Daily code optimisation using benchmarks and profiling in Golang

Karthic Rao
@hackintoshrao
medium.com/@hackintoshrao

Recipe for code optimisation

- Write benchmark
- CPU Profile
- Memory Profile
- Blocking profile
- Other tricks

Writing benchmarks

- Comes bundled with Golang testing package
- And its easy (overcoming the psychological barrier)
- Compare performance easily

```go
package gobench

import (
        "testing"
)

func BenchmarkGoMapAdd(b *testing.B) {

        for i := 0; i < b.N; i++ {
                GoMapAdd()
        }

}
func BenchmarkGoStructAdd(b *testing.B) {

        for i := 0; i < b.N; i++ {
                GoStructAdd()
        }

}
```

```go
package gobench

func GoMapAdd() {
        m := map[int]int{0: 0, 1: 1}
        _ = m[0] + m[1]
}

func GoStructAdd() {
        m := struct{ a, b int }{0, 1}
        _ = m.a + m.b
}
```

$ go test -bench=.

```
BenchmarkGoMapAdd     5000000               286 ns/op
BenchmarkGoStructAdd 2000000000             0.56 ns/op
```

```
type add struct {
        Sum int
}

func handleStructAdd(w http.ResponseWriter, r *http.Request) {

        var html bytes.Buffer
        first, second := r.FormValue("first"), r.FormValue("second")
        one, err := strconv.Atoi(first)
        if err != nil {
                http.Error(w, err.Error(), 500)
        }
        two, err := strconv.Atoi(second)
        if err != nil {
                http.Error(w, err.Error(), 500)
        }
        m := struct{ a, b int }{one, two}
        structSum := add{Sum: m.a + m.b}

        t, err := template.ParseFiles("template.html")
        if err != nil {
                http.Error(w, err.Error(), 500)
        }
        err = t.Execute(&html, structSum)

        if err != nil {
                http.Error(w, err.Error(), 500)
        }
        w.Header().Set("Content-Type", "text/html; charset=utf-8")
        w.Write([]byte(html.String()))
}
func main() {

        http.HandleFunc("/struct", handleStructAdd)
        log.Fatal(http.ListenAndServe("127.0.0.1:8081", nil))
}
```

```
func TestHandleStructAdd(t *testing.T) {

        r := request(t, "/?first=20&second=30")

        rw := httptest.NewRecorder()

        handleStructAdd(rw, r)
        if rw.Code == 500 {
                t.Fatal("Internal server Error: " + rw.Body.String())
        }
        if rw.Body.String() != "<h2>Here is the sum 50</h2>" {
                t.Fatal("Expected " + rw.Body.String())
        }

}
func BenchmarkHandleStructAdd(b *testing.B) {
        r := request(b, "/?first=20&second=30")
        for i := 0; i < b.N; i++ {
                rw := httptest.NewRecorder()
                handleStructAdd(rw, r)
        }

}

func request(t testing.TB, url string) *http.Request {
        req, err := http.NewRequest("GET", url, nil)
        if err != nil {
                t.Fatal(err)
        }
        return req
}
```

$ go test run=xxx -bench=.

```
BenchmarkHandleStructAdd-4    30000        40219 ns/op
```

Profiling from the benchmarks

- go test -run=^$ -bench=. -cpuprofile=profile.cpu
- 2 new files are created.
- A binary ending with .test and the profile info in profile.cpu
- go tool pprof <binary> <profile file>
- go tool pprof simple-http-benchmark.test profile.cpu
- $ go test -run=xxx -bench=. | tee bench0

