Golang & Performance

A Brief but Effective Introduction to Benchmarking

November 2024

Ismail Simsek
Senior Software Engineer

Ismail Simsek

Tech Lead of Observability Metrics

https://github.com/itsmylife

https://www.linkedin.com/in/ismailsimsek09

Agenda

1. Prerequisites
2. Benchmarking
3. Tips & Tricks
4. Real Life Examples
5. Final Notes

What is this?

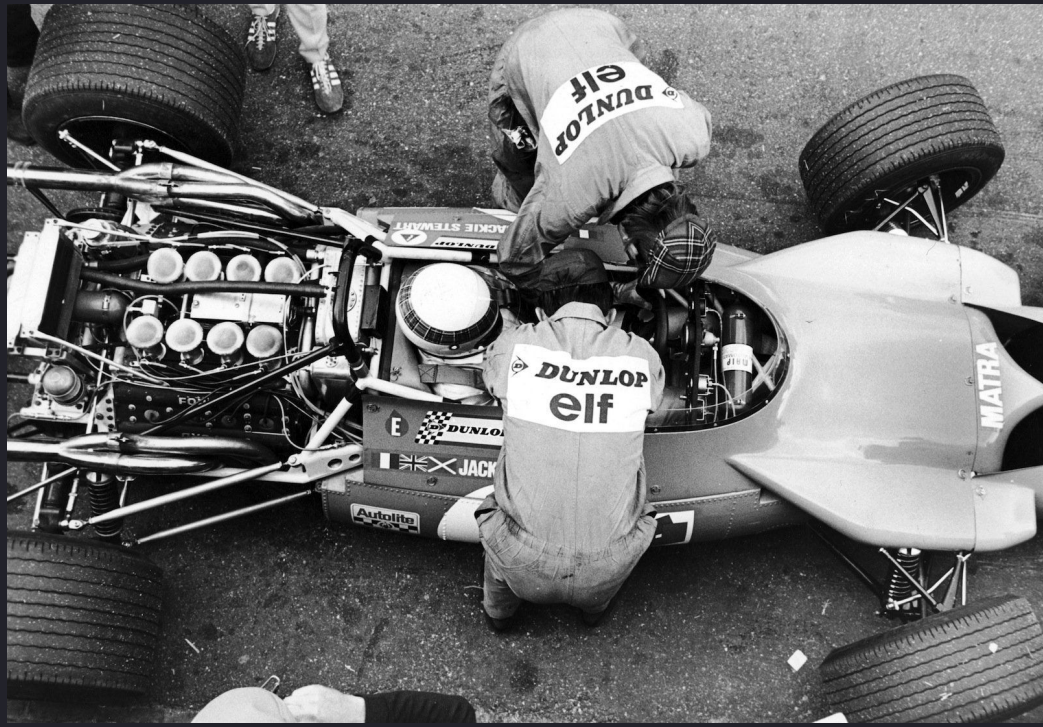
- Dave Cheney's High Performance Go Workshop
- ~~We won't cover the whole workshop~~
- Scratching the surface
- How to read the comparison results

Prerequisites

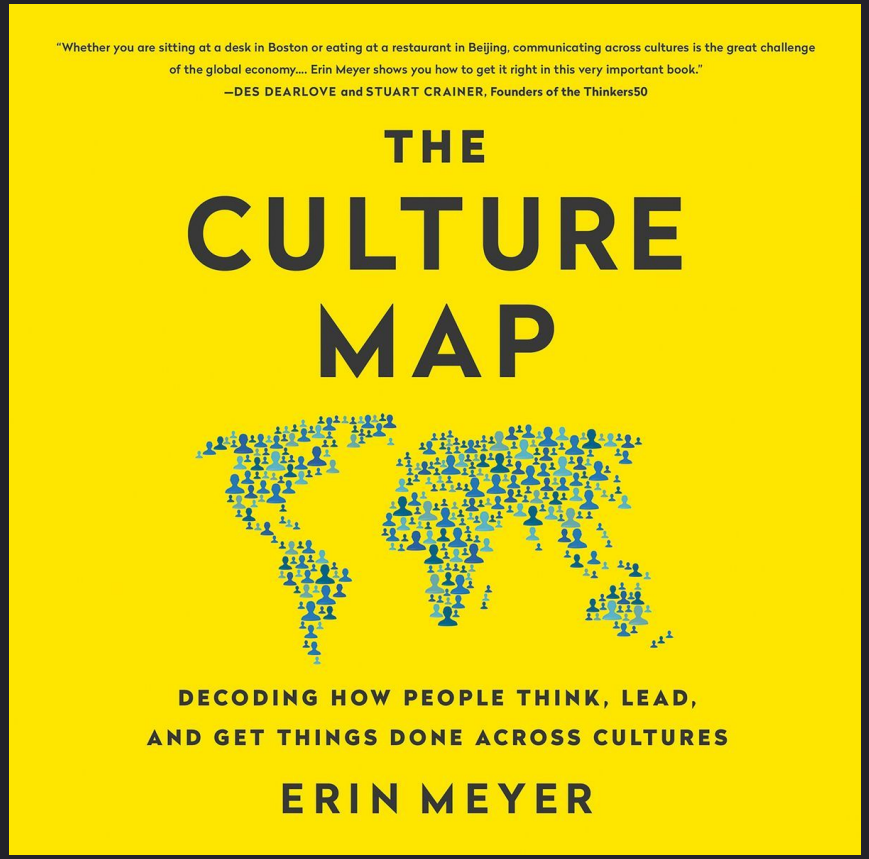
- Go development environment
- Clone this git repository:
  - https://github.com/davecheney/high-performance-go-workshop
  - `go mod download` after fetching, for dependencies.
- Benchstat
  - go install golang.org/x/perf/cmd/benchstat@latest

Mechanical Sympathy

To excel in any field, you don't need to build the tools you use, but you must understand how they work to use them effectively.



Mechanical Sympathy



Let’s GO!

