

Can 'Go' address the multicore issues of today and the manycore problems of tomorrow ?

Sébastien Binet



2011-09-06

- 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...
 - ★ à la AthenaMP/GaudiMP (see Vahko's talk on Thu.)
 - ★ leverage fork+COW on GNU/Linux
 - ★ event-level based parallelism

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

- 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++
- ▶ remember ROOT/BOOT ? (CHEP-06)

Time for a new language ?

“Successful new languages build on existing languages and where possible, support legacy software. C++ grew out 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:

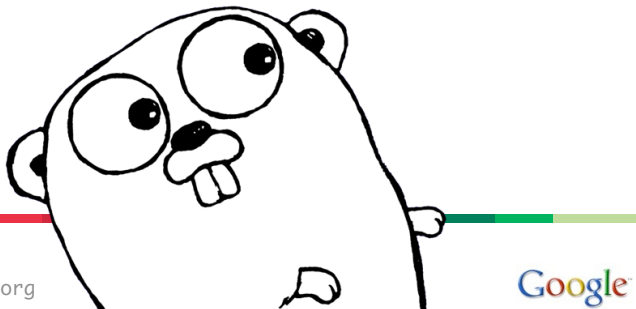
- as easy as **python**,
- as fast (or nearly as fast) as C/C++/FORTRAN,
- with none of the deficiencies of C++,
- and is multicore/manycore friendly ?

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

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

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"*

Go concurrent - III

```
package evtproc
import "gaudi/kernel"

// --- evt state ---
type evtstate struct {
    idx int
    sc kernel.Error
    data kernel.DataStore
}

// --- evt processor ---
type evtproc struct {
    kernel.Service
    algs []kernel.IAlgorithm
    nworkers int
}

func (self *evtproc)
NextEvent(evtmax int) kernel.Error {

    if self.nworkers > 1 {
        return self.mp_NextEvent(evtmax)
    }
    return self.seq_NextEvent(evtmax)
}
```

Go concurrent - IV

```
import "gaudi/kernel"

func (self *evtproc) mp_NextEvent(evtmax int) kernel.Error {
    // ... setup event server ...
    in_queue, out_queue, quit := start_evt_server(self.nworkers)
    for i := 0; i < evtmax; i++ {
        in_queue <- new_evtstate(i)
    }

    for evt := range out_queue {
        if !evt.sc.IsSuccess() {
            n_fails++
        }
        n_processed++
        if n_processed == evtmax {
            quit <- true
            close(out_queue)
            break
        }
    }
    if n_fails != 0 {
        return kernel.StatusCode(1)
    }
}
```

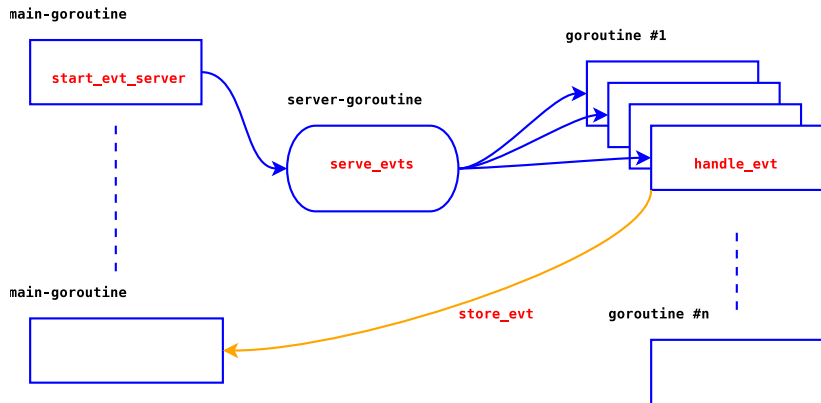
Go concurrent - V

```
func (self *evtproc) mp_NextEvent(evtmax int) kernel.Error {  
  
    start_evt_server := func(nworkers int) (in_evt_queue,  
                                     out_evt_queue chan *evtstate,  
                                     quit chan bool) {  
        in_evt_queue = make(chan *evtstate, nworkers)  
        out_evt_queue = make(chan *evtstate)  
        quit = make(chan bool)  
        go serve_evts(in_evt_queue, out_evt_queue, quit)  
        return in_evt_queue, out_evt_queue, quit  
    }  
  
    // ...  
    return kernel.StatusCode(0)  
}
```