The interactive profiler

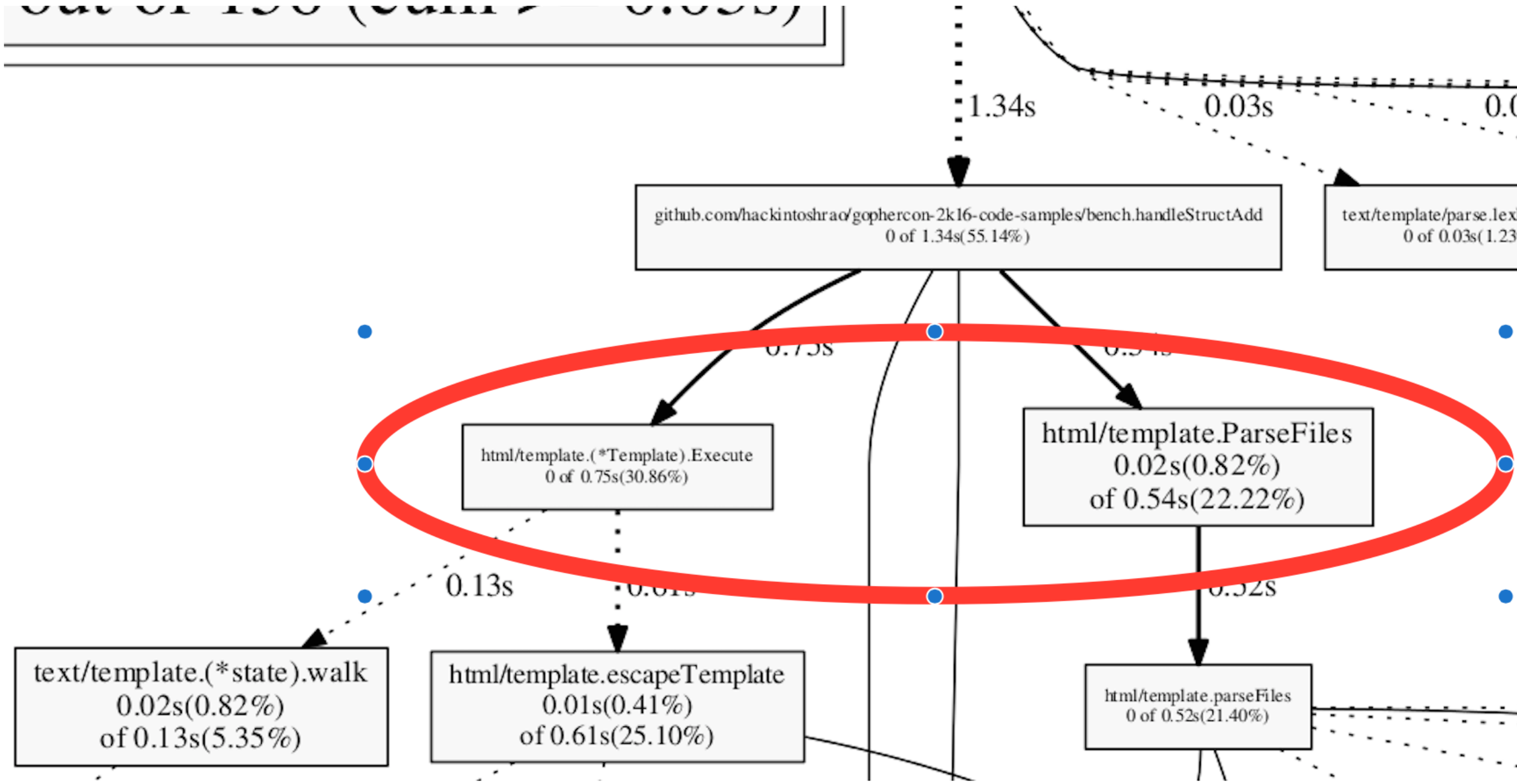
- topN

```
(pprof) top20
1340ms of 2230ms total (60.09%)
Dropped 78 nodes (cum <= 11.15ms)
Showing top 20 nodes out of 154 (cum >= 220ms)
      flat  flat%   sum%        cum   cum%
     320ms 14.35% 14.35%      320ms 14.35%  runtime/internal/atomic.Xchg
     120ms  5.38% 19.73%      120ms  5.38%  runtime/internal/atomic.Xadd
     110ms  4.93% 24.66%      850ms 38.12%  runtime.findrunnable
      70ms  3.14% 27.80%      350ms 15.70%  runtime.mallocgc
      70ms  3.14% 30.94%      360ms 16.14%  runtime.mapassign1
      70ms  3.14% 34.08%       70ms  3.14%  runtime.usleep
      50ms  2.24% 36.32%       50ms  2.24%  runtime.acquirep1
      50ms  2.24% 38.57%       50ms  2.24%  runtime.heapBitsSetType
      50ms  2.24% 40.81%       80ms  3.59%  runtime.heapBitsSweepSpan
      50ms  2.24% 43.05%       90ms  4.04%  runtime.scanobject
      50ms  2.24% 45.29%       50ms  2.24%  runtime.stringiter2
```

