Programming in Go

Matt Holiday
Christmas 2020

Profiling

Web proxy example

We're going to write a proxy for the "todo" JSON demo server

```go
package main

import (
    "encoding/json"
    "fmt"
    "io/ioutil"
    "log"
    "net/http"

    _ "net/http/pprof"
)

const url = "https://jsonplaceholder.typicode.com/todos/"
```

Web proxy example

```go
type todo struct {
    UserID    int    `json:"userID"`
    ID        int    `json:"id"`
    Title     string `json:"title"`
    Completed bool   `json:"completed"`
}

var mark = map[bool]string{
    false: " ",
    true:  "x",
}

func handler(w http.ResponseWriter, r *http.Request) {
    var item todo

    req, _ := http.NewRequest("GET", url+r.URL.Path[1:], nil)
    . . .
```

Web proxy example

```
. . .

tr := &http.Transport{}
cli := &http.Client{Transport: tr}
resp, err := cli.Do(req)

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

if resp.StatusCode != http.StatusOK {
    http.NotFound(w, r)
    return
}

body, _ := ioutil.ReadAll(resp.Body)
. . .
```

Web proxy example

```go
    . . .

    if err := json.Unmarshal(body, &item); err != nil {
        http.Error(w, err.Error(),
            http.StatusInternalServerError)
        return
    }

    fmt.Fprintf(w, "[%s] %d - %s\n", mark[item.Completed],
        item.ID, item.Title)
}

func main() {
    http.HandleFunc("/", handler)

    log.Fatal(http.ListenAndServe(":8080", nil))
}
```

Running it

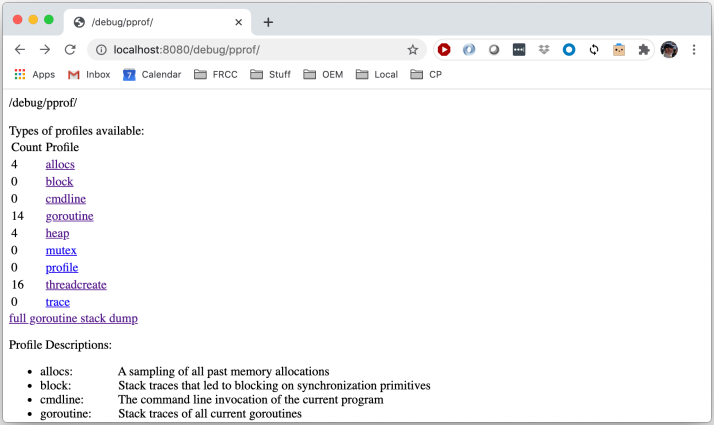
We can now open `http://localhost:8080/debug/pprof` which is the default route

If we look at goroutines, we'll see 4 goroutines initially, but as we exercise the server, that number will grow

```
$ curl http://localhost:8080/1
[ ] 1 - delectus aut autem

$ curl http://localhost:8080/8
[x] 8 - quo adipisci enim quam ut ab

$ curl http://localhost:8080/14
[x] 14 - repellendus sunt dolores architecto voluptatum

$ curl http://localhost:8080/1133
404 page not found
```

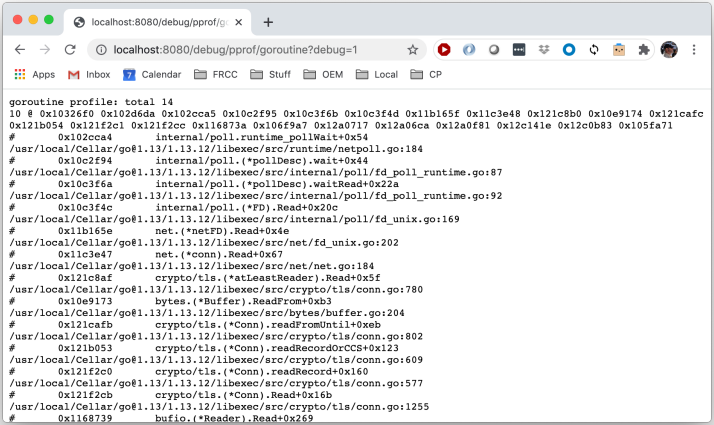
Main pprof page

This is what including `pprof` provides for us



Goroutine page

In this view we see a list of (hanging) goroutines



Fixing it

We'll see that all the goroutines are handling HTTP sockets

And the allocations happen where we read from the response body but forgot to close it

```go
// we need this to recover the socket

defer resp.Body.Close()

if resp.StatusCode != http.StatusOK {
    http.NotFound(w, r)
    return
}

body, _ := ioutil.ReadAll(resp.Body)
```

