

Matching

Removing the go statement from `Seek` will turn the program into a synchronous one, where all steps can be predicted. Anna is going to send a message to Bob, Cody to Dave, and Eva to no one.

Changing wait group declarations will throw an error because we don't pass the wait group by reference to `Seek` method and so the wait group inside `Seek` isn't able to communicate with the one in `main`. Two messages will be sent/received and the last one will fail as all go routines are asleep.

Removing the buffer will thrown an error because unbuffered channels accept sends only if there is a corresponding receive. There doesn't seem to be a clear receiver of the sent information through the channel.

By default channels are unbuffered, meaning that they will only accept sends (`chan j-`) if there is a corresponding receive (`j- chan`) ready to receive the sent value. Buffered channels accept a limited number of values without a corresponding receiver for those values.¹

Removing default will not cause any problems in the current set up as it is constructed in such a way that one will be left unmatched. However, decreasing the size of the array of people, for example by removing *Eva* will throw a an error (a deadlock) because there is no one in the match channel left and it is still open. Thus, we are waiting for something that never arrives, which leads to a deadlock. A solution would be to add `close(match)` right after `wg.Wait()`.

Julia

The program uses all CPUs. The speed improved in the order of 20s. It seems like some might argue that we should have go routines for each pixel, but I think that we should have four that can be spread out on 4 processors.

```
1 // Stefan Nilsson 2013-02-27
2
3 // This program creates pictures of Julia sets
  (en.wikipedia.org/wiki/Julia_set).
4 package main
5
6 import (
7     "fmt"
8     "image"
9     "image/color"
10    "image/png"
11    "log"
12    "math/cmplx"
13    "os"
14    "runtime"
15    "strconv"
16    "sync"
17    "time"
18 )
19
20 type ComplexFunc func(complex128) complex128
21
22 var Funcs [] ComplexFunc = [] ComplexFunc{
23     func(z complex128) complex128 { return z*z - 0.61803398875 },
24     func(z complex128) complex128 { return z*z + complex(0, 1) },
```

¹<https://gobyexample.com/channel-buffering>, accessed 2015.04.14

```

25     func(z complex128) complex128 { return z*z + complex(-0.835, -0.2321)
        },
26     func(z complex128) complex128 { return z*z + complex(0.45, 0.1428) },
27     func(z complex128) complex128 { return z*z*z + 0.400 },
28     func(z complex128) complex128 { return cmplx.Exp(z*z*z) - 0.621 },
29     func(z complex128) complex128 { return (z*z+z)/cmplx.Log(z) +
        complex(0.268, 0.060) },
30     func(z complex128) complex128 { return cmplx.Sqrt(cmplx.Sinh(z*z)) +
        complex(0.065, 0.122) },
31 }
32
33 func main() {
34     // use all the power!
35     numcpu := runtime.NumCPU()
36     runtime.GOMAXPROCS(numcpu)
37
38     start := time.Now()
39
40     ch := make(chan error, len(Funcs)) // new channel
41
42     wg := new(sync.WaitGroup)
43     wg.Add(len(Funcs)) // the number of functions = no. of pictures.
44     for n, fn := range Funcs {
45         go CreatePng("picture-"+strconv.Itoa(n)+".png", fn, 1024, wg, ch)
46     }
47
48     wg.Wait()
49     close(ch)
50
51     for k := range ch {
52         if k != nil {
53             log.Fatal(k)
54         }
55     }
56     fmt.Println(time.Since(start))
57 }
58
59 // CreatePng creates a PNG picture file with a Julia image of size n x n.
60 func CreatePng(filename string, f ComplexFunc, n int, wg *sync.WaitGroup,
    ch chan error) { // need pointer to wg.
61     file, err := os.Create(filename)
62     if err != nil {
63         ch <- err
64         return
65     }
66     defer file.Close()
67     ch <- png.Encode(file, Julia(f, n))
68
69     wg.Done()
70     return
71 }
72
73 // Julia returns an image of size n x n of the Julia set for f.
74 func Julia(f ComplexFunc, n int) image.Image {
75     bounds := image.Rect(-n/2, -n/2, n/2, n/2)
76     img := image.NewRGBA(bounds)