- top —cum

```
(pprof) top --cum
0.13s of 2.23s total ( 5.83%)
Dropped 78 nodes (cum <= 0.01s)
Showing top 10 nodes out of 154 (cum >= 0.68s)
```

go tool pprof —pdf bench.test cpu.out > cpu0.pdf



```
var templates = template.Must(template.ParseFiles("template.html"))

func handleStructAdd(w http.ResponseWriter, r *http.Request) {

        var html bytes.Buffer
        first, second := r.FormValue("first"), r.FormValue("second")
        one, err := strconv.Atoi(first)
        if err != nil {
                http.Error(w, err.Error(), 500)
        }
        two, err := strconv.Atoi(second)
        if err != nil {
                http.Error(w, err.Error(), 500)
        }
        m := struct{ a, b int }{one, two}
        structSum := add{Sum: m.a + m.b}

        err = templates.Execute(&html, structSum)

        if err != nil {
                http.Error(w, err.Error(), 500)
        }
        w.Header().Set("Content-Type", "text/html; charset=utf-8")
        w.Write([]byte(html.String()))
}

func main() {

        http.HandleFunc("/struct", handleStructAdd)
```

Now compare the performance

- go test -run=xxx -bench=. | tee bench1
- Use benchcmp for performance comparison

```
$ benchcmp bench0 bench1

benchmark                    old ns/op  new ns/op  delta
BenchmarkHandleStructAdd-4 41808      3965       -90.52%
```

Build your own tools

- Want to build your own tools around Golang benchmark data?
- Use benchmark parse

tools: golang.org/x/tools/benchmark/parse Index | Files

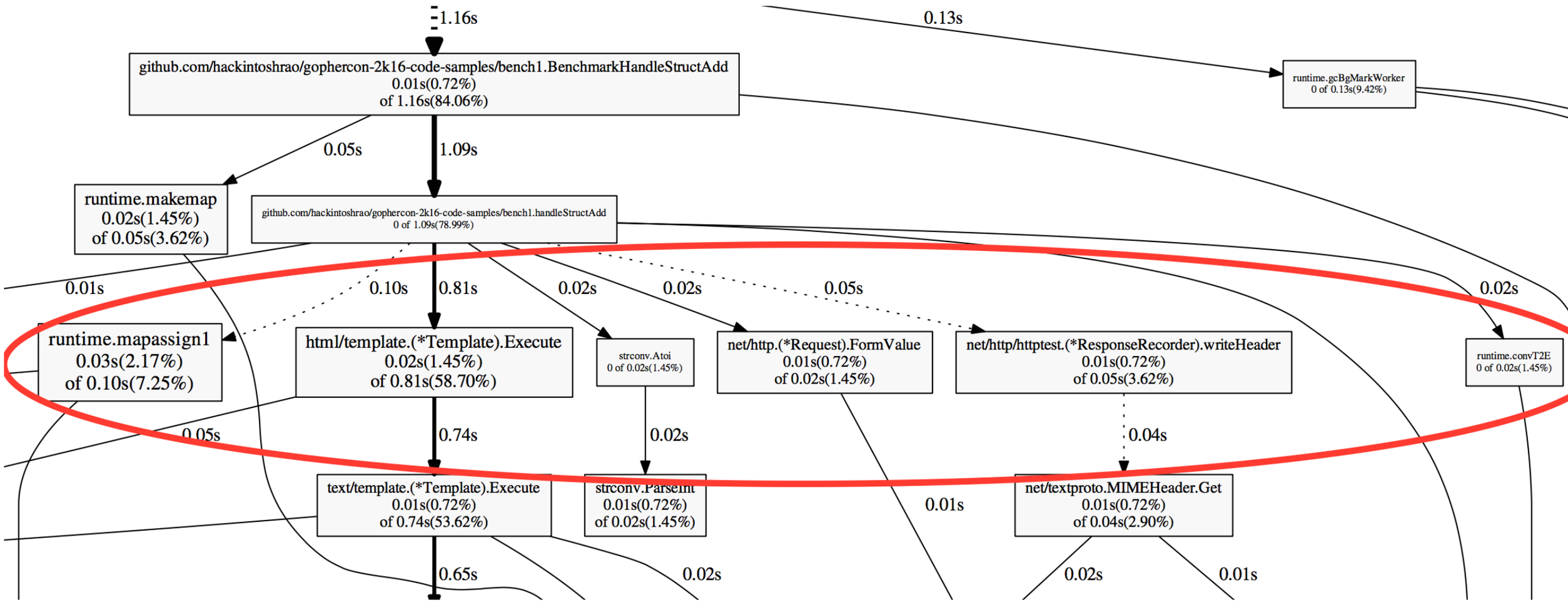
package parse

```
import "golang.org/x/tools/benchmark/parse"
```