Go concurrent - VI

```
func (self *evtproc) mp_NextEvent(evtmax int) kernel.Error {  
  
    handle := func(evt *evtstate, out_queue chan *evtstate) {  
        self.MsgInfo("nextEvent [%v]...\n", evt.idx)  
        evt.sc = self.ExecuteEvent(evt)  
        out_queue <- evt  
    }  
  
    serve_evts := func(in_evt_queue, out_evt_queue chan *evtstate,  
        quit chan bool) {  
        for {  
            select {  
            case ievt := <-in_evt_queue:  
                go handle(ievt, out_evt_queue)  
            case <-quit:  
                return  
            }  
        }  
    }  
  
    // ...  
    return kernel.StatusCode(0)  
}
```


Go concurrent - VII



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
 - ▶ 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 **goinstall**:
 - ▶ `goinstall bitbucket.org/binet/igo`
 - ▶ no root access required
 - ▶ automatically handle **dependencies**
- `goinstall`-able packages are listed on the dashboard:
 - ▶ godashboard.appspot.com

Go and C/C++

Interfacing with C:

- done with the CGo foreign function interface
- `#include` the header file to the C library to be wrapped
- access the C types and functions under the artificial “C” package

```
package myclib
```

```
// #include <stdio.h>
```

```
// #include <stdlib.h>
```

```
import "C"
```

```
import "unsafe"
```

```
func foo(s string) {
```

```
    c_str := C.CString(s) // create a C string from a Go o
```

```
    C.fputs(c_str, C.stdout)
```

```
    C.free(unsafe.Pointer(c_str))
```

```
}
```

Interfacing with C++:

- a bit more involved
- uses SWIG
 - ▶ you write the SWIG interface file for the library to be wrapped
 - ▶ SWIG will generate the C stub functions
 - ▶ which can then be called using the CGo machinery
 - ▶ the Go files doing so are automatically generated as well
- handles overloading, multiple inheritance
- allows to provide a Go implementation for a C++ abstract class

Problem

SWIG doesn't understand all of C++03

- e.g. can't parse `TObject.h`

Go and FORTRAN

Two cases:

- lucky enough to wrap “legacy” Fortran 03 code with the ISO C interface:
 - ▶ just use CGo
- wrapping legacy F77 code:
 - ▶ write the C interface code
 - ▶ use CGo to call this interface code
- examples:
 - ▶ <http://bitbucket.org/binet/go-hepevt>
 - ▶ <http://bitbucket.org/binet/go-herwig>
- no automatic press-button solution
 - ▶ although there is no technical blocker to write such a thing
 - ▶ this has been done for python (e.g.: fwrap)

Go and ROOT

- step 1 of evil plan for (HENP) world domination:
 - ▶ Go bindings to ROOT
- <http://bitbucket.org/binet/go-croot>
 - ▶ hand written CGo bindings to a hand written library exposing a C interface to (a subset of) ROOT
 - ★ TFile, TTree/TChain
 - ★ Reflex, Cint
 - ★ TRandom
 - ▶ handles automatic conversion of Go structs into their C counter-part
 - ▶ and *vice versa*
 - ★ two-way conversion is done by connecting the C++ introspection library (Reflex) with its Go counter-part (the reflect package)

Go and ROOT

- running the ROOT TTree example, in C++, via the C API and through go-croot over 10000000 events:

```
29.04s user 1.03s system 86% cpu 34.83 total (C++)
29.12s user 1.09s system 85% cpu 35.48 total (CRoot)
44.83s user 1.24s system 87% cpu 54.36 total (go-croot)
```

```
$ uname -a
```

```
Linux farnsworth 3.0-ARCH #1 SMP PREEMPT
x86_64 Intel(R) Core(TM)2 Duo
CPU T9400 @ 2.53GHz GenuineIntel GNU/Linux
```

additional overhead *w.r.t.* CRoot

- different calling conventions (b/w C and Go) need to be handled
- *Note:* for such loopy