Calculate Fibonacci

```
// Fib computes the n'th number in the Fibonacci series.
func Fib(n int) int {
    switch n {
    case 0:
        return 0
    case 1:
        return 1
    default:
        return Fib(n-1) + Fib(n-2)
    }
}
```

Benchmark Test (*wrong*)

```
func BenchmarkFibWrong(b *testing.B) {
    Fib(N)
}

func BenchmarkFibWrong2(b *testing.B) {
   for n := 0; n < b.N; n++ {
      Fib(n)
   }
}
```

Benchmark Test

```
func BenchmarkFib20(b *testing.B) {
    b.ReportAllocs()
    b.ResetTimer()
    for n := 0; n < b.N; n++ {
        Fib(20) // run the Fib function b.N times
    }
}
```

Benchmark Test

```
func BenchmarkFib20(b *testing.B) {
    b.ReportAllocs()
    b.ResetTimer()
    for n := 0; n < b.N; n++ {
        Fib(20) // run the Fib function b.N times
    }
}
```

Benchmark Test

```
func BenchmarkFib20(b *testing.B) {
    b.ReportAllocs()
    b.ResetTimer()
    for n := 0; n < b.N; n++ {
        Fib(20) // run the Fib function b.N times
    }
}
```

Benchmark Test

```
func BenchmarkFib20(b *testing.B) {
    b.ReportAllocs()
    b.ResetTimer()
    for n := 0; n < b.N; n++ {
        Fib(20) // run the Fib function b.N times
    }
}
```

Run benchmark test

```
go test -benchmem -run=^$ \
 -memprofile mem.out \
 -cpuprofile cpu.out \
 -count 10 \
 -bench ^BenchmarkFib20 ./02-benchmarking/examples/fib |tee old.txt
```

Run benchmark test

```
go test -benchmem -run=^$ \
 -memprofile mem.out \
 -cpuprofile cpu.out \
 -count 10 \
 -bench ^BenchmarkFib20 ./02-benchmarking/examples/fib |tee old.txt
```

Run benchmark test

```
go test -benchmem -run=^$ \
 -memprofile mem.out \
 -cpuprofile cpu.out \
 -count 10 \
 -bench ^BenchmarkFib20 ./02-benchmarking/examples/fib |tee old.txt
```

Run benchmark test

```
go test -benchmem -run=^$ \
 -memprofile mem.out \
 -cpuprofile cpu.out \
 -count 10 \
 -bench ^BenchmarkFib20 ./02-benchmarking/examples/fib |tee old.txt
```

Run benchmark test

```
go test -benchmem -run=^$ \
 -memprofile mem.out \
 -cpuprofile cpu.out \
 -count 10 \
 -bench ^BenchmarkFib20 ./02-benchmarking/examples/fib |tee old.txt
```

Before improvement

```
goos: darwin
goarch: arm64
pkg: ./high-performance-go-workshop/02-benchmarking/examples/fib
BenchmarkFib20-10          41174        29086 ns/op          0 B/op         0 allocs/op
BenchmarkFib20-10          41322        29068 ns/op          0 B/op         0 allocs/op
BenchmarkFib20-10          41119        29074 ns/op          0 B/op         0 allocs/op
BenchmarkFib20-10          41247        29062 ns/op          0 B/op         0 allocs/op
BenchmarkFib20-10          41319        29076 ns/op          0 B/op         0 allocs/op
BenchmarkFib20-10          41229        29100 ns/op          0 B/op         0 allocs/op
BenchmarkFib20-10          41328        29087 ns/op          1 B/op         0 allocs/op
BenchmarkFib20-10          41282        29256 ns/op          0 B/op         0 allocs/op
BenchmarkFib20-10          41300        29083 ns/op          0 B/op         0 allocs/op
BenchmarkFib20-10          41254        29046 ns/op          0 B/op         0 allocs/op
PASS
ok      ./high-performance-go-workshop/02-benchmarking/examples/fib   15.140s
```

Improve the code

```
// Fib computes the n'th number in the Fibonacci series.
func Fib(n int) int {
    switch n {
    case 0:
        return 0
    case 1:
        return 1
    case 2:
        return 2
    default:
        return Fib(n-1) + Fib(n-2)
    }
}
```

Improve the code

```
// Fib co                                    les.
func Fib
    sw
    ca
        return 0
    case 1:
        return 1
    case 2:
        return 2
    default:
        return F
    }
}
```

I can make things very fast
if they don't have to be correct.

Russ Cox
https://github.com/rsc/

Improve the code

```
// Fib computes the n'th number in the Fibonacci series.
func Fib(n int) int {
    switch n {
    case 0:
        return 0
    case 1:
        return 1
    case 2:
        return 2
    default:
        return Fib(n-1) + Fib(n-2)
    }
}
```

Have unit tests!

Improve the code

```
// Fib computes the n'th number in the Fibonacci series.
func Fib(n int) int {
    switch n {
    case 0:
        return 0
    case 1:
        return 1
    case 2:
        return 1
    default:
        return Fib(n-1) + Fib(n-2)
    }
}
```

Run benchmark test again

```
go test -benchmem -run=^$ \
 -memprofile mem.out \
 -cpuprofile cpu.out \
 -count 10 \
 -bench ^BenchmarkFib20 ./02-benchmarking/examples/fib |tee new.txt
```

After improvement

```
goos: darwin
goarch: arm64
pkg: ./high-performance-go-workshop/02-benchmarking/examples/fib
BenchmarkFib20-10          66427        17889 ns/op          0 B/op         0 allocs/op
BenchmarkFib20-10          67062        17880 ns/op          0 B/op         0 allocs/op
BenchmarkFib20-10          67140        17935 ns/op          0 B/op         0 allocs/op
BenchmarkFib20-10          66949        17923 ns/op          0 B/op         0 allocs/op
BenchmarkFib20-10          67134        19924 ns/op          0 B/op         0 allocs/op
BenchmarkFib20-10          67029        17867 ns/op          0 B/op         0 allocs/op
BenchmarkFib20-10          67058        17869 ns/op          0 B/op         0 allocs/op
BenchmarkFib20-10          67023        17855 ns/op          0 B/op         0 allocs/op
BenchmarkFib20-10          67134        17879 ns/op          0 B/op         0 allocs/op
BenchmarkFib20-10          67088        17869 ns/op          0 B/op         0 allocs/op
PASS
ok      ./high-performance-go-workshop/02-benchmarking/examples/fib   14.369s
```

Compare the results

```
benchstat old.txt new.txt
goos: darwin
goarch: arm64
pkg: github.com/grafana/high-performance-go-workshop/02-benchmarking/examples/fib
        |   old.txt   |               new.txt               |
        |   sec/op    |   sec/op     vs base                |
Fib20-10   29.08µ ± 0%   17.88µ ± 0%  -38.52% (p=0.000 n=10)

        |  old.txt   |           new.txt            |
        |    B/op    |    B/op     vs base          |
Fib20-10   0.000 ± 0%   0.000 ± 0%  ~ (p=1.000 n=10)

        |  old.txt   |            new.txt             |
        | allocs/op  | allocs/op   vs base             |
Fib20-10   0.000 ± 0%   0.000 ± 0%  ~ (p=1.000 n=10) ¹
¹ all samples are equal
```

