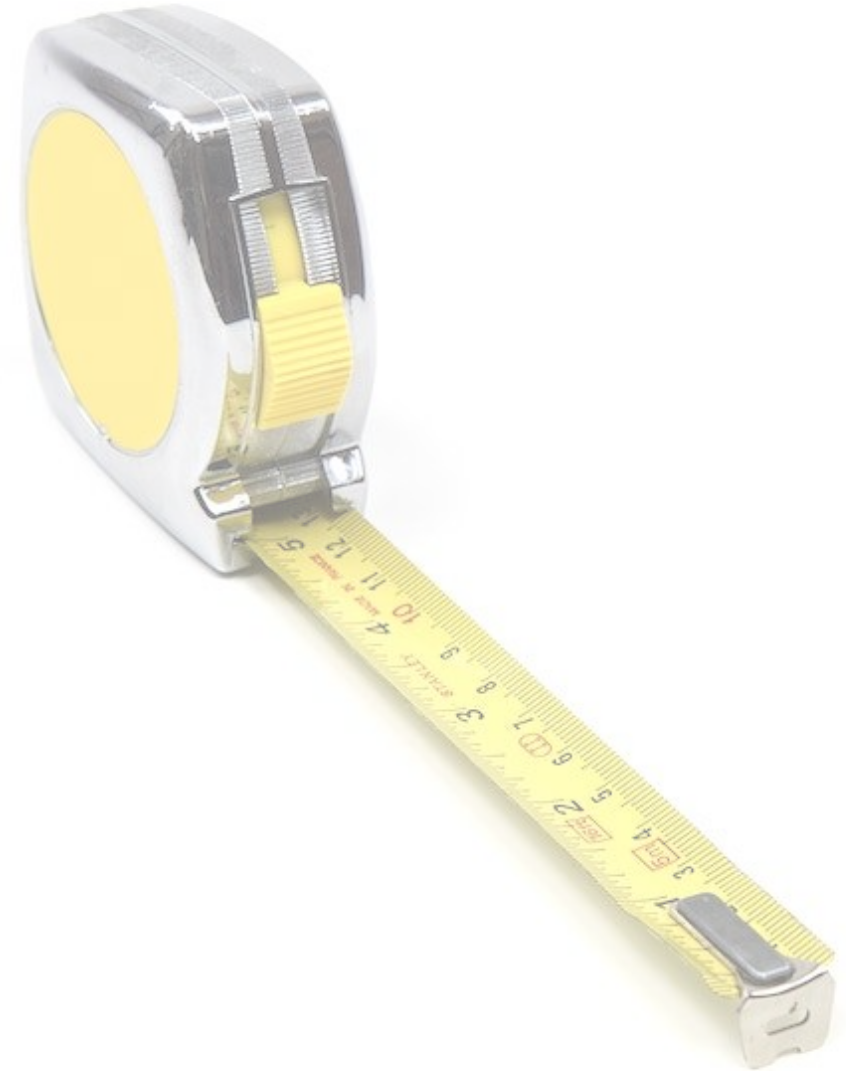
flat	flat%	sum%	cum	cum%	
74.51MB	84.66%	84.66%	74.51MB	84.66%	os.newFile
9.50MB	10.79%	95.45%	9.50MB	10.79%	syscall.ByteSliceFromString
3MB	3.41%	98.86%	3MB	3.41%	runtime.allocm
0.50MB	0.57%	99.43%	0.50MB	0.57%	runtime.malg
0.50MB	0.57%	100%	0.50MB	0.57%	runtime.(*scavengerState).init
0	0%	100%	84.01MB	95.45%	example.BenchmarkMD5Hash
0	0%	100%	84.01MB	95.45%	os.Open (inline)
0	0%	100%	84.01MB	95.45%	os.OpenFile
0	0%	100%	84.01MB	95.45%	os.openFileNolog
0	0%	100%	0.50MB	0.57%	runtime.bgscavenge
0	0%	100%	2.50MB	2.84%	runtime.mcall
0	0%	100%	0.50MB	0.57%	runtime.mstart
0	0%	100%	0.50MB	0.57%	runtime.mstart0
0	0%	100%	0.50MB	0.57%	runtime.mstart1

```
...
```

