Go tour - introduction

Sébastien Binet

2012-05-31

Motivations

- Moore's law ceased to provide the traditional single-threaded performance increases
  - clock-frequency wall of 2003
  - still deliver increases in transistor density
- multicore systems become the norm
- need to "go parallel" to get scalability

In a C++ world...

- parallel programming in C++ is doable:
  - C/C++ "locking + threads" (`pthreads`, `WinThreads`)
    - ★ excellent performance
    - ★ good generality
    - ★ relatively low productivity
  - multi-threaded applications...
    - ★ hard to get right
    - ★ hard to keep right
    - ★ hard to keep efficient and optimized across releases
  - multi-process applications...
    - ★ leverage `fork+COW` on `GNU/Linux`

Parallel programming in C++ is doable,
but *no panacea*

In a C++ world...

- in C++03, we have libraries to help with parallel programming
  - `boost::lambda`
  - `boost::MPL`
  - `boost::thread`
  - Threading Building Blocks (TBB)
  - Concurrent Collections (CnC)
  - `OpenMP`
  - `...`

In a C++11 world...

- in C++11, we get:
  - λ functions (and a new syntax to define them)
  - `std::thread`,
  - `std::future`,
  - `std::promise`

Helps taming the beast
... at the price of sprinkling templates everywhere...
... and complicating further a not so simple language...

In a C++11 world...

yay! for C++11, but old problems are still there...

- build scalability
  - templates
  - headers system
  - still no module system (WG21 - N2073)
    - ★ maybe in the next Technical Report ?
- code distribution
  - no CPAN like readily available infrastructure (and cross-platform) for C++

Time for a new language ?

"Successful new languages build on existing languages and where possible, support legacy software. C++ grew our of C. java grew out of C++. To the programmer, they are all one continuous family of C languages." (T. Mattson)

- notable exception (which confirms the rule): python

Can we have a language:

- as easy as python,
- as fast (or nearly as fast) as `C/C++/FORTRAN`,
- with none of the deficiencies of `C++`,
- and is multicore/manycore friendly ?

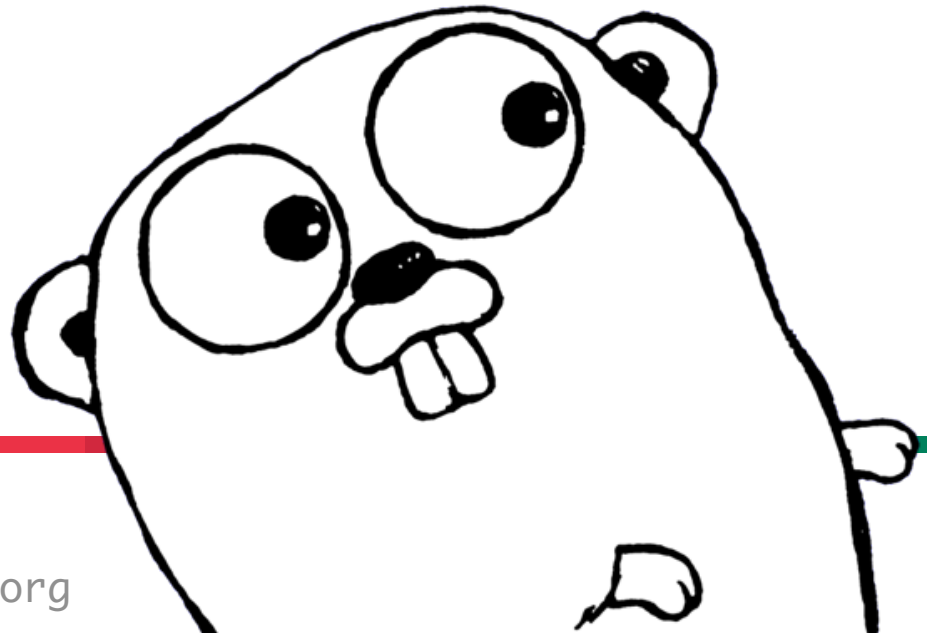
Why not Go ?

golang.org

Elements of go

- obligatory `hello world` example...

```
package main
import "fmt"
func main() {
    fmt.Println("Hello, World")
}
```



http://golang.org

Google

Elements of go - II

- founding fathers:
  - Russ Cox, Robert Griesemer, Ian Lance Taylor
  - Rob Pike, Ken Thompson
- concurrent, compiled
- garbage collected
- an open-source general programming language
- best of both 'worlds':
  - feel of a dynamic language
    - ⋆ limited verbosity thanks to **type inference system**, map, slices
  - safety of a static type system
  - compiled down to machine language (so it is fast)
    - ⋆ goal is within 10% of C
- object-oriented (but w/o classes), builtin reflection
- first-class functions with closures
- duck-typing à la `python`

Go concurrent

goroutines

- a function executing concurrently as other goroutines in the same address space
- starting a goroutine is done with the go keyword
  - `go myfct(arg1, arg2)`
- growable stack
  - lightweight threads
  - starts with a few kB, grows (and shrinks) as needed
    - ⋆ now, also available in GCC 4.6 (thanks to the GCC-Go front-end)
  - no stack overflow

Go concurrent - II

channels

- provide (type safe) communication and synchronization

```
// create a channel of mytype
my_chan := make(chan mytype)
my_chan <- some_data     // sending data
some_data = <- my_chan   // receiving data
```

- send and receive are atomic

"Do not communicate by sharing memory; instead, share memory by communicating"

Non-elements of Go

- no dynamic libraries (frown upon)
- no dynamic loading (yet)
  - but can either rely on separate processes
    - IPC is made easy *via* the `netchan` package
    - many RPC substrates too (`JSON`, `XML`, `protobuf`, ...)
  - or rebuild executables on the fly
    - compilation of `Go` code is fast
    - even faster than `FORTRAN` and/or `C`
- no templates/generics
  - still open issue
  - looking for the proper `Go` -friendly design
- no operator overloading

Go from anywhere to everywhere

- code compilation and distribution are (*de facto*) standardized
- put your code on some repository
  - `bitbucket`, `launchpad`, `googlecode`, `github`, ...
- check out, compile and install in one go with go get:
  - `go get bitbucket.org/binet/igo`
  - no `root` access required
  - automatically handle dependencies
- `go get` -able packages are listed on the dashboard:
  - godashboard.appspot.com

Tour content

- bases of go: types, slices, maps, functions, closures, interfaces
- goroutines, channels
- mini load-balancer


GO
http://golang.org

Bibliography

- http://golang.org
- http://tour.golang.org
- http://concur.rspace.googlecode.com/hg/talk/concur.html