Compare the results

```
benchstat old.txt new.txt
goos: darwin
goarch: arm64
pkg: github.com/grafana/high-performance-go-workshop/02-benchmarking/examples/fib
        |   old.txt    |               new.txt                |
        |   sec/op     |   sec/op     vs base                 |
Fib20-10   29.08µ ± 0%   17.88µ ± 0%  -38.52% (p=0.000 n=10)

        |  old.txt    |           new.txt            |
        |    B/op     |    B/op     vs base          |
Fib20-10   0.000 ± 0%   0.000 ± 0%  ~ (p=1.000 n=10)

        |  old.txt    |            new.txt            |
        | allocs/op   | allocs/op   vs base            |
Fib20-10   0.000 ± 0%   0.000 ± 0%  ~ (p=1.000 n=10) ¹
¹ all samples are equal
```

Allocation

Compare the results

```
benchstat old.txt new.txt
goos: darwin
goarch: arm64
pkg: github.com/grafana/high-performance-go-workshop/02-benchmarking/examples/fib
        |   old.txt    |               new.txt                |
        |   sec/op     |   sec/op     vs base                 |
Fib20-10   29.08µ ± 0%   17.88µ ± 0%  -38.52% (p=0.000 n=10)

        |  old.txt    |           new.txt            |
        |    B/op     |    B/op     vs base          |
Fib20-10   0.000 ± 0%   0.000 ± 0%  ~ (p=1.000 n=10)

        |  old.txt    |           new.txt            |
        | allocs/op   | allocs/op   vs base          |
Fib20-10   0.000 ± 0%   0.000 ± 0%  ~ (p=1.000 n=10) ¹
¹ all samples are equal
```

Memory

Compare the results

```
benchstat old.txt new.txt
goos: darwin
goarch: arm64
pkg: github.com/grafana/high-performance-go-workshop/02-benchmarking/examples/fib
        |   old.txt    |               new.txt                |
        |   sec/op     |   sec/op     vs base                 |
Fib20-10   29.08µ ± 0%   17.88µ ± 0%  -38.52% (p=0.000 n=10)

        |  old.txt    |           new.txt            |
        |    B/op     |    B/op     vs base          |
Fib20-10   0.000 ± 0%   0.000 ± 0%  ~ (p=1.000 n=10)

        |  old.txt    |            new.txt             |
        | allocs/op   | allocs/op   vs base            |
Fib20-10   0.000 ± 0%   0.000 ± 0%  ~ (p=1.000 n=10) ¹
¹ all samples are equal
```

CPU

Compare the results

```
benchstat old.txt new.txt
goos: darwin
goarch: arm64
pkg: github.com/grafana/high-performance-go-workshop/02-benchmarking/examples/fib
        |   old.txt    |               new.txt                |
        |   sec/op     |   sec/op     vs base                 |
Fib20-10   29.08µ

        |  old.tx   |              new.txt                 |
        |    B/op   |    B/op     vs base                  |
Fib20-10   0.000 ± 0%   0.000 ± 0%  ~ (p=1.000 n=10)

        |  old.txt   |            new.txt                  |
        | allocs/op  | allocs/op   vs base                  |
Fib20-10   0.000 ± 0%   0.000 ± 0%  ~ (p=1.000 n=10) ¹
¹ all samples are equal
```

CPU

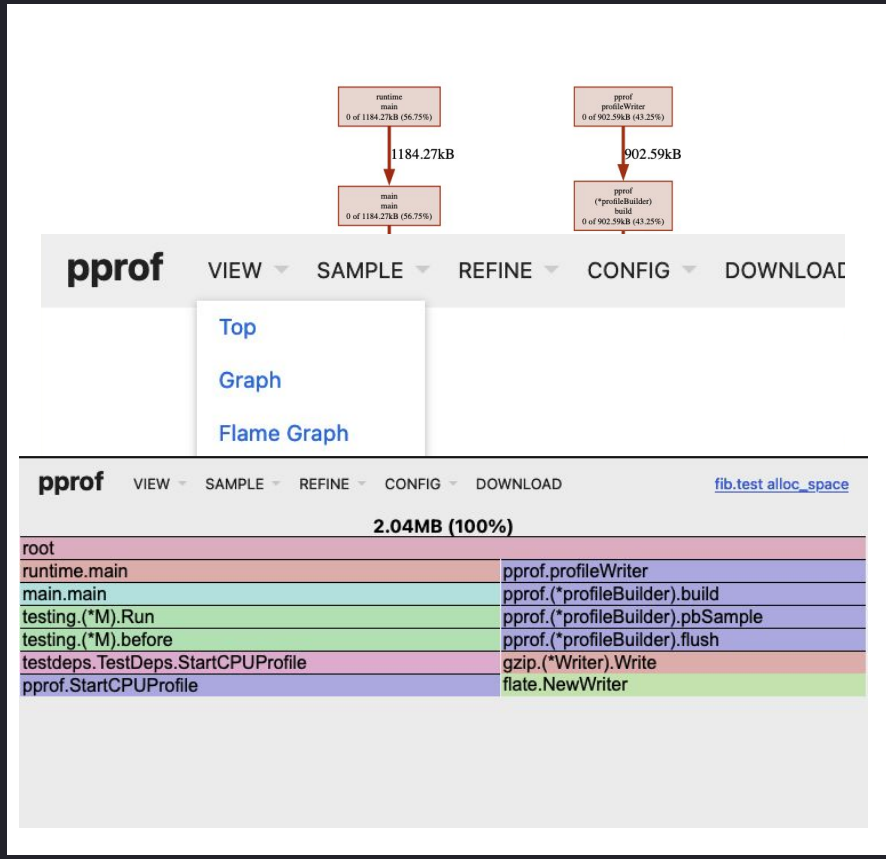
Have a stable environment

cpu.out & mem.out

```
go test -benchmem -run=^$ \
 -memprofile mem.out \
 -cpuprofile cpu.out \
 -count 10 \
 -bench ^BenchmarkFib20 ./02-benchmarking/examples/fib |tee old.txt
```

pprof

```
go tool pprof -http :9999 cpu.out

go tool pprof -http :7777 mem.out
```



Tips & Tricks

Reduce Allocations

```
// Allocates a buffer
func (r *Reader) Read() ([]byte, error)

// Uses the buffer it was given
func (r *Reader) Read(buf []byte) (int, error)
```

strings and [ ]byte

- In Go `string` values are immutable, `[ ]byte` are mutable.
- IO operations prefer [ ]byte
- Avoid [ ]byte to string conversions wherever possible
- bytes package already contains many of the operations
  - Split, Compare, HasPrefix, Trim, etc

strings

- Go strings are immutable.
- Concatenating two strings generates a third.
- Which of the following is fastest?

