

# Flamegraphs, Tracers...

Muhammad Falak R Wani

IIIT-D

# Agenda

- Profiling
- Flamegraphs
- Tracing

Code and Slides:

[github.com/mfrw/flamegraphs-go\\_meetup](https://github.com/mfrw/flamegraphs-go_meetup) (#ZgotmplZ)

NOTE: The code was delbrately written as badly as possible.  
The only intention was to show how to use tools on bad code.

# Profiling

Periodic sampling of stack traces.

## **pprof**

Go has powerful profiling built into the runtime

- CPU profiles (stack sampling)
- Heap profiles using allocation profiling
- and more (block profiles, traces, etc)

Accessible through **pprof** and **runtime** packages.

Started off as [google/gperftools](https://github.com/google/gperftools) (C++)

# pprof

- Tests/Bechmarks: -cpuprofile & -memprofile flags

```
go test . -bench=. -cpuprofile prof.cpu  
go tool pprof [binary] prof.cpu
```

- Manual Instrumentation:

```
runtime.StartCPUProfile & runtime.WriteHeapProfile
```

- Import:

```
import _ net/http/pprof (adds /debug/pprof endpoints)  
  
http.ListenAndServe(":8080", nil) || If you don't have a listener  
  
go tool pprof -seconds 5 http://localhost:8080/debug/pprof/profile
```

# Profile Types

```
" Press ? for help
.. (up a dir)
/Users/mfrw/go/src/runtime
├─ cgo/
├─ debug/
├─ internal/
├─ msan/
├─ pprof/
│   ├─ internal/
│   └─ testdata/
│       elf.go
│       label.go
│       label_test.go
│       map.go
│       mprof_test.go
│       pprof.go
│       pprof_test.go
│       proto.go
│       proto_test.go
│       protobuf.go
│       protomem.go
│       protomem_test.go
│       runtime.go
│       runtime_test.go
├─ race/
├─ testdata/
├─ trace/
│   alg.go
│   asm.s
│   asm_386.s
│   asm_amd64.s
│   asm_amd64p32.s
│   asm_arm.s
│   asm_arm64.s
│   asm_mips64x.s
│   asm_mipsx.s
│   asm_ppc64x.h
│   asm_ppc64x.s
│   asm_s390x.s
│   atomic_arm64.s
│   atomic_mips64x.s
│   atomic_mipsx.s
│   atomic_pointer.go
└─ 127 }
128
129 // profiles records all registered profiles.
130 var profiles struct {
131     mu sync.Mutex
132     m map[string]*Profile
133 }
134
135 var goroutineProfile = &Profile{
136     name: "goroutine",
137     count: countGoroutine,
138     write: writeGoroutine,
139 }
140
141 var threadcreateProfile = &Profile{
142     name: "threadcreate",
143     count: countThreadCreate,
144     write: writeThreadCreate,
145 }
146
147 var heapProfile = &Profile{
148     name: "heap",
149     count: countHeap,
150     write: writeHeap,
151 }
152
153 var blockProfile = &Profile{
154     name: "block",
155     count: countBlock,
156     write: writeBlock,
157 }
158
159 var mutexProfile = &Profile{
160     name: "mutex",
161     count: countMutex,
162     write: writeMutex,
163 }
164
165 func lockProfiles() {
166     profiles.mu.Lock()
167     if profiles.m == nil {
168         // Initial built-in profiles.
169         profiles.m = map[string]*Profile{
170             "goroutine": goroutineProfile,
171             "heapProfile": heapProfile,
172             "mutexProfile": mutexProfile,
173             "profiles": profiles,
174             "threadcreateProfile": threadcreateProfile,
175         }
176     }
177 }
178
179 // Profile : struct
180 [fields]
181 -count : func() int
182 -m : map[interface{}][]uintptr
183 -mu : sync.Mutex
184 -name : string
185 -write : func(io.Writer, int) error
186 [methods]
187 +Add(value interface{}, skip int)
188 +Count() : int
189 +Name() : string
190 +Remove(value interface{})
191 +WriteTo(w io.Writer, debug int) : error
192 [functions]
193 +Lookup(name string) : *Profile
194 +NewProfile(name string) : *Profile
195
196 // keysByCount : struct
197 [fields]
198 -count : map[string]int
199 -keys : []string
200 [methods]
201 +Len() : int
202 +Less(i, j int) : bool
203 +Swap(i, j int)
204
205 // runtimeProfile : []runtime.StackRecord
206 [methods]
207 +Len() : int
208 +Stack(i int) : []uintptr
209
210 // stackProfile : [][]uintptr
211 [methods]
212 +Len() : int
213 +Stack(i int) : []uintptr
214
215 // countProfile : interface
216 [methods]
```

NERD NORMAL +0 ~0 -0 P master <c/runtime/pprof/pprof.go heapProfile go utf-8[unix] 16% 147/896 ln : 9 Tagbar Name pprof.go

# Tools

- Native

```
go tool pprof [binary] prof.cpu
```

- [google/pprof](#)

```
go get github.com/google/pprof
```

The profile is a **gzipped** protbuff. So you can write your own tooling around it.

- For more information:

```
runtime/pprof/proto.go
```

# Flamegraph

- What ?

FlameGraph (Brendan Gregg) (<http://www.brendangregg.com/flamegraphs.html>)

ACMQ FlameGraph (<https://queue.acm.org/detail.cfm?id=2927301>)

- Old way: **go-torch** (UBER)

```
go-torch prof.cpu [generates torch.svg]
```

- Now: **pprof**

Flame Graph support on Web UI #188 (<https://github.com/google/pprof/pull/188>)

```
pprof -http :8080 [binary] prof.cpu
```

# Tracing

Unlike profiling, trace events exactly. Track down latency issues, poor parallelism etc.

- Direct:

```
import runtime/trace
trace.Start & trace.Stop
```

- Testing:

```
go test -trace ...
```

- Import:

```
import _ net/http/pprof
```

It has an intuitive UI (Chrome only)



## MISC

Please feel free to email/ping whatever works for you. I am not an expert and would really love to learn more. (help if possible)

- GOGC : control the gc (GARBAGE COLLECTOR)
- httptrace : tracing http on steroids
- GOSSAFUNC : SSA dump of the function (all the phases)
- ssadump : Displaying and interpreting the SSA form of Go programs.

# Thank you

Muhammad Falak R Wani

IIIT-D

[@iamfrw](http://twitter.com/iamfrw) (<http://twitter.com/iamfrw>)

[falakreyaz@gmail.com](mailto:falakreyaz@gmail.com) (<mailto:falakreyaz@gmail.com>)