```

```

77     s := float64(n / 4)
78
79     // code duplication... not good.
80
81     for i := bounds.Min.X; i < bounds.Max.X; i++ {
82         wg := new(sync.WaitGroup)
83         wg.Add(4)
84         go func() {
85             for j := bounds.Min.Y; j < bounds.Max.Y/4; j++ {
86
87                 n := Iterate(f, complex(float64(i)/s, float64(j)/s), 256)
88                 r := uint8(0)
89                 g := uint8(0)
90                 b := uint8(n % 32 * 8)
91                 img.Set(i, j, color.RGBA{r, g, b, 255})
92
93             }
94
95             wg.Done()
96         }()
97
98         go func() {
99             for j := bounds.Max.Y / 4; j < 2*(bounds.Max.Y/4); j++ {
100
101                 n := Iterate(f, complex(float64(i)/s, float64(j)/s), 256)
102                 r := uint8(0)
103                 g := uint8(0)
104                 b := uint8(n % 32 * 8)
105                 img.Set(i, j, color.RGBA{r, g, b, 255})
106
107             }
108             wg.Done()
109         }()
110
111         go func() {
112             for j := 2 * (bounds.Max.Y / 4); j < 3*(bounds.Max.Y/4); j++ {
113
114                 n := Iterate(f, complex(float64(i)/s, float64(j)/s), 256)
115                 r := uint8(0)
116                 g := uint8(0)
117                 b := uint8(n % 32 * 8)
118                 img.Set(i, j, color.RGBA{r, g, b, 255})
119
120             }
121             wg.Done()
122         }()
123
124         go func() {
125             for j := 3*(bounds.Max.Y/4) + (bounds.Max.Y % 4); j <
126                 bounds.Max.Y; j++ {
127
128                 n := Iterate(f, complex(float64(i)/s, float64(j)/s), 256)
129                 r := uint8(0)
130                 g := uint8(0)
131                 b := uint8(n % 32 * 8)
132                 img.Set(i, j, color.RGBA{r, g, b, 255})

```

```

132
133         }
134         wg.Done()
135     }()
136
137     wg.Wait()
138 }
139
140     return img
141 }
142
143 // Iterate sets  $z_0 = z$ , and repeatedly computes  $z_n = f(z_{n-1})$ ,  $n \geq 1$ ,
144 // until  $|z_n| > 2$  or  $n = \text{max}$  and returns this  $n$ .
145 func Iterate(f ComplexFunc, z complex128, max int) (n int) {
146     for ; n < max; n++ {
147         if real(z)*real(z)+imag(z)*imag(z) > 4 {
148             break
149         }
150         z = f(z)
151     }
152     return
153 }

```

Weather Station

```

1 // Stefan Nilsson 2013-02-27
2
3 // This program creates pictures of Julia sets
4 // (en.wikipedia.org/wiki/Julia_set).
5
6 package main
7
8 import (
9     "fmt"
10    "image"
11    "image/color"
12    "image/png"
13    "log"
14    "math/cmplx"
15    "os"
16    "runtime"
17    "strconv"
18    "sync"
19    "time"
20 )
21
22 type ComplexFunc func(complex128) complex128
23
24 var Funcs []ComplexFunc = []ComplexFunc{
25     func(z complex128) complex128 { return z*z - 0.61803398875 },
26     func(z complex128) complex128 { return z*z + complex(0, 1) },
27     func(z complex128) complex128 { return z*z + complex(-0.835, -0.2321) },
28     func(z complex128) complex128 { return z*z + complex(0.45, 0.1428) },
29     func(z complex128) complex128 { return z*z*z + 0.400 },
30 }