Concatenate

```
func BenchmarkConcatenate(b *testing.B) {
   // setup test
   var r string
   for n := 0; n < b.N; n++ {
      s := request.ID
      s += " " + client.Addr().String()
      s += " " + time.Now().String()
      r = s
   }
   Result = r
}
```

https://github.com/davecheney/high-performance-go-workshop/blob/master/examples/concat/concat_test.go#L27-L43

Fprintf()

```
func BenchmarkFprintf(b *testing.B) {
   // setup test
   var r string
   for n := 0; n < b.N; n++ {
      var b bytes.Buffer
      fmt.Fprintf(&b, "%s %v %v", request.ID, client.Addr(), time.Now())
      r = b.String()
   }
   Result = r
}
```

https://github.com/davecheney/high-performance-go-workshop/blob/master/examples/concat/concat_test.go#L45-L60

Sprintf()

```
func BenchmarkSprintf(b *testing.B) {
    // setup test
    var r string
    for n := 0; n < b.N; n++ {
        r = fmt.Sprintf("%s %v %v", request.ID, client.Addr(), time.Now())
    }
    Result = r
}
```

https://github.com/davecheney/high-performance-go-workshop/blob/master/examples/concat/concat_test.go#L62-L75

Strconv

```
func BenchmarkStrconv(b *testing.B) {
   // setup test
   var r string
   for n := 0; n < b.N; n++ {
      b := make([]byte, 0, 40)
      b = append(b, request.ID...)
      b = append(b, ' ')
      b = append(b, client.Addr().String()...)
      b = append(b, ' ')
      b = time.Now().AppendFormat(b, "2006-01-02 15:04:05.999999999 -0700 MST")
      r = string(b)
   }
   Result = r
}
```

https://github.com/davecheney/high-performance-go-workshop/blob/master/examples/concat/concat_test.go#L77-L96

StringsBuilder

```
func BenchmarkStringsBuilder(b *testing.B) {
    // setup test
    var r string
    for n := 0; n < b.N; n++ {
        var b strings.Builder
        b.WriteString(request.ID)
        b.WriteString(" ")
        b.WriteString(client.Addr().String())
        b.WriteString(" ")
        b.WriteString(time.Now().String())
        r = b.String()
    }
    Result = r
}
```

https://github.com/davecheney/high-performance-go-workshop/blob/master/examples/concat/concat_test.go#L98-L117

Avoid string concatenation

./examples/concat

```
❯ go test -bench=. ./examples/concat
goos: darwin
goarch: arm64
pkg: github.com/grafana/high-performance-go/examples/concat
BenchmarkConcatenate-10          2547897       462.6 ns/op       245 B/op      8 allocs/op
BenchmarkFprintf-10              1890890       643.7 ns/op       397 B/op     11 allocs/op
BenchmarkSprintf-10              2072682       568.7 ns/op       269 B/op      9 allocs/op
BenchmarkStrconv-10              3434775       350.2 ns/op       165 B/op      5 allocs/op
BenchmarkStringsBuilder-10       2662629       451.2 ns/op       255 B/op      9 allocs/op
PASS
ok      github.com/grafana/high-performance-go/examples/concat 8.650s
```

Avoid string concatenation

./examples/concat

```
❯ go test -bench=. ./examples/concat
goos: darwin
goarch: arm64
pkg: github.com/grafana/high-performance-go/examples/concat
BenchmarkConcatenate-10          2547897       462.6 ns/op       245 B/op      8 allocs/op
BenchmarkFprintf-10              1890890       643.7 ns/op       397 B/op     11 allocs/op
BenchmarkSprintf-10              2072682       568.7 ns/op       269 B/op      9 allocs/op
BenchmarkStrconv-10              3434775       350.2 ns/op       165 B/op      5 allocs/op
BenchmarkStringsBuilder-10       2662629       451.2 ns/op       255 B/op      9 allocs/op
PASS
ok      github.com/grafana/high-performance-go/examples/concat 8.650s
```

Avoid string concatenation

🥇 Strconv

```
func BenchmarkStrconv(b *testing.B) {
   // setup test
   var r string
   for n := 0; n < b.N; n++ {
      b := make([]byte, 0, 40)
      b = append(b, request.ID...)
      b = append(b, ' ')
      b = append(b, client.Addr().String()...)
      b = append(b, ' ')
      b = time.Now().AppendFormat(b, "2006-01-02 15:04:05.999999999 -0700 MST")
      r = string(b)
   }
   Result = r
}
```

https://github.com/davecheney/high-performance-go-workshop/blob/master/examples/concat/concat_test.go#L77-L96

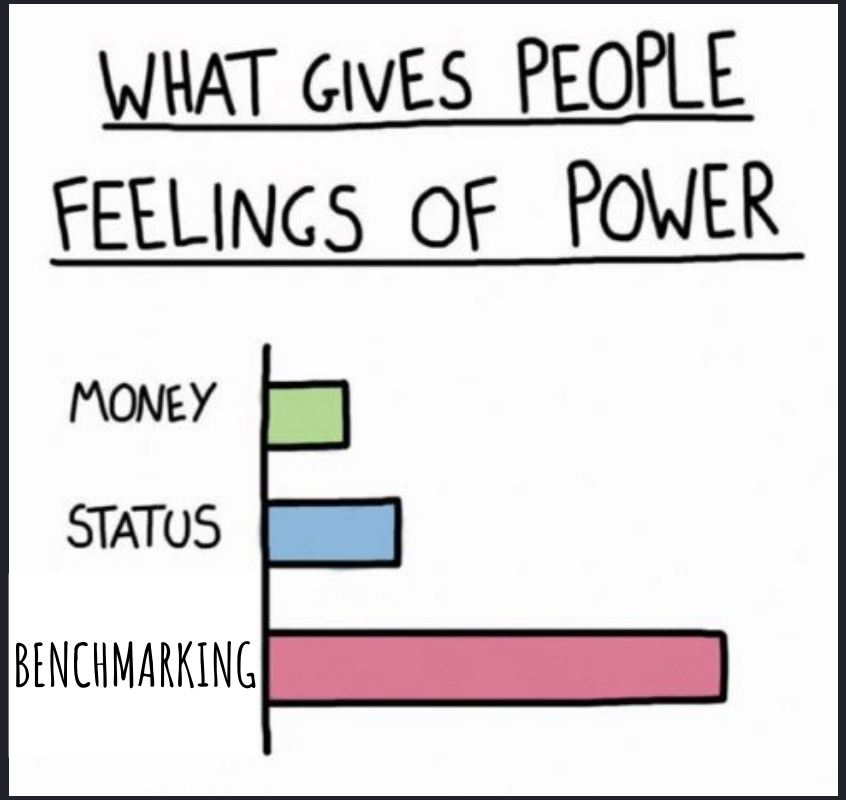
Preallocate slices if the length is known

```
// Before
var s []string
for _, v := range fn() {
    s = append(s, v)
}
return s
```

```
// After
vals := fn()
s := make([]string, len(vals))
for i, v := range vals {
    s[i] = v
}
return s
```