Prometheus metrics

We can add a Prometheus client which exposes metrics
(they can be scraped by hitting `/metrics`)

```go
import ()
    "github.com/prometheus/client_golang/prometheus"
    "github.com/prometheus/client_golang/prometheus/promhttp"
)

var queries = prometheus.NewCounter(prometheus.CounterOpts{
    Name: "all_queries",
    Help: "How many queries we've received.",
})
```

Prometheus metrics

We add a line of code to the end of our handler

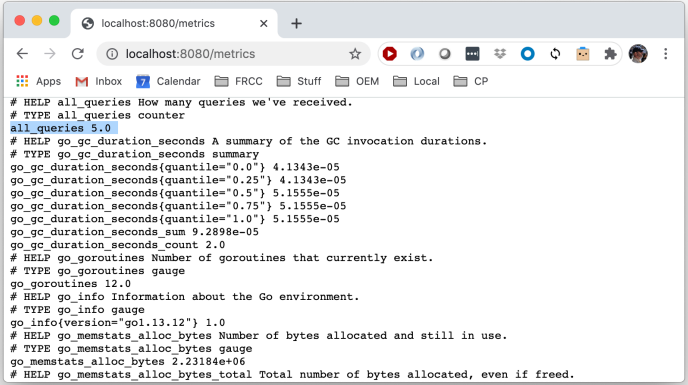
```
    . . .
    fmt.Fprintf(w, "[%s] %d - %s\n", mark[item.Completed], item.ID, item.Title)
    queries.Inc()
}
```

And an extra route in our main function

```
func main() {
    prometheus.MustRegister(queries)

    http.HandleFunc("/", handler)
    http.Handle("/metrics", promhttp.Handler())
    . . .
```

Metrics page

We get our metrics as well as a many system metrics



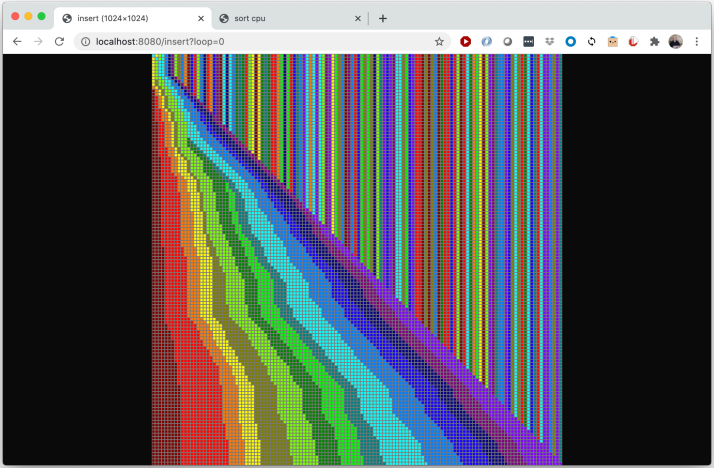
Prometheus metrics

Certain metrics will be a dead giveaway that we're leaking sockets

Not only will the goroutine count grow, but also the count of open file descriptors (FDs) *[Linux only — not on macOS]*

```
# HELP go_goroutines Number of goroutines that currently exist.
# TYPE go_goroutines gauge
go_goroutines 19.0

# HELP go_threads Number of OS threads created.
# TYPE go_threads gauge
go_threads 17.0

# HELP process_open_fds Number of open file descriptors.
# TYPE process_open_fds gauge
process_open_fds 19.0
```

Sort animation example



Sort animation example

The program animates various sort algorithms

- insertion sort: `insert`
- versions of quicksort: `qsort`, `qsortm`, `qsort3`, `qsorti`, `qsortf`

It creates an animated GIF with one frame for each row that changes (each step in the sort until it's done)

It must draw a square of an entry's color outlined in gray

- 1024 rows and columns
- each entry is a square 8 pixels on a side

How to profile

We must build the program as a binary: `go build .`

We're going to use `pprof` again and hit the endpoint
`http://localhost:8080/debug/pprof/profile`