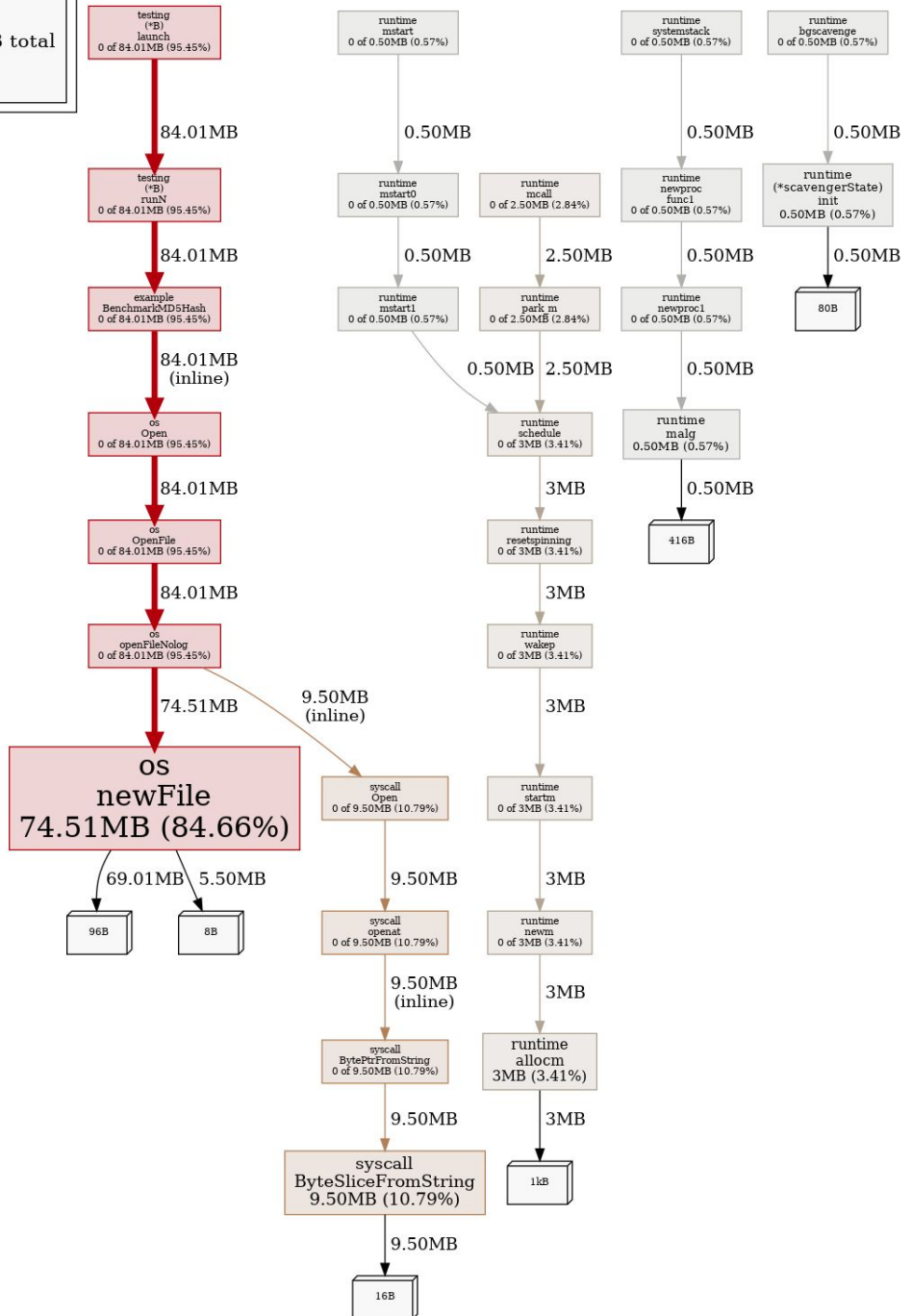


Profiling

```
$ go tool pprof -svg memprofile.out  
Generating report in profile001.svg
```



File: example.test
 Type: alloc_space
 Time: Feb 2, 2023 at 10:20am (CET)
 Showing nodes accounting for 88.01MB, 100% of 88.01MB total
 See <https://git.io/JfYMW> for how to read the graph



Profiling

```
$ go tool pprof -list os.newFile memprofile.out
Total: 88.01MB
ROUTINE ===== os.newFile in /usr/lib/go-1.19/src/os/file_unix.go
74.51MB 74.51MB (flat, cum) 84.66% of Total
.      .      122:func newFile(fd uintptr, name string, kind newFileKind) *File {
.      .      123:     fdi := int(fd)
.      .      124:     if fdi < 0 {
.      .      125:         return nil
.      .      126:     }
74.51MB 74.51MB 127:     f := &File{&file{
.      .      128:         pfd: poll.FD{
.      .      129:             Sysfd:     fdi,
.      .      130:             IsStream:  true,
.      .      131:             ZeroReadIsEOF: true,
.      .      132:         },
```



Profiling

```
$ go test -bench . -cpuprofile cpuprofile.out
$ go tool pprof -text cpuprofile.out
File: example.test
Type: cpu
Time: Feb 2, 2023 at 10:46am (CET)
Duration: 2.30s, Total samples = 4.02s (174.51%)
Showing nodes accounting for 3.83s, 95.27% of 4.02s total
Dropped 33 nodes (cum <= 0.02s)
```