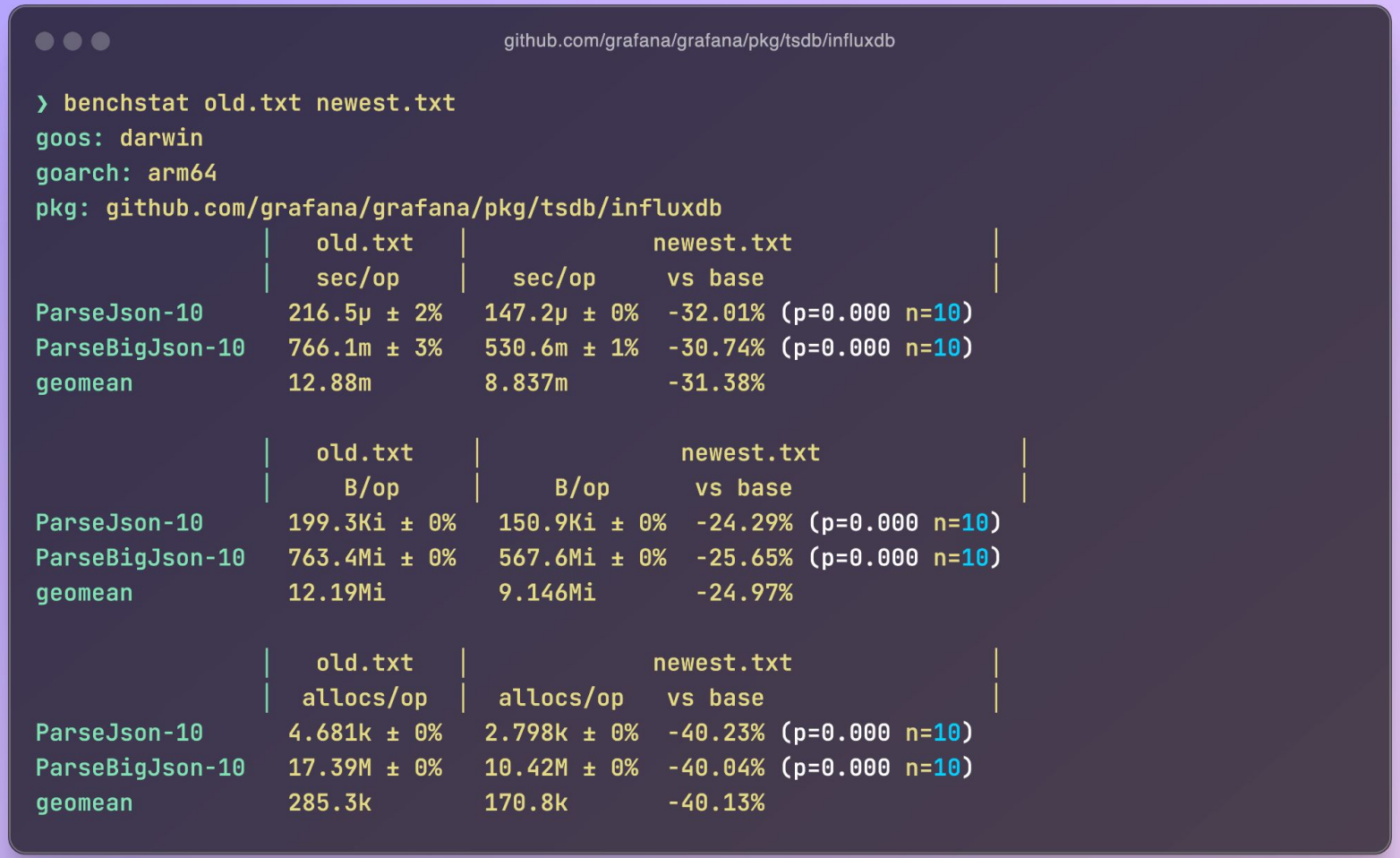
https://github.com/alexkohler/prealloc

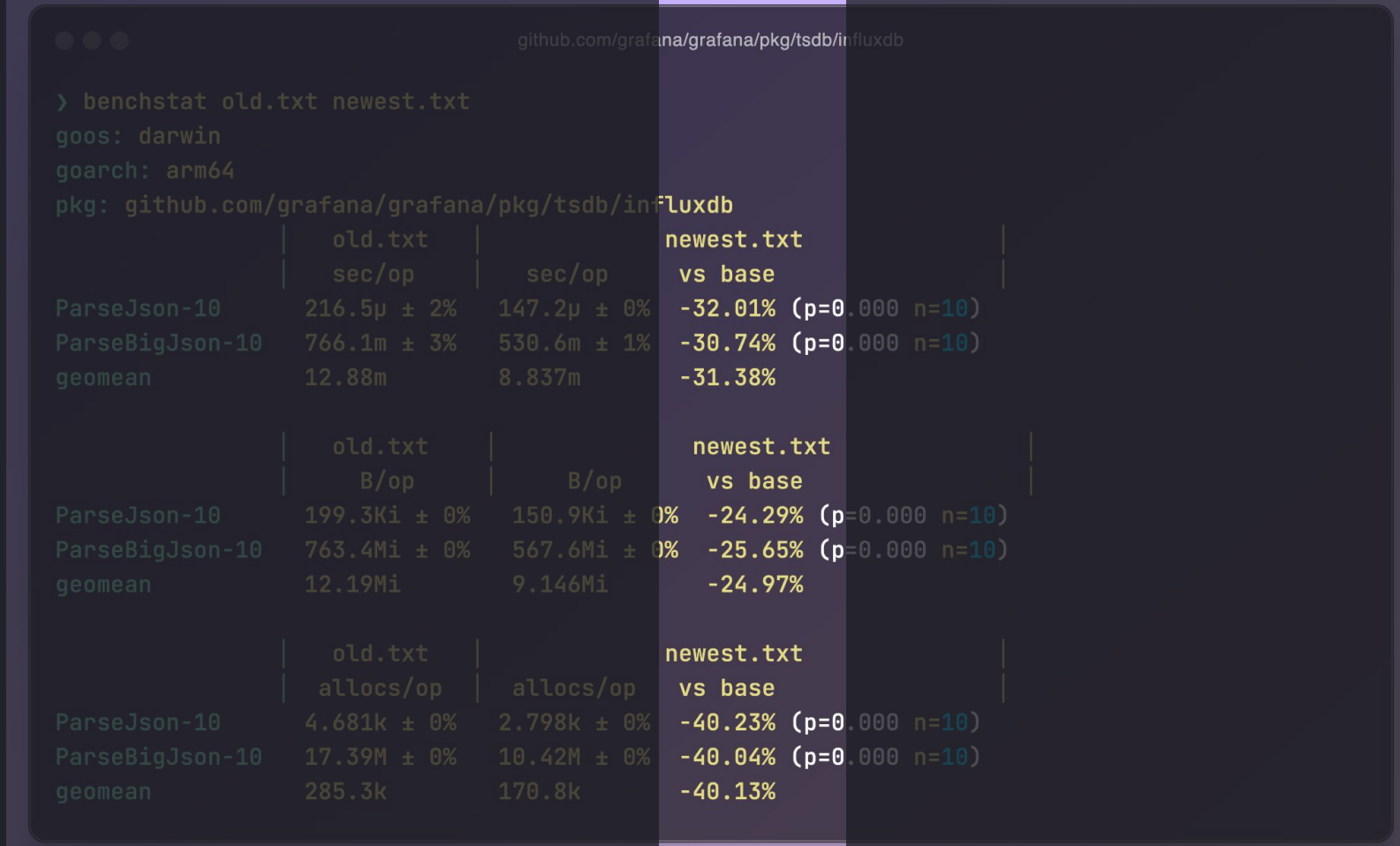
Real world examples



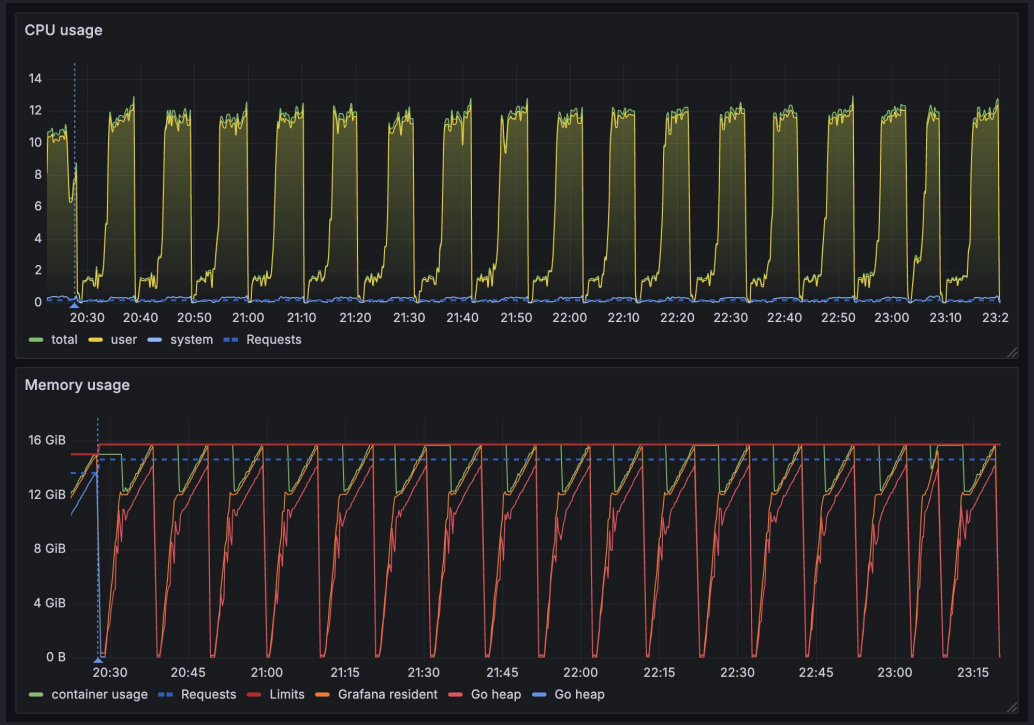
Benchmarking in the wild #1

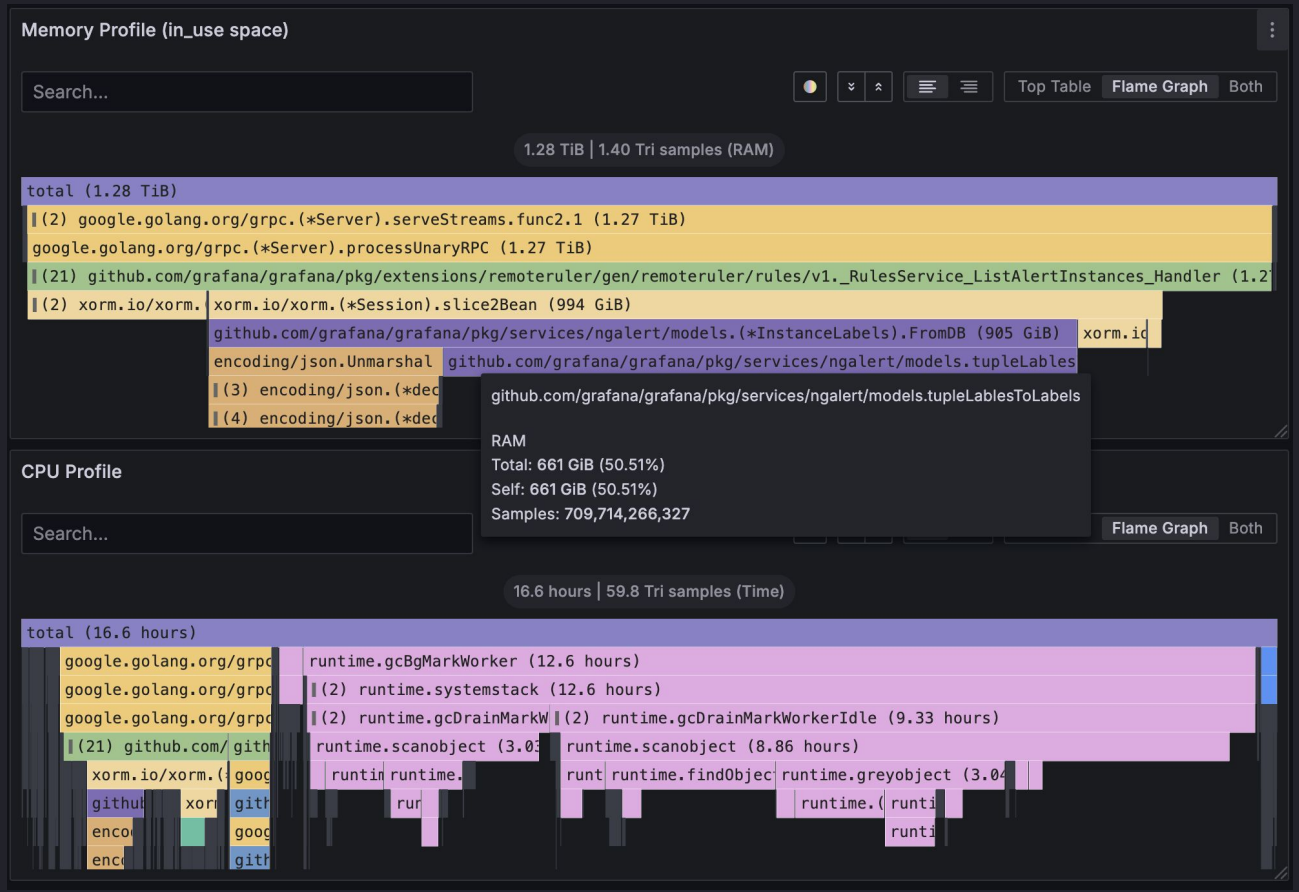
```
65    func transformRows(rows []Row, query Query) data.Frames {
66  +     // pre-allocate frames - this can save many allocations
67  +     cols := 0
68  +     for _, row := range rows {
69  +         cols += len(row.Columns)
70  +     }
71  +     frames := make([]*data.Frame, 0, len(rows)+cols)
72  +
73  +     // frameName is pre-allocated so we can reuse it, saving memory.
74  +     // It's sized for a reasonably-large name, but will grow if needed.
75  +     frameName := make([]byte, 0, 128)
76  +
```