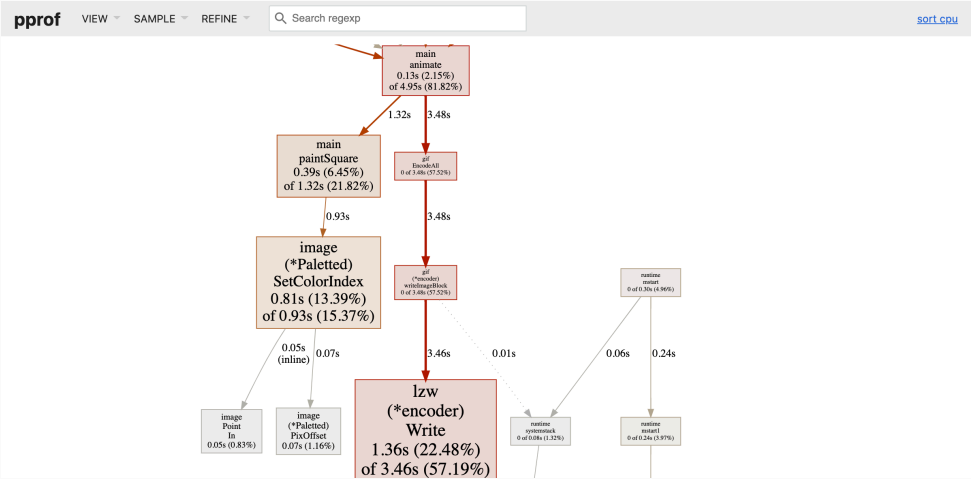
After 30 seconds it will download a profile file

Then we can run the tool: `go tool pprof <binary> <profile-file>`
in one of three ways

- interactive, with a prompt
- just get the top entries with `-top`
- open a browser with `-http=":6060"`

CPU flame graph

CPU profile

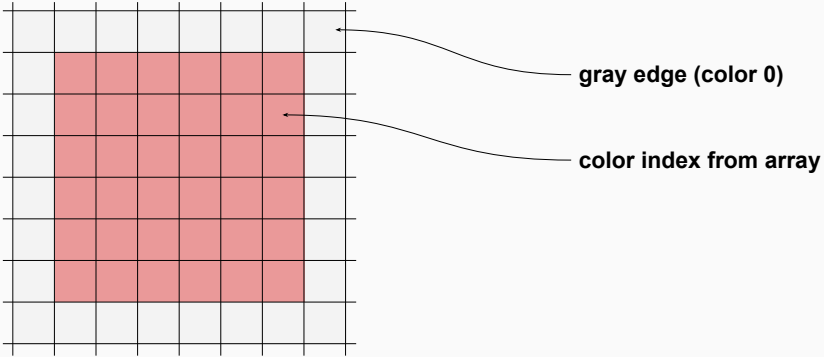

pprof
VIEW
SAMPLE
REFINE
Search regexp
sort cpu
main
animate
0.13s (2.15%)
of 4.95s (81.82%)
1.32s
3.48s
main
paintSquare
0.39s (6.45%)
of 1.32s (21.82%)
gif
EncodeAll
0 of 3.48s (57.52%)
0.93s
3.48s
image
(*Paletted)
SetColorIndex
0.81s (13.39%)
of 0.93s (15.37%)
gif
(*encoder)
writeImageBlock
0 of 3.48s (57.52%)
runtime
mstart
0 of 0.30s (4.96%)
0.05s
(inline)
0.07s
3.46s
0.01s
0.06s
0.24s
image
Point
In
0.05s (0.83%)
image
(*Paletted)
PixOffset
0.07s (1.16%)
lzw
(*encoder)
Write
1.36s (22.48%)
of 3.46s (57.19%)
runtime
systemstack
0 of 0.08s (1.32%)
runtime
mstart1
0 of 0.24s (3.97%)

CPU profile

```
$ go tool pprof -top sort profile-slow
File: sort
Type: cpu
Time: Jan 31, 2021 at 8:29am (MST)
Duration: 30s, Total samples = 6.05s (20.17%)
Showing nodes accounting for 5.99s, 99.01% of 6.05s total
Dropped 31 nodes (cum <= 0.03s)
      flat  flat%   sum%        cum   cum%
     1.68s 27.77% 27.77%      1.68s 27.77%  syscall.syscall
     1.36s 22.48% 50.25%      3.46s 57.19%  compress/lzw.(*encoder).Write
     0.81s 13.39% 63.64%      0.93s 15.37%  image.(*Paletted).SetColorIndex
     0.39s  6.45% 70.08%      1.32s 21.82%  main.paintSquare
     0.25s  4.13% 74.21%      0.25s  4.13%  runtime.nanotime1
     0.22s  3.64% 77.85%      0.22s  3.64%  runtime.kevent
     0.19s  3.14% 80.99%      2.01s 33.22%  compress/lzw.(*encoder).writeLSB
     0.19s  3.14% 84.13%      0.41s  6.78%  runtime.netpoll
     0.14s  2.31% 86.45%      1.82s 30.08%  image/gif.blockWriter.WriteByte
     0.14s  2.31% 88.76%      0.14s  2.31%  runtime.pthread_cond_wait
     0.13s  2.15% 90.91%      4.95s 81.82%  main.animate
```