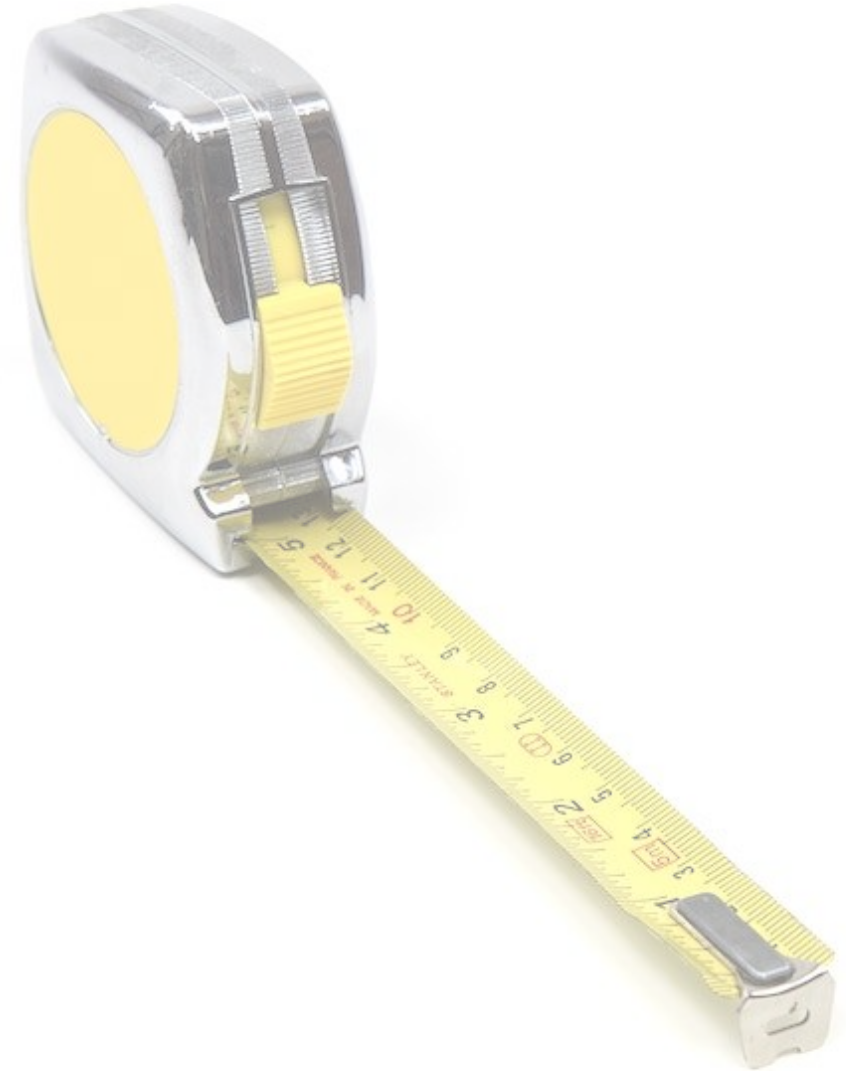
flat	flat%	sum%	cum	cum%	
1.21s	30.10%	30.10%	1.21s	30.10%	runtime/internal/syscall.Syscall6
0.91s	22.64%	52.74%	0.91s	22.64%	runtime.epollctl
0.59s	14.68%	67.41%	0.59s	14.68%	runtime.epollwait
0.17s	4.23%	71.64%	0.17s	4.23%	runtime.(*randomEnum).next (inline)
0.12s	2.99%	74.63%	0.35s	8.71%	runtime.stealWork
0.06s	1.49%	76.12%	0.06s	1.49%	runtime.addspecial
0.06s	1.49%	77.61%	0.06s	1.49%	runtime.exitsyscallfast
0.05s	1.24%	78.86%	0.89s	22.14%	internal/poll.(*FD).Close
0.05s	1.24%	80.10%	0.07s	1.74%	runtime.(*pollCache).alloc
0.05s	1.24%	81.34%	1.14s	28.36%	runtime.findRunnable
0.05s	1.24%	82.59%	0.08s	1.99%	runtime.lock2
0.05s	1.24%	83.83%	0.05s	1.24%	runtime.unlock2
0.04s	1%	84.83%	0.04s	1%	runtime.casgstatus
0.03s	0.75%	85.57%	0.05s	1.24%	internal/poll.runtime_pollUnblock

...