github.com/grafana/grafana/pkg/tsdb/influxdb
❯ benchstat old.txt newest.txt
goos: darwin
goarch: arm64
pkg: github.com/grafana/grafana/pkg/tsdb/influxdb
| old.txt | newest.txt |
| sec/op | sec/op vs base |
ParseJson-10 216.5µ ± 2% 147.2µ ± 0% -32.01% (p=0.000 n=10)
ParseBigJson-10 766.1m ± 3% 530.6m ± 1% -30.74% (p=0.000 n=10)
geomean 12.88m 8.837m -31.38%
| old.txt | newest.txt |
| B/op | B/op vs base |
ParseJson-10 199.3Ki ± 0% 150.9Ki ± 0% -24.29% (p=0.000 n=10)
ParseBigJson-10 763.4Mi ± 0% 567.6Mi ± 0% -25.65% (p=0.000 n=10)
geomean 12.19Mi 9.146Mi -24.97%
| old.txt | newest.txt |
| allocs/op | allocs/op vs base |
ParseJson-10 4.681k ± 0% 2.798k ± 0% -40.23% (p=0.000 n=10)
ParseBigJson-10 17.39M ± 0% 10.42M ± 0% -40.04% (p=0.000 n=10)
geomean 285.3k 170.8k -40.13%


github.com/grafana/grafana/pkg/tsdb/influxdb
❯ benchstat old.txt newest.txt
goos: darwin
goarch: arm64
pkg: github.com/grafana/grafana/pkg/tsdb/influxdb
| old.txt | newest.txt |
| sec/op | sec/op vs base |
ParseJson-10 216.5µ ± 2% 147.2µ ± 0% -32.01% (p=0.000 n=10)
ParseBigJson-10 766.1m ± 3% 530.6m ± 1% -30.74% (p=0.000 n=10)
geomean 12.88m 8.837m -31.38%
| old.txt | newest.txt |
| B/op | B/op vs base |
ParseJson-10 199.3Ki ± 0% 150.9Ki ± 0% -24.29% (p=0.000 n=10)
ParseBigJson-10 763.4Mi ± 0% 567.6Mi ± 0% -25.65% (p=0.000 n=10)
geomean 12.19Mi 9.146Mi -24.97%
| old.txt | newest.txt |
| allocs/op | allocs/op vs base |
ParseJson-10 4.681k ± 0% 2.798k ± 0% -40.23% (p=0.000 n=10)
ParseBigJson-10 17.39M ± 0% 10.42M ± 0% -40.04% (p=0.000 n=10)
geomean 285.3k 170.8k -40.13%

Benchmarking in the wild #2





https://grafana.com/docs/pyroscope/latest/

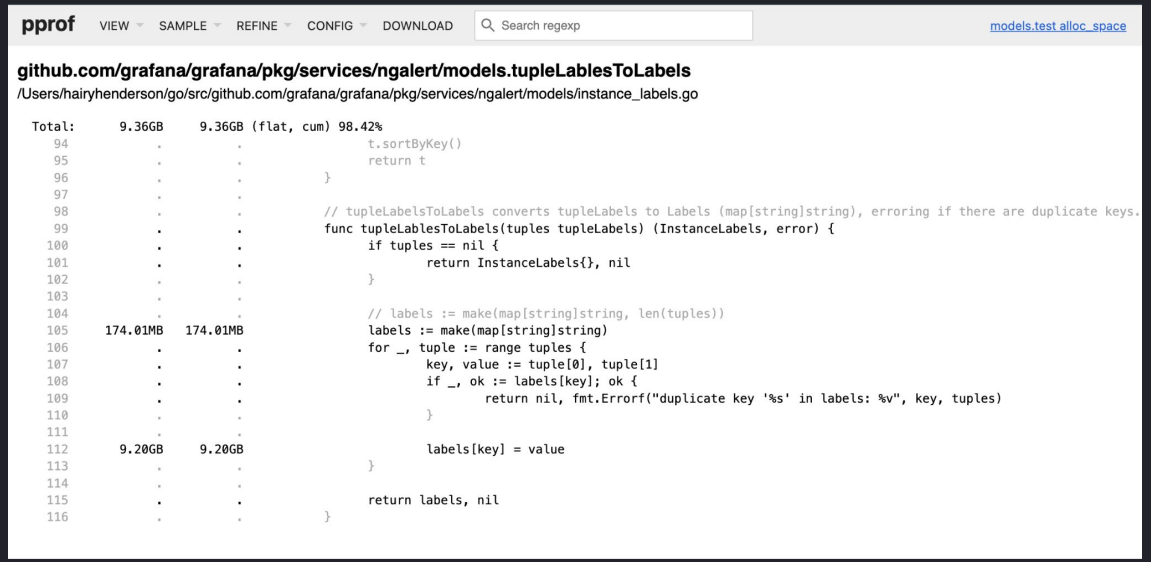
github.com/grafana/grafana/pkg/tsdb/influxdb

```
// tupleLabelsToLabels converts tupleLabels to Labels (map[string]string), erroring if there
are duplicate keys.
func tupleLablesToLabels(tuples tupleLabels) (InstanceLabels, error) {
  if tuples == nil {
    return InstanceLabels{}, nil
  }
  labels := make(map[string]string)
  for _, tuple := range tuples {
    if key, ok := labels[tuple[0]]; ok {
      return nil, fmt.Errorf("duplicate key '%v' in lables: %v", key, tuples)
    }
    labels[tuple[0]] = tuple[1]
  }
  return labels, nil
}
```

```go
func BenchmarkTupleLabelsToLabels(b *testing.B) {
    b.Run("10 labels", func(b *testing.B) {
        in := make(tupleLabels, 0, 10)
        for i := 0; i < 10; i++ {
            key := fmt.Sprintf("key%d", i)
            value := fmt.Sprintf("value%d", i)
            in = append(in, tupleLabel{key, value})
        }

        b.ResetTimer()

        for i := 0; i < b.N; i++ {
            _, err := tupleLablesToLabels(in)
            if err != nil {
                b.Fatal(err)
            }
        }
    })
```

```go
b.Run("1_000_000 labels", func(b *testing.B) {
    in := make(tupleLabels, 0, 1_000_000)
    for i := 0; i < 1_000_000; i++ {
        key := fmt.Sprintf("key%d", i)
        value := fmt.Sprintf("value%d", i)
        in = append(in, tupleLabel{key, value})
    }

    b.ResetTimer()

    for i := 0; i < b.N; i++ {
        _, err := tupleLablesToLabels(in)
        if err != nil {
            b.Fatal(err)
        }
    }
})
```

```
└─± go test -count=1 -benchmem -run=^$ -memprofile mem.p -bench ^Benchmark ./pkg/services/ngalert/models
goos: darwin
goarch: arm64
pkg: github.com/grafana/grafana/pkg/services/ngalert/models
BenchmarkTupleLabelsToLabels/10_labels-12                2627703               439.5 ns/op           918 B/op          3 allocs/op
BenchmarkTupleLabelsToLabels/100_labels-12                181608              6710 ns/op           10508 B/op         11 allocs/op
BenchmarkTupleLabelsToLabels/10_000_labels-12               1484            758730 ns/op         1275273 B/op        214 allocs/op
BenchmarkTupleLabelsToLabels/1_000_000_labels-12               6         168927542 ns/op       162557768 B/op      38218 allocs/op
PASS
ok      github.com/grafana/grafana/pkg/services/ngalert/models  6.695s
```



```
labels := make(map[string]string, len(tuples))
```

```
└─± go test -count=1 -benchmem -run=^$ -memprofile mem.p -bench ^Benchmark ./pkg/services/ngalert/models
goos: darwin
goarch: arm64
pkg: github.com/grafana/grafana/pkg/services/ngalert/models
BenchmarkTupleLabelsToLabels/10_labels-12                3777117               317.4 ns/op           630 B/op          2 allocs/op
BenchmarkTupleLabelsToLabels/100_labels-12                285547              4157 ns/op            5449 B/op          4 allocs/op
BenchmarkTupleLabelsToLabels/10_000_labels-12               2772            424409 ns/op          598089 B/op          3 allocs/op
BenchmarkTupleLabelsToLabels/1_000_000_labels-12              12          87055326 ns/op        75759688 B/op          3 allocs/op
PASS
ok      github.com/grafana/grafana/pkg/services/ngalert/models  6.145s
```