Package parse provides support for parsing benchmark results as generated by 'go test -bench'.

Index

go tool pprof --pdf bench.test cpu.out > cpu1.pdf



$go tool pprof bench.test cpu.out

list handleStructAdd

```
(pprof) list handleStructAdd
Total: 1.39s
ROUTINE ======================== github.com/hackintoshrao/gophercon-2k16-code-samples/bench1.handleStructAdd in /home/hac
ercon-2k16-code-samples/bench1/simple_add.go
      20ms      1.12s (flat, cum) 80.58% of Total
         .          .     14:
         .          .     15:var templates = template.Must(template.ParseFiles("template.html"))
         .          .     16:
         .          .     17:func handleStructAdd(w http.ResponseWriter, r *http.Request) {
         .          .     18:
         .       10ms     19:   var html bytes.Buffer
         .          .     20:   first, second := r.FormValue("first"), r.FormValue("second")
         .       10ms     21:   one, err := strconv.Atoi(first)
         .          .     22:   if err != nil {
         .          .     23:           http.Error(w, err.Error(), 500)
         .          .     24:   }
         .          .     25:   two, err := strconv.Atoi(second)
         .          .     26:   if err != nil {
         .          .     27:           http.Error(w, err.Error(), 500)
         .          .     28:   }
         .          .     29:   m := struct{ a, b int }{one, two}
         .          .     30:   structSum := add{Sum: m.a + m.b}
         .          .     31:
      10ms      900ms     32:   err = templates.Execute(&html, structSum)
         .          .     33:
         .          .     34:   if err != nil {
         .          .     35:           http.Error(w, err.Error(), 500)
         .          .     36:   }
         .      140ms     37:   w.Header().Set("Content-Type", "text/html; charset=utf-8")
      10ms       60ms     38:   w.Write([]byte(html.String()))
         .          .     39:}
         .          .     40:
         .          .     41:func main() {
         .          .     42:
         .          .     43:   http.HandleFunc("/struct", handleStructAdd)
(pprof)
```