Painted squares

We represent each “item” in the array as a square



Painting each square

We're using a standard library function `SetColorIndex`

```go
func paintSquare(i, k int, src []int, img *image.Paletted) {
    // lay down a square with an outline using the default
    // color (gray; we deliberately excluded it from the data)

    for x := 0; x < scale; x++ {
        for y := 0; y < scale; y++ {
            idx := uint8(src[i])

            if x == 0 || y == 0 || x == scale-1 || y == scale-1 {
                idx = 0
            }
            img.SetColorIndex(i*scale+x, k*scale+y, idx)
        }
    }
}
```

Setting the color index

In `image/image.go`, we see it does extra work

- checking the point's location
- recalculating the pixel's offset

```go
func (p *Paletted) PixOffset(x, y int) int {
    return (y-p.Rect.Min.Y)*p.Stride + (x-p.Rect.Min.X)*1
}

func (p *Paletted) SetColorIndex(x, y int, index uint8) {
    if !(Point{x, y}.In(p.Rect)) {
        return
    }
    i := p.PixOffset(x, y)
    p.Pix[i] = index
}
```

Optimize the function

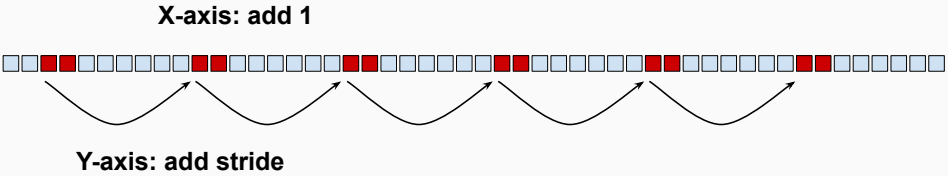
There are four things we can do:

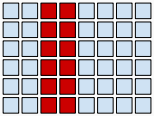
- eliminate the unnecessary check
- move some multiplication out of the loop
- *strength reduction:* replace multiplication with addition *in the loop*
- provided that we reorder the loops (`y` then `x`)

We take advantage of the layout of pixels in the image
as well as the knowledge that we're filling in a square

Strength reduction

Pixels are kept in a slice, one row after another

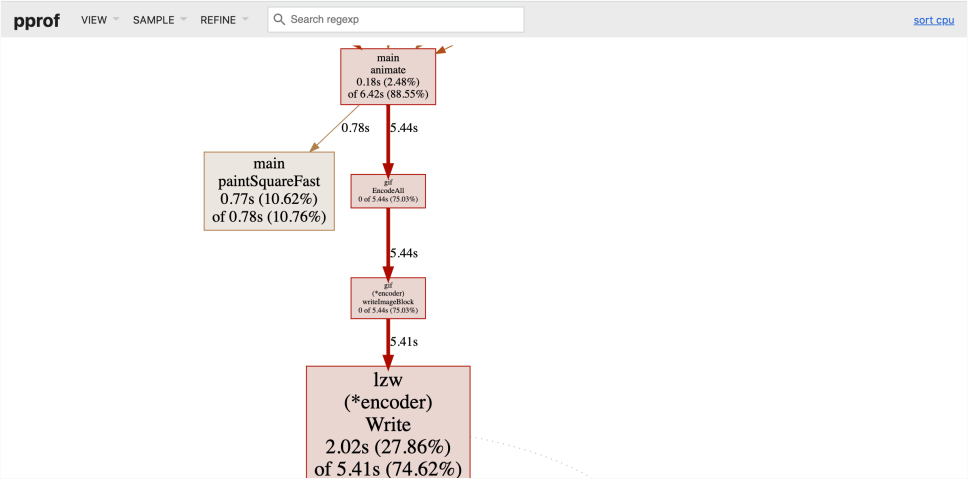




Painting each square #2

```go
func paintSquareFast(i, k int, src []int, img *image.Paletted) {
    ci := uint8(src[i])
    is, ks := i * scale, k * scale
    px := (ks-img.Rect.Min.Y)*img.Stride + (is-img.Rect.Min.X)*1

    for y := 0; y < scale; y++ {
        for x := 0; x < scale; x++ {
            idx := ci

            if x == 0 || y == 0 || x == scale-1 || y == scale-1 {
                idx = 0
            }
            img.Pix[px+x] = idx
        }
        px += img.Stride
    }
}
```