pprof VIEW SAMPLE REFINE CONFIG DOWNLOAD Search regexp

github.com/grafana/grafana/pkg/services/ngalert/models.tupleLablesToLabels

/Users/hairyhenderson/go/src/github.com/grafana/grafana/pkg/services/ngalert/models/instance_labels.go

```
Total:      6.89GB     6.89GB (flat, cum) 97.75%
   99            .          .           func tupleLablesToLabels(tuples tupleLabels) (InstanceLabels, error) {
  100            .          .                   if tuples == nil {
  101            .          .                           return InstanceLabels{}, nil
  102            .          .                   }
  103            .          .
  104      6.71GB     6.71GB                   labels := make(map[string]string, len(tuples))
  105            .          .                   for _, tuple := range tuples {
  106            .          .                           key, value := tuple[0], tuple[1]
  107            .          .                           if _, ok := labels[key]; ok {
  108            .          .                                   return nil, fmt.Errorf("duplicate key '%s' in labels: %v", key, tuples)
  109            .          .                           }
  110            .          .
  111    186.05MB   186.05MB                           labels[key] = value
  112            .          .                   }
  113            .          .
  114            .          .                   return labels, nil
  115            .          .           }
```

CPU

Memory

Allocations

```
$ benchstat before.txt after.txt
goos: darwin
goarch: arm64
pkg: github.com/grafana/grafana/pkg/services/ngalert/models
                                          |  before.txt   |             after.txt              |
                                          |    sec/op     |   sec/op     vs base               |
TupleLabelsToLabels/10_labels-12            443.1n ±  4%   327.1n ± 7%  -26.18% (p=0.001 n=7)
TupleLabelsToLabels/100_labels-12           6.553µ ±  4%   4.263µ ± 1%  -34.95% (p=0.001 n=7)
TupleLabelsToLabels/10_000_labels-12        757.5µ ± 12%   429.6µ ± 1%  -43.29% (p=0.001 n=7)
TupleLabelsToLabels/1_000_000_labels-12     162.58m ±  1%  86.02m ± 1%  -47.09% (p=0.001 n=7)
geomean                                     137.5µ         84.73µ       -38.39%

                                          |  before.txt   |             after.txt              |
                                          |     B/op      |     B/op      vs base              |
TupleLabelsToLabels/10_labels-12              918.0 ± 0%     630.0 ± 0%  -31.37% (p=0.001 n=7)
TupleLabelsToLabels/100_labels-12           10.260Ki ± 0%  5.321Ki ± 0%  -48.13% (p=0.001 n=7)
TupleLabelsToLabels/10_000_labels-12        1245.2Ki ± 0%  584.1Ki ± 0%  -53.10% (p=0.001 n=7)
TupleLabelsToLabels/1_000_000_labels-12     155.03Mi ± 0%  72.25Mi ± 0%  -53.40% (p=0.001 n=7)
geomean                                     206.5Ki        109.1Ki       -47.19%

                                          |   before.txt    |             after.txt              |
                                          |   allocs/op     | allocs/op   vs base                 |
TupleLabelsToLabels/10_labels-12              3.000 ± 0%      2.000 ± 0%  -33.33% (p=0.001 n=7)
TupleLabelsToLabels/100_labels-12            11.000 ± 0%      4.000 ± 0%  -63.64% (p=0.001 n=7)
TupleLabelsToLabels/10_000_labels-12        214.000 ± 0%      3.000 ± 0%  -98.60% (p=0.001 n=7)
TupleLabelsToLabels/1_000_000_labels-12   38237.000 ± 0%      3.000 ± 0%  -99.99% (p=0.001 n=7)
geomean                                     128.2           2.913       -97.73%
```

In conclusion

- Have “Mechanical Sympathy”
- Have unit tests
- Change one thing at a time
- Small changes makes big difference
- Benchmarking is easy
- We have all the tools in place

“

Premature optimization is the root of all evil (or at least most of it) in programming.

”

Donald Knuth

Further Reading

- https://playground.flamegraph.com/playground
- High Performance Go Workshop
- From slow to SIMD: A Go optimization story
- High-Performance JSON Parsing in Go
- go-perfbook

Acknowledgments

- Dave Cheney https://dave.cheney.net/
  - High Performance Go Workshop
- Bryan Boreham https://github.com/bboreham
- Code coloring https://www.ray.so

Q/A