$go test -run=xxx -bench=. -cpuprofile=cpu.out

$go tool pprof bench.test cpu.out

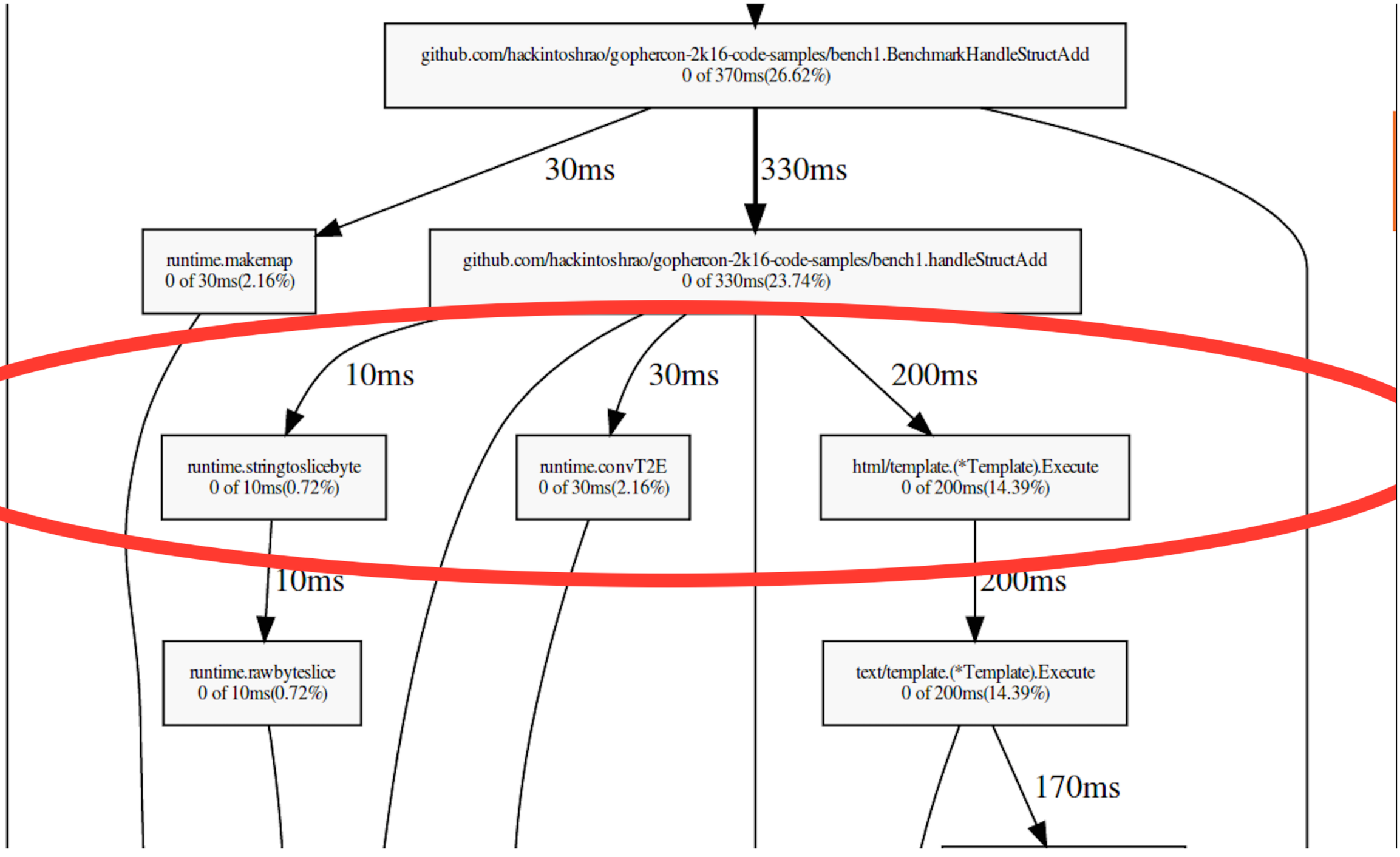
Top10

```
(pprof) top10
660ms of 1390ms total (47.48%)
Showing top 10 nodes out of 136 (cum >= 690ms)
      flat  flat%   sum%        cum   cum%
     170ms 12.23% 12.23%      490ms 35.25%  runtime.mallocgc
      90ms  6.47% 18.71%       90ms  6.47%  runtime.(*mspan).sweep.func1
      70ms  5.04% 23.74%      430ms 30.94%  reflect.Value.call
      70ms  5.04% 28.78%       70ms  5.04%  runtime.heapBitsSetType
      60ms  4.32% 33.09%       70ms  5.04%  runtime.mapaccess1_faststr
      40ms  2.88% 35.97%       40ms  2.88%  net/textproto.CanonicalMIMEHeaderKey
      40ms  2.88% 38.85%       40ms  2.88%  runtime.deferreturn
      40ms  2.88% 41.73%      130ms  9.35%  runtime.heapBitsSweepSpan
      40ms  2.88% 44.60%       40ms  2.88%  runtime.memclr
      40ms  2.88% 47.48%      690ms 49.64%  text/template.(*state).walk
(pprof)
```

Solving the Mallocgc challenge

- Mallocgc is Golang garbage collector
- GC sweeps the heap allocations once it starts spiking up
- But how to identify the reason behind the high CPU usage of some these runtime functions ?
- Let’s say I want to know about the functions which are contributing highly for the mallogc invocation?