CPU profile #2



CPU profile #2

```
$ go tool pprof -top sort profile-fast
File: sort
Type: cpu
Time: Jan 31, 2021 at 8:31am (MST)
Duration: 30s, Total samples = 7.25s (24.17%)
Showing nodes accounting for 7.11s, 98.07% of 7.25s total
Dropped 40 nodes (cum <= 0.04s)
      flat  flat%   sum%        cum   cum%
     2.93s 40.41% 40.41%      2.93s 40.41%  syscall.syscall
     2.02s 27.86% 68.28%      5.41s 74.62%  compress/lzw.(*encoder).Write
     0.77s 10.62% 78.90%      0.78s 10.76%  main.paintSquareFast
     0.35s  4.83% 83.72%      0.35s  4.83%  runtime.nanotime1
     0.30s  4.14% 87.86%      3.37s 46.48%  compress/lzw.(*encoder).writeLSB
     0.18s  2.48% 90.34%      6.42s 88.55%  main.animate
```

CPU profile #2

pprof VIEW SAMPLE REFINE Search regexp sort cpu

main.paintSquareFast

/Users/mholiday/sort/sort.go

```
Total:      1.55s      1.58s (flat, cum) 11.69%
   64          .          .                                img.SetColorIndex(i*scale+x, k*scale+y, idx)
   65          .          .                            }
   66          .          .                    }
   67          .          .            }
   68          .          .
   69       10ms       10ms            func paintSquareFast(i, k int, src []int, img *image.Paletted) {
   70          .          .                    // lay down a square with an outline using the default
   71          .          .                    // color (gray; we deliberately excluded it from the data)
   72          .          .
   73          .          .                    ci := uint8(src[i])
   74          .          .                    is, ks := i*scale, k*scale
   75       10ms       20ms                    px := (ks-img.Rect.Min.Y)*img.Stride + (is-img.Rect.Min.X)*1
   76          .          .
   77          .       10ms                    for y := 0; y < scale; y++ {
   78      660ms      660ms                            for x := 0; x < scale; x++ {
   79          .          .                                    idx := ci
   80          .          .
   81       10ms       20ms                                    if x == 0 || y == 0 || x == scale-1 || y == scale-1 {
   82          .          .                                            idx = 0
   83          .          .                                    }
   84          .          .
   85      830ms      830ms                                    img.Pix[px+x] = idx
   86          .          .                            }
   87          .          .
   88       30ms       30ms                            px += img.Stride
```

Optimize the function, part deux

There are two more things we can do:

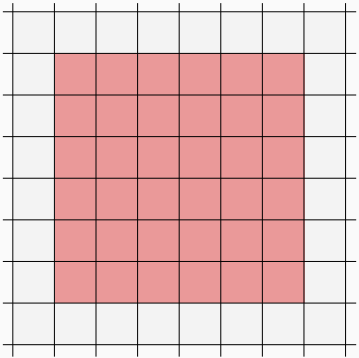
- eliminate the if-then logic in the loop
- reduce slice indexing (& bounds checks) from $O(n^2)$ to $O(n)$

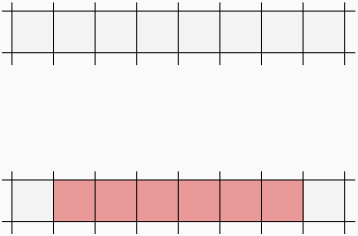
We can do this by

- splitting the square into top, middle, bottom sections
- copying a slice of pixels of the correct color into the image