```

```

28     func(z complex128) complex128 { return cmplx.Exp(z*z*z) - 0.621 },
29     func(z complex128) complex128 { return (z*z+z)/cmplx.Log(z) +
        complex(0.268, 0.060) },
30     func(z complex128) complex128 { return cmplx.Sqrt(cmplx.Sinh(z*z)) +
        complex(0.065, 0.122) },
31 }
32
33 func main() {
34     // use all the power!
35     numcpu := runtime.NumCPU()
36     runtime.GOMAXPROCS(numcpu)
37
38     start := time.Now()
39
40     ch := make(chan error, len(Funcs)) // new channel
41
42     wg := new(sync.WaitGroup)
43     wg.Add(len(Funcs)) // the number of functions = no. of pictures.
44     for n, fn := range Funcs {
45         go CreatePng("picture-"+strconv.Itoa(n)+".png", fn, 1024, wg, ch)
46     }
47
48     wg.Wait()
49     close(ch)
50
51     for k := range ch {
52         if k != nil {
53             log.Fatal(k)
54         }
55     }
56     fmt.Println(time.Since(start))
57 }
58
59 // CreatePng creates a PNG picture file with a Julia image of size n x n.
60 func CreatePng(filename string, f ComplexFunc, n int, wg *sync.WaitGroup,
    ch chan error) { // need pointer to wg.
61     file, err := os.Create(filename)
62     if err != nil {
63         ch <- err
64         return
65     }
66     defer file.Close()
67     ch <- png.Encode(file, Julia(f, n))
68
69     wg.Done()
70     return
71 }
72
73 // Julia returns an image of size n x n of the Julia set for f.
74 func Julia(f ComplexFunc, n int) image.Image {
75     bounds := image.Rect(-n/2, -n/2, n/2, n/2)
76     img := image.NewRGBA(bounds)
77     s := float64(n / 4)
78
79     // code duplication... not good.
80

```

```

81  for i := bounds.Min.X; i < bounds.Max.X; i++ {
82      wg := new(sync.WaitGroup)
83      wg.Add(4)
84      go func() {
85          for j := bounds.Min.Y; j < bounds.Max.Y/4; j++ {
86
87              n := Iterate(f, complex(float64(i)/s, float64(j)/s), 256)
88              r := uint8(0)
89              g := uint8(0)
90              b := uint8(n % 32 * 8)
91              img.Set(i, j, color.RGBA{r, g, b, 255})
92
93          }
94
95          wg.Done()
96      }()
97
98      go func() {
99          for j := bounds.Max.Y / 4; j < 2*(bounds.Max.Y/4); j++ {
100
101              n := Iterate(f, complex(float64(i)/s, float64(j)/s), 256)
102              r := uint8(0)
103              g := uint8(0)
104              b := uint8(n % 32 * 8)
105              img.Set(i, j, color.RGBA{r, g, b, 255})
106
107          }
108          wg.Done()
109      }()
110
111      go func() {
112          for j := 2 * (bounds.Max.Y / 4); j < 3*(bounds.Max.Y/4); j++ {
113
114              n := Iterate(f, complex(float64(i)/s, float64(j)/s), 256)
115              r := uint8(0)
116              g := uint8(0)
117              b := uint8(n % 32 * 8)
118              img.Set(i, j, color.RGBA{r, g, b, 255})
119
120          }
121          wg.Done()
122      }()
123
124      go func() {
125          for j := 3*(bounds.Max.Y/4) + (bounds.Max.Y % 4); j <
            bounds.Max.Y; j++ {
126
127              n := Iterate(f, complex(float64(i)/s, float64(j)/s), 256)
128              r := uint8(0)
129              g := uint8(0)
130              b := uint8(n % 32 * 8)
131              img.Set(i, j, color.RGBA{r, g, b, 255})
132
133          }
134          wg.Done()
135      }()

```

```

136
137     wg.Wait()
138 }
139
140     return img
141 }
142
143 // Iterate sets  $z_0 = z$ , and repeatedly computes  $z_n = f(z_{n-1})$ ,  $n \geq 1$ ,
144 // until  $|z_n| > 2$  or  $n = \text{max}$  and returns this  $n$ .
145 func Iterate(f ComplexFunc, z complex128, max int) (n int) {
146     for ; n < max; n++ {
147         if real(z)*real(z)+imag(z)*imag(z) > 4 {
148             break
149         }
150         z = f(z)
151     }
152     return
153 }

```