Removing the noise
web mallocgc



Again, reduce the noise in the profiling graph

- go tool pprof --nodefraction=0.2 bench.test pro.cpu

Tools in Testing.B

```
type B
    func (c *B) Error(args ...interface{})
    func (c *B) Errorf(format string, args ...interface{})
    func (c *B) Fail()
    func (c *B) FailNow()
    func (c *B) Failed() bool
    func (c *B) Fatal(args ...interface{})
    func (c *B) Fatalf(format string, args ...interface{})
    func (c *B) Log(args ...interface{})
    func (c *B) Logf(format string, args ...interface{})
    func (b *B) ReportAllocs()
    func (b *B) ResetTimer()
    func (b *B) RunParallel(body func(*PB))
    func (b *B) SetBytes(n int64)
    func (b *B) SetParallelism(p int)
    func (c *B) Skip(args ...interface{})
    func (c *B) SkipNow()
    func (c *B) Skipf(format string, args ...interface{})
    func (c *B) Skipped() bool
    func (b *B) StartTimer()
    func (b *B) StopTimer()
type BenchmarkResult
    func Benchmark(f func(b *B)) BenchmarkResult
    func (r BenchmarkResult) AllocedBytesPerOp() int64
    func (r BenchmarkResult) AllocsPerOp() int64
```

Memory profiling

- Use *testing.B.ReportAlloc()

```
func BenchmarkHandleStructAdd(b *testing.B) {
        b.ReportAllocs()
        r := request(b, "/?first=20&second=30")
        for i := 0; i < b.N; i++ {
                rw := httptest.NewRecorder()
                handleStructAdd(rw, r)
        }

}
```

Running the benchmark

```
hackintosh@Adori: ~/Code/go/src/github.com/hackintoshrao/gophercon-2k16-
h@Adori:~/Code/go/src/github.com/hackintoshrao/gophercon-2k16-code-samples/bench3$ go test -bench=.
HandleStructAdd-4        500000              3733 ns/op            1080 B/op         18 allocs/op
ithub.com/hackintoshrao/gophercon-2k16-code-samples/bench3     1.926s
h@Adori:~/Code/go/src/github.com/hackintoshrao/gophercon-2k16-code-samples/bench3$
```

Memory profiler

- $go test -run=^$ -bench=. -memprofile=mem0.out
- — inuse_objects (show count by number of allocations)
- — alloc_space (shows the total allocation size0
- $go tool pprof --alloc_space bench.test mem0.out