Painted squares

We only have two kinds of rows to paint





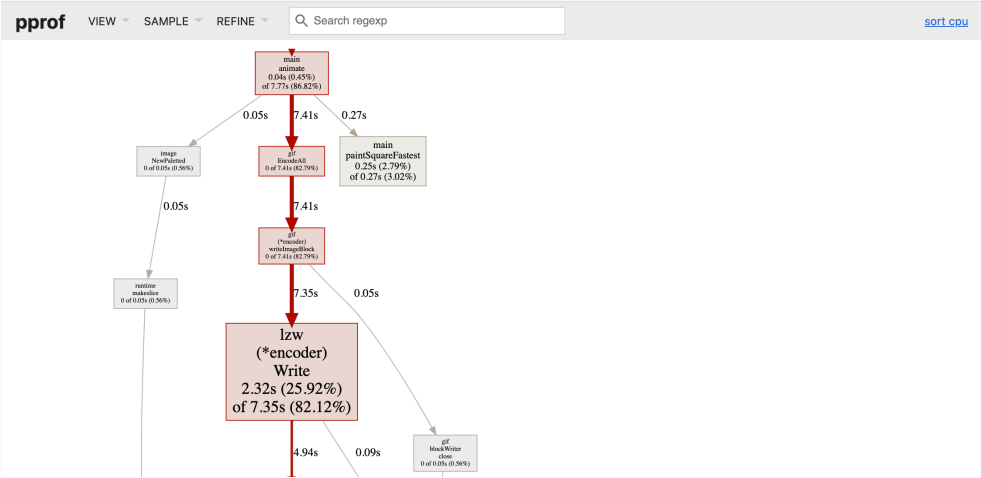
Painting each square #3

```go
func paintSquareFastest(i, k int, src []int, img *image.Paletted) {
    ci := uint8(src[i])
    is, ks := i*scale, k*scale
    px := (ks-img.Rect.Min.Y)*img.Stride + (is-img.Rect.Min.X)*1
    py := px + (scale-1)*img.Stride

    // create a row for the top and bottom

    rw := make([]uint8, scale)

    // paint the top & bottom rows of the square

    copy(img.Pix[px:px+scale], rw)
    copy(img.Pix[py:py+scale], rw)

    . . .
```

Painting each square #3

```
    . . .

    // fill in the middle part of the row

    for x := 1; x < scale-1; x++ {
        rw[x] = ci
    }

    // paint the middle part of the square

    for y := 1; y < scale-1; y++ {
        px += img.Stride
        copy(img.Pix[px:px+scale], rw)
    }
}
```

CPU profile #3



CPU profile #3

```
$ go tool pprof -top sort /Users/mholiday/Downloads/profile-fastest-3
File: sort
Type: cpu
Time: Jan 31, 2021 at 12:40pm (MST)
Duration: 30.11s, Total samples = 8.95s (29.72%)
Showing nodes accounting for 8.80s, 98.32% of 8.95s total
Dropped 44 nodes (cum <= 0.04s)
      flat  flat%   sum%        cum   cum%
     4.52s 50.50% 50.50%      4.53s 50.61%  syscall.syscall
     2.32s 25.92% 76.42%      7.35s 82.12%  compress/lzw.(*encoder).Write
     0.35s  3.91% 80.34%      0.35s  3.91%  runtime.kevent
     0.32s  3.58% 83.91%      0.32s  3.58%  runtime.nanotime1
     0.25s  2.79% 86.70%      0.27s  3.02%  main.paintSquareFastest
     0.24s  2.68% 89.39%      4.94s 55.20%  compress/lzw.(*encoder).writeLSB
     0.15s  1.68% 91.06%      4.70s 52.51%  image/gif.blockWriter.WriteByte

     . . .

     0.04s  0.45% 97.32%      7.77s 86.82%  main.animate
```

CPU profile #3

pprof VIEW SAMPLE REFINE Search regexp sort cpu

main.paintSquareFastest
/Users/mholiday/sort/sort.go

```
Total:        250ms      270ms (flat, cum)  3.02%
   87            .          .
   88            .          .                        px += img.Stride
   89            .          .                }
   90            .          .            }
   91            .          .
   92            .          .            func paintSquareFastest(i, k int, src []int, img *image.Paletted) {
   93            .          .                ci := uint8(src[i])
   94            .       10ms                is, ks := i*scale, k*scale
   95            .          .                px := (ks-img.Rect.Min.Y)*img.Stride + (is-img.Rect.Min.X)*1
   96            .          .                py := px + (scale-1)*img.Stride
   97            .          .                rw := make([]uint8, scale)
   98            .          .
   99         60ms       60ms                copy(img.Pix[px:px+scale], rw)
  100         30ms       30ms                copy(img.Pix[py:py+scale], rw)
  101            .          .
  102         20ms       20ms                for x := 1; x < scale-1; x++ {
  103            .          .                        rw[x] = ci
  104            .          .                }
  105            .          .
  106            .       10ms                for y := 1; y < scale-1; y++ {
  107            .          .                        px += img.Stride
  108        140ms      140ms                        copy(img.Pix[px:px+scale], rw)
  109            .          .                }
  110            .          .            }
  111            .          .
```