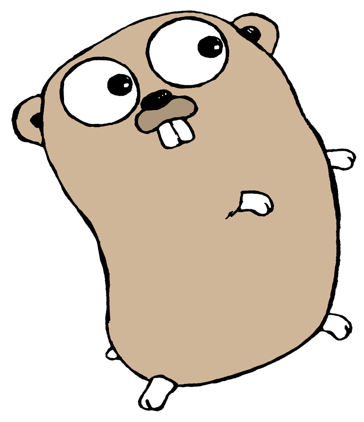

New Language, Go, Promising for Scientific Programming

Sonia Keys

Smithsonian Astrophysical Observatory
skeys@cfa.harvard.edu

Potential use of Go, a free and open source computer language sponsored by Google, is explored by porting a small program and comparing to other languages. Ported was Digest2, a program in use at the Minor Planet Center to screen submitted but unidentified asteroid astrometry tracklets for potential detections of Near Earth Objects. Go seems a promising new choice for scientific programming. It was found to have execution speed comparable to Fortran and C, source code simplicity comparable to Python, and a concurrency model that is unique and intuitive.

Go[1] Synopsis

Simplicity

- Clean library mechanism with no header files.
- Type inferencing.
- Complexity of modern object oriented languages consciously avoided.

Safety

- Strong static typing.
- Garbage collection.
- No pointer arithmetic.

Concurrency

- Based on CSP[2], but innovative in a new direction.
- "Goroutines" have runtime multiplexed threading.
- "Channels" are analogous to Unix pipes, but in memory and more flexible. Synchronous and asynchronous communication, buffered and unbuffered, receipt checked or unchecked, multiple reader, multiple writer. Channels are typed first class objects.

Features Enabling Scientific Programming

Easy

- Imperative, procedural programming style is the style familiar to most scientists. It is the style of Fortran and C.
- A clean, new language design without historical baggage.

Fast

- Native code compilers generate fast object code. Performance is comparable to other compiled languages like Fortran and C.
- Modern concurrency features designed for today's multicore CPUs.
- IEEE floating point, native complex types.

Accessible

- Free and open source.
- Runs on Linux, Darwin, Windows operating systems. x86, x86-64, and ARM architectures.
- SWIG support. This allows Go to link to existing well tested and highly optimized C and C++ libraries like BLAS/LAPACK, GNU Scientific Library, FFTW.
- Two compiler suites, one a GCC front end and the other stand-alone.

Language Comparison

Go keeps

- Speed of compiled languages like Fortran, C, and C++.
- Light declaration syntax of dynamically typed languages like Python or Javascript.
- Memory safety typical of interpreted languages.

Go adds

- Innovative concurrency model.
- Innovative abstraction mechanism, interfaces.

Go rejects

- Header files of C and C++, with their long dependency chains and long compilation times.
- Type hierarchies of C++ and most OO languages.
- Interpreters and VMs as with Python, Scala.
- Functional style characteristic of Haskell, Erlang.

```
top - 14:32:30 up 28 days, 16:25,  2 users,  load average: 2.38, 0.85, 0.38
Tasks: 132 total,   2 running, 130 sleeping,   0 stopped,   0 zombie
Cpu0  : 99.7%us,  0.0%sy,  0.0%ni,  0.3%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu1  : 98.7%us,  0.3%sy,  0.0%ni,  1.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu2  : 99.3%us,  0.0%sy,  0.0%ni,  0.7%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu3  : 99.3%us,  0.0%sy,  0.0%ni,  0.7%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:   3980400k total,  1965584k used,  2014816k free,   359708k buffers
Swap:  2040212k total,      196k used,  2040016k free,  1097044k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 1697 skeys     21   0 89084  74m  31m R 397.5  1.9   1:01.32 digest2
```

Concurrency

A parallelizable problem

- Given a list of asteroid "tracklets", Digest2 computes a "NEO score" for each. The computations are independent and so trivially parallelizable.

Method

- Start "workers." Goroutines are not tied to an OS thread, but in this case it makes sense to have the same number of workers as CPU cores.
- A channel of channels will be used to accumulate results.
- Start "dispatcher." On a four core CPU, six goroutines are now running: main, four workers, and dispatcher.
- Dispatcher attaches a "return ticket" to each tracklet. Annotated tracklets go in one channel, return tickets in another.
- All workers get annotated tracklets from the same source.
- They work independently and may complete out of order. Results are sent by return ticket.
- Scores arrive by return ticket, in order, on a single channel.

```
for i := 0; i < nWorkers; i++ {
    go solve(tkch)

prch := make(chan chan string, nWorkers*2)

go dispatch(tracklets, tkch, prch)

for rch := range prch {
    fmt.Println(<-rch)

for _, tk := range tracklets {
    tk.rch = make(chan string, 1)
    tkch <- tk
    prch <- tk.rch

for tk := range tkch {

tk.rch <- r
```

[1] http://golang.org
[2] Hoare, C. A. R. (1985). Communicating Sequential Processes. Prentice Hall. ISBN 0-13-153289-8.

Smithsonian Astrophysical Observatory

Harvard Smithsonian Center for Astrophysics