Find the top cumulative memory consumers

```
pprof) top --cum
03.54MB of 294.04MB total (69.22%)
howing top 10 nodes out of 25 (cum >= 90.51MB)
     flat  flat%   sum%        cum   cum%
    60MB 20.41% 20.41%   294.04MB   100%  github.com/hackintoshrao/gophercon-2k16-code-samples/bench3.BenchmarkHandleStructAdd
       0     0% 20.41%   294.04MB   100%  runtime.goexit
       0     0% 20.41%   294.04MB   100%  testing.(*B).launch
       0     0% 20.41%   294.04MB   100%  testing.(*B).runN
    39MB 13.26% 33.67%   234.04MB 79.59%  github.com/hackintoshrao/gophercon-2k16-code-samples/bench3.handleStructAdd
       0     0% 33.67%   102.01MB 34.69%  html/template.(*Template).Execute
 11.50MB  3.91% 37.58%   102.01MB 34.69%  text/template.(*Template).Execute
       0     0% 37.58%    93.03MB 31.64%  net/http.Header.Set
 93.03MB 31.64% 69.22%    93.03MB 31.64%  net/textproto.MIMEHeader.Set
       0     0% 69.22%    90.51MB 30.78%  text/template.(*state).evalCommand
```

```
(pprof) top10
294.04MB of 294.04MB total (  100%)
Showing top 10 nodes out of 25 (cum >= 2.50MB)
      flat  flat%   sum%        cum   cum%
   93.03MB 31.64% 31.64%    93.03MB 31.64%  net/textproto.MIMEHeader.Set
      61MB 20.75% 52.38%    79.51MB 27.04%  reflect.Value.call
      60MB 20.41% 72.79%   294.04MB   100%  github.com/hackintoshrao/gophercon-2k16-code-samples/bench3.BenchmarkHandleStructAdd
      39MB 13.26% 86.06%   234.04MB 79.59%  github.com/hackintoshrao/gophercon-2k16-code-samples/bench3.handleStructAdd
   11.50MB  3.91% 89.97%   102.01MB 34.69%  text/template.(*Template).Execute
    9.50MB  3.23% 93.20%     9.50MB  3.23%  reflect.unsafe_New
       8MB  2.72% 95.92%    87.51MB 29.76%  text/template.(*state).evalCall
    6.50MB  2.21% 98.13%        9MB  3.06%  reflect.MakeSlice
       3MB  1.02% 99.15%        3MB  1.02%  reflect.(*structType).Field
    2.50MB  0.85%   100%     2.50MB  0.85%  reflect.unsafe_NewArray
(pprof) web MakeSlice
(pprof)
```

```
 makeslice
t handle
04MB
==================== github.com/hackintoshrao/gophercon-2k16-code-samples/bench3.handleStructAdd in /home/hackintosh/Code/g
code-samples/bench3/simple_add.go
  234.04MB (flat, cum) 79.59% of Total
         .     14:
         .     15:var templates = template.Must(template.ParseFiles("template.html"))
         .     16:
         .     17:func handleStructAdd(w http.ResponseWriter, r *http.Request) {
         .     18:
      29MB     19:   var html bytes.Buffer
         .     20:   first, second := r.FormValue("first"), r.FormValue("second")
         .     21:   one, err := strconv.Atoi(first)
         .     22:   if err != nil {
         .     23:           http.Error(w, err.Error(), 500)
         .     24:   }
         .     25:   two, err := strconv.Atoi(second)
         .     26:   if err != nil {
         .     27:           http.Error(w, err.Error(), 500)
         .     28:   }
         .     29:   m := struct{ a, b int }{one, two}
         .     30:   structSum := add{Sum: m.a + m.b}
         .     31:
  103.51MB     32:   err = templates.Execute(&html, structSum)
         .     33:
         .     34:   if err != nil {
         .     35:           http.Error(w, err.Error(), 500)
         .     36:   }
   93.03MB     37:   w.Header().Set("Content-Type", "text/html; charset=utf-8")
    8.50MB     38:   w.Write([]byte(html.String()))
         .     39:}
         .     40:
         .     41:func main() {
         .     42:
         .     43:   http.HandleFunc("/struct", handleStructAdd)
```

```
.          .     43:   http.HandleFunc("/struct", handleStructAdd)
list MIMEHeader.Set
94.04MB
======================== net/textproto.MIMEHeader.Set in /usr/local/go/src/net/textproto/header.go
MB    93.03MB (flat, cum) 31.64% of Total
 .          .     17:
 .          .     18:// Set sets the header entries associated with key to
 .          .     19:// the single element value.  It replaces any existing
 .          .     20:// values associated with key.
 .          .     21:func (h MIMEHeader) Set(key, value string) {
MB    93.03MB     22:   h[CanonicalMIMEHeaderKey(key)] = []string{value}
 .          .     23:}
 .          .     24:
 .          .     25:// Get gets the first value associated with the given key.
 .          .     26:// If there are no values associated with the key, Get returns "".
 .          .     27:// Get is a convenience method.  For more complex queries,
```

The modification

- func (t *Template) Execute(wr io.Writer, data interface{}) (err error)

```
func handleStructAdd(w http.ResponseWriter, r *http.Request) {

        first, second := r.FormValue("first"), r.FormValue("second")
        one, err := strconv.Atoi(first)
        if err != nil {
                http.Error(w, err.Error(), 500)
        }
        two, err := strconv.Atoi(second)
        if err != nil {
                http.Error(w, err.Error(), 500)
        }
        m := struct{ a, b int }{one, two}
        structSum := add{Sum: m.a + m.b}

        err = templates.Execute(w, structSum)

        if err != nil {
                http.Error(w, err.Error(), 500)
        }
```

Benchmark and compare

```
$go test -run=^$ -bench=. | tee profile.2

$benchcmp profile.1 profile.2
```

```
benchmark                    old ns/op     new ns/op     delta
BenchmarkHandleStructAdd-4   3853          4419          +14.69%

benchmark                    old allocs     new allocs     delta
BenchmarkHandleStructAdd-4   18             16             -11.11%

benchmark                    old bytes     new bytes     delta
BenchmarkHandleStructAdd-4   1080          936           -13.33%
```

Other tools

1. Golang blocking profiler
2. sync.Pool , to pool and reuse resources
3. Garbage collector tracer
4. Memory Allocator tracer
5. Scheduler tracer
6. runtime.ReadMemstats

Thank you

@hackintoshrao

medium.com/@hackintoshrao