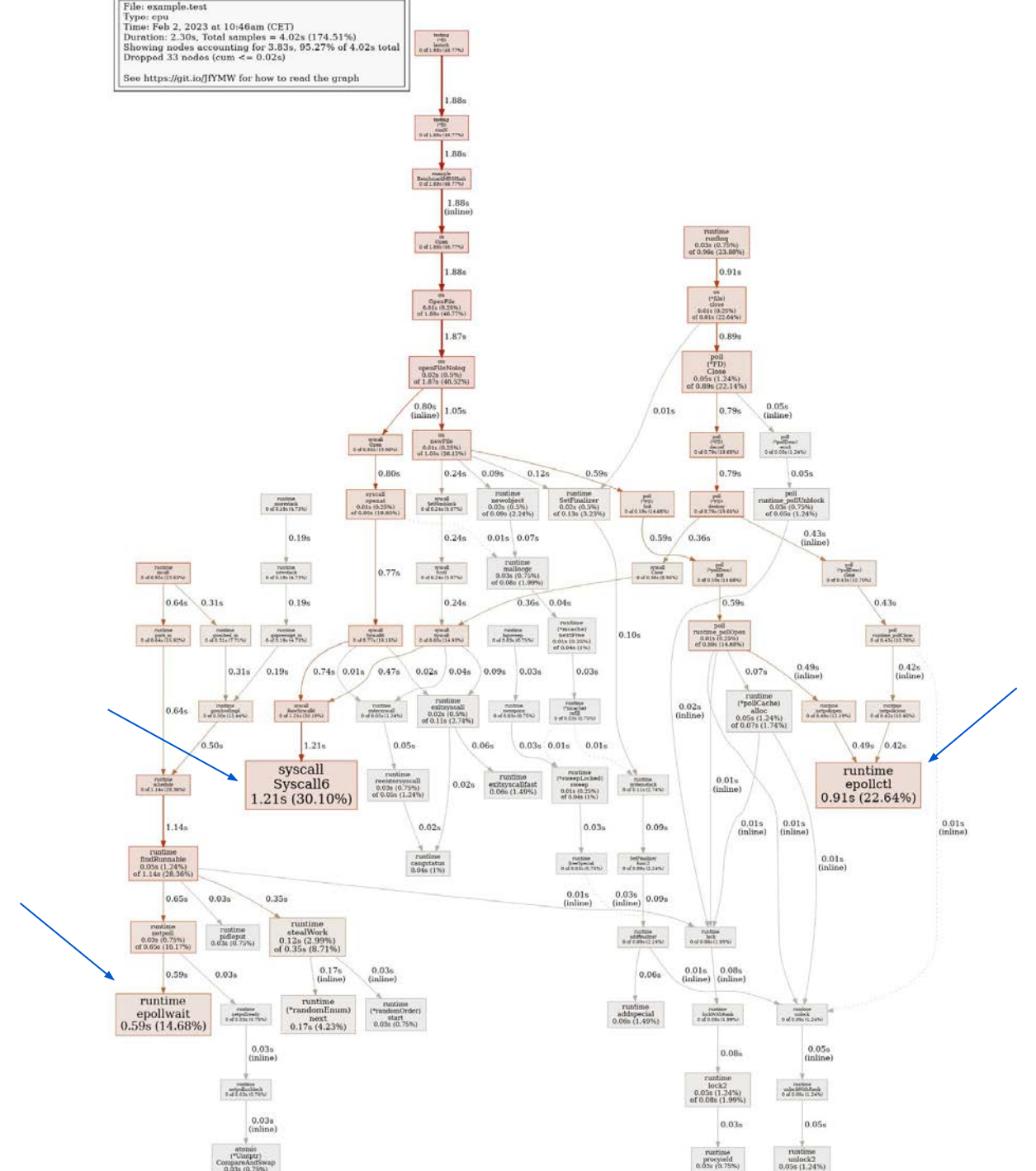


Profiling

```
$ go tool pprof -svg cpuprofile.out  
Generating report in profile002.svg
```



File: example.test
 Type: cpu
 Times: Feb 2, 2023 at 10:46am (CET)
 Duration: 2.30s, Total samples = 4.02s (174.51%)
 Showing nodes accounting for 3.83s, 95.27% of 4.02s total
 Dropped 33 nodes (cum <= 0.02s)
 See <https://git.io/JfYMW> for how to read the graph



Profiling

```
$ go tool pprof -list syscall.Syscall6 cpuprofile.out
```

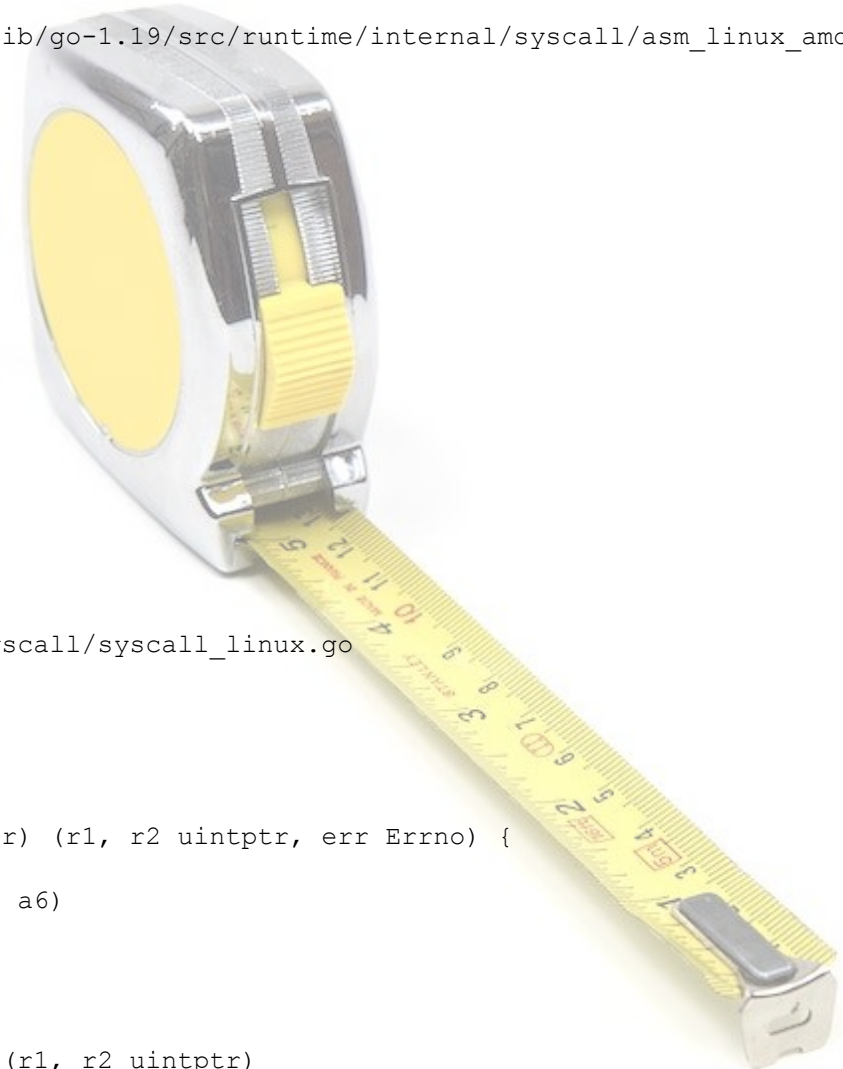
```
Total: 4.02s
```

```
ROUTINE ===== runtime/internal/syscall.Syscall6 in /usr/lib/go-1.19/src/runtime/internal/syscall/asm_linux_amd64.s
```

```
1.21s      1.21s (flat, cum) 30.10% of Total
.          .      31:  MOVQ   DI, DX  // a3
.          .      32:  MOVQ   CX, SI  // a2
.          .      33:  MOVQ   BX, DI  // a1
.          .      34:  // num already in AX.
.          .      35:  SYSCALL
1.20s      1.20s 36:  CMPQ   AX, $0xffffffffffffffff001
.          .      37:  JLS    ok
.          .      38:  NEGQ   AX
.          .      39:  MOVQ   AX, CX  // errno
.          .      40:  MOVQ   $-1, AX // r1
.          .      41:  MOVQ   $0, BX  // r2
.          .      42:  RET
.          .      43:ok:
.          .      44:  // r1 already in AX.
10ms      10ms 45:  MOVQ   DX, BX // r2
.          .      46:  MOVQ   $0, CX // errno
.          .      47:  RET
```

```
ROUTINE ===== syscall.Syscall6 in /usr/lib/go-1.19/src/syscall/syscall_linux.go
```

```
0          770ms (flat, cum) 19.15% of Total
.          .      85:
.          .      86://go:uintptrkeepalive
.          .      87://go:nosplit
.          .      88://go:linkname Syscall6
.          .      89:func Syscall6(trap, a1, a2, a3, a4, a5, a6 uintptr) (r1, r2 uintptr, err Errno) {
.          10ms 90:  runtime_entersyscall()
.          740ms 91:  r1, r2, err = RawSyscall6(trap, a1, a2, a3, a4, a5, a6)
.          20ms 92:  runtime_exitsyscall()
.          .      93:  return
.          .      94:}
.          .      95:
.          .      96:func rawSyscallNoError(trap, a1, a2, a3 uintptr) (r1, r2 uintptr)
.          .      97:
```



Reducing cpu usage



Reduce CPU usage

```
func find(needle string, haystack []string) int {  
    result := -1  
    for idx, i := range haystack {  
        if i == needle {  
            result = idx  
        }  
    }  
    return result  
}
```



Reduce CPU usage

```
var haystack = []string{
    "test0", "test1", "test2", "test3", "test4", "test5", "test6", "test8", "test9",
    "test10", "test11", "test12", "test13", "test14", "test15", "test16", "test18", "test19",
    "test20", "test21", "test22", "test23", "test24", "test25", "test26", "test28", "test29",
    "test30", "test31", "test32", "test33", "test34", "test35", "test36", "test38", "test39",
    "test40", "test41", "test42", "test43", "test44", "test45", "test46", "test48", "test49",
    "test50", "test51", "test52", "test53", "test54", "test55", "test56", "test58", "test59",
    "test60", "test61", "test62", "test63", "test64", "test65", "test66", "test68", "test69",
    "test70", "test71", "test72", "test73", "test74", "test75", "test76", "test78", "test79",
}

func BenchmarkFind(b *testing.B) {
    for i := 0; i < b.N; i++ {
        find("test50", haystack)
    }
}
```



Reduce CPU usage

```
goos: linux
goarch: amd64
pkg: example
cpu: Intel(R) Core(TM) i7-10750H CPU @ 2.60GHz
BenchmarkFind-12      4147192      289.7 ns/op
PASS
ok   example   1.497s
```



Reduce CPU usage

```
func find(needle string, haystack []string) int {  
    for idx, i := range haystack {  
        if i == needle {  
            return idx  
        }  
    }  
    return -1  
}
```



Reduce CPU usage

```
goos: linux
goarch: amd64
pkg: example
cpu: Intel(R) Core(TM) i7-10750H CPU @ 2.60GHz
BenchmarkFind-12      7065786      172.9 ns/op
PASS
ok   example  1.396s
```



Reducing allocations



Reduce allocations

```
func sliceFunc() {  
    slice := []int{}  
    for x := 0; x < 1000000; x++ {  
        slice = append(slice, x)  
    }  
}
```



Reduce allocations

```
func BenchmarkSliceFunc(b *testing.B) {  
    b.ReportAllocs()  
    for i := 0; i < b.N; i++ {  
        sliceFunc()  
    }  
}
```



Reduce allocations

```
goos: linux
goarch: amd64
pkg: github.com/jespino/squeezing-your-go-function
cpu: Intel(R) Core(TM) i7-10750H CPU @ 2.60GHz
BenchmarkSliceFunc-12      100      10455610 ns/op      41678303 B/op      39 allocs/op
PASS
ok   github.com/jespino/squeezing-your-go-function 1.058s
```



Reduce allocations

```
func sliceFunc() {  
    slice := make([]int, 1000000)  
    for x := 0; x < 1000000; x++ {  
        slice[x] = x  
    }  
}
```



Reduce allocations

```
goos: linux
goarch: amd64
pkg: github.com/jespino/squeezing-your-go-function
cpu: Intel(R) Core(TM) i7-10750H CPU @ 2.60GHz
BenchmarkSliceFunc-12      1417      824672 ns/op      8003596 B/op      1 allocs/op
PASS
ok   github.com/jespino/squeezing-your-go-function 1.258s
```



Reduce allocations

```
func sliceFunc2() {  
    array := [1000000]int{}  
    for x := 0; x < 1000000; x++ {  
        array[x] = x  
    }  
}
```



Reduce allocations

```
goos: linux
goarch: amd64
pkg: github.com/jespino/squeezing-your-go-function
cpu: Intel(R) Core(TM) i7-10750H CPU @ 2.60GHz
BenchmarkSliceFunc-12      4094      291574 ns/op
PASS
ok   github.com/jespino/squeezing-your-go-function 2.150s
```



Packing



Packing

```
type X struct {  
    a bool  
    // the compiler adds 7 bytes here  
    b float64  
    c int32  
    // the compiler adds 4 bytes here  
} // Total: 24 bytes
```

```
func Packing() {  
    slice := make([]X, 1000000)  
    for x := 0; x < 1000000; x++ {  
        slice[x] = X{}  
    }  
}
```



Packing

```
goos: linux
goarch: amd64
pkg: packing
cpu: Intel(R) Core(TM) i7-10750H CPU @ 2.60GHz
BenchmarkPacking-12      453      2623811 ns/op 24002567 B/op      1 allocs/op
PASS
ok   packing   1.460s
```



Packing

```
type X struct {  
    b float64  
    c int32  
    a bool  
    // the compiler adds 3 bytes here  
} // Total: 16 bytes
```



Packing

```
goos: linux
goarch: amd64
pkg: packing
cpu: Intel(R) Core(TM) i7-10750H CPU @ 2.60GHz
BenchmarkWithGoodPacking-12      619      2036888 ns/op  16007170 B/op      1 allocs/op
PASS
ok   packing   1.461s
```



Function Inlining



Function inlining

```
//go:noinline  
func notInlined() int {  
    return 7  
}
```



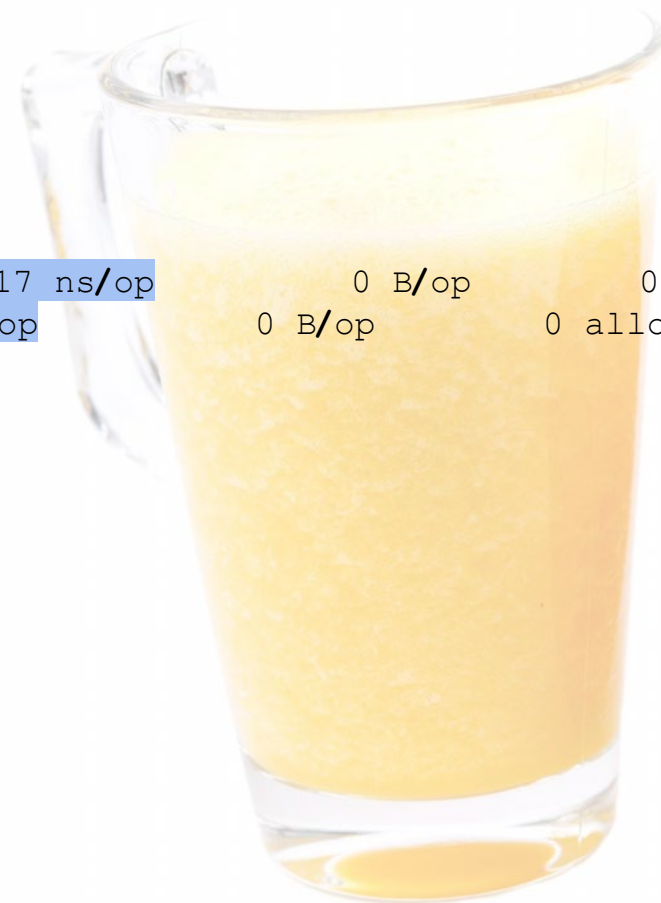
Function inlining

```
func inlined() int {  
    return 7  
}
```



Function inlining

```
goos: linux
goarch: amd64
pkg: github.com/jespino/squeezing-your-go-function
cpu: Intel(R) Core(TM) i7-10750H CPU @ 2.60GHz
BenchmarkInlinedFunc-12      1000000000      0.2817 ns/op      0 B/op      0 allocs/op
BenchmarkNotInlinedFunc-12  834508006      1.413 ns/op      0 B/op      0 allocs/op
PASS
ok  github.com/jespino/squeezing-your-go-function 1.642s
```



Escape Analysis



Escape analysis

```
//go:noinline  
func escaped() *int {  
    val := 7  
    return &val  
}
```



Escape analysis

```
//go:noinline  
func notEscaped() int {  
    val := 7  
    return val  
}
```



Escape analysis

```
goos: linux
goarch: amd64
pkg: github.com/jespino/squeezing-your-go-function
cpu: Intel(R) Core(TM) i7-10750H CPU @ 2.60GHz
BenchmarkEscaedFunc-12      74748568      13.38 ns/op      8 B/op      1 allocs/op
BenchmarkNotEscapedFunc-12 850284787      1.407 ns/op      0 B/op      0 allocs/op
PASS
ok      github.com/jespino/squeezing-your-go-function 2.360s
```



Escape Analysis and inlining



Escape analysis and function inlining

```
type Document struct {  
    path []string  
}  
  
func myFunc() {  
    _ = NewDocument("/proc/cpuinfo")  
}  
  
func NewDocument(path string) *Document {  
    d := &Document{}  
    for _, path := range filepath.SplitList(path) {  
        d.path = append(d.path, path)  
    }  
    return d  
}
```



Escape analysis and function inlining

```
goos: linux
goarch: amd64
pkg: github.com/jespino/squeezing-your-go-function
cpu: Intel(R) Core(TM) i7-10750H CPU @ 2.60GHz
BenchmarkMyFunc-12      9090330      128.2 ns/op      56 B/op      3 allocs/op
PASS
ok   github.com/jespino/squeezing-your-go-function  2.496s
```



Escape analysis and function inlining

```
func myFunc() {  
    d := NewDocument()  
    d.Init("/proc/cpuinfo")  
}  
  
func NewDocument() *Document {  
    return &Document{}  
}  
  
func (d *Document) Init(path string) {  
    for _, path := range filepath.SplitList(path) {  
        d.path = append(d.path, path)  
    }  
}
```



Escape analysis and function inlining

```
goos: linux
goarch: amd64
pkg: github.com/jespino/squeezing-your-go-function
cpu: Intel(R) Core(TM) i7-10750H CPU @ 2.60GHz
BenchmarkMyFunc-12      15195864      73.23 ns/op      32 B/op      2 allocs/op
PASS
ok      github.com/jespino/squeezing-your-go-function  2.496s
```



Concurrency



Concurrency

```
func fakeIO() {
    counter := 100
    for x := 0; x < counter; x++ {
        time.Sleep(10 * time.Millisecond)
    }
}

func fakeIOParallel(goroutines int) {
    var wg sync.WaitGroup
    wg.Add(goroutines)
    counter := 100
    for x := 0; x < goroutines; x++ {
        go func(idx int) {
            defer wg.Done()
            for y := 0; y < counter; y++ {
                if y%goroutines == idx {
                    time.Sleep(10 * time.Millisecond)
                }
            }
        }(x)
    }
    wg.Wait()
}
```



Concurrency

```
func BenchmarkIOParallelOnePerJob(b *testing.B) {  
    b.ReportAllocs()  
    for i := 0; i < b.N; i++ {  
        fakeIOParallel(100)  
    }  
}
```

```
func BenchmarkIOParallelOnePerCpu(b *testing.B) {  
    b.ReportAllocs()  
    cpus := runtime.NumCPU()  
    for i := 0; i < b.N; i++ {  
        fakeIOParallel(cpus)  
    }  
}
```

```
func BenchmarkIOSerial(b *testing.B) {  
    b.ReportAllocs()  
    for i := 0; i < b.N; i++ {  
        fakeIO()  
    }  
}
```



Concurrency

```
goos: linux
goarch: amd64
pkg: concurrency
cpu: Intel(R) Core(TM) i7-10750H CPU @ 2.60GHz
BenchmarkIOParallelOnePerJob-12      100      10599470 ns/op      15488 B/op      305 allocs/op
BenchmarkIOParallelOnePerCpu-12     12      91591538 ns/op      1648 B/op       37 allocs/op
BenchmarkIOSerial-12                 1    1014305834 ns/op      80 B/op         1 allocs/op
PASS
ok   concurrency    3.282s
```



Concurrency

```
func fakeCPU() {
    counter := 1000
    for x := 0; x < counter; x++ {
        _ = md5.Sum([]byte("test"))
    }
}

func fakeCPUParallel(goroutines int) {
    var wg sync.WaitGroup
    wg.Add(goroutines)
    counter := 1000
    for x := 0; x < goroutines; x++ {
        go func(idx int) {
            defer wg.Done()
            for y := 0; y < counter; y++ {
                if y%goroutines == idx {
                    _ = md5.Sum([]byte("test"))
                }
            }
        }(x)
    }
    wg.Wait()
}
```



Concurrency

```
func BenchmarkCPUParallelOnePerJob(b *testing.B) {  
    b.ReportAllocs()  
    for i := 0; i < b.N; i++ {  
        fakeCPUParallel(1000)  
    }  
}
```

```
func BenchmarkCPUParallelOnePerCpu(b *testing.B) {  
    b.ReportAllocs()  
    cpus := runtime.NumCPU()  
    for i := 0; i < b.N; i++ {  
        fakeCPUParallel(cpus)  
    }  
}
```

```
func BenchmarkCPUSerial(b *testing.B) {  
    b.ReportAllocs()  
    for i := 0; i < b.N; i++ {  
        fakeCPU()  
    }  
}
```



Concurrency

```
goos: linux
goarch: amd64
pkg: concurrency
cpu: Intel(R) Core(TM) i7-10750H CPU @ 2.60GHz
BenchmarkCPUParallelOnePerJob-12      1042    1296818 ns/op    56066 B/op    2001 allocs/op
BenchmarkCPUParallelOnePerCpu-12     20469      65477 ns/op     688 B/op     25 allocs/op
BenchmarkCPUSerial-12                 10000    114459 ns/op     0 B/op       0 allocs/op
PASS
ok   concurrency    4.563s
```



Summary



References

- [Efficient Go \(By Bartłomiej Plotka\)](#)
- [High Performance Go Workshop \(By Dave Cheney\)](#)
- [Go-perfbook \(By Damian Gryski\)](#)
- [Ultimate Go \(By Ardan Labs\)](#)



Creative commons images authors

- <https://www.flickr.com/photos/30478819@N08/48337042122>
- <https://www.flickr.com/photos/30478819@N08/48228177342>
- <https://www.flickr.com/photos/bensutherland/205587168>
- <https://www.flickr.com/photos/kecko/4632134325>
- <https://www.flickr.com/photos/suckamc/5685064151>
- [https://commons.wikimedia.org/wiki/File:Hong Kong Juice Brand %282879276310%29.jpg](https://commons.wikimedia.org/wiki/File:Hong_Kong_Juice_Brand_%282879276310%29.jpg)
- <https://www.flickr.com/photos/sagamiono/4209299708>
- <https://www.flickr.com/photos/dolmansaxlil/5347064183>
- <https://www.pexels.com/photo/orange-on-squeezer-8679404/>



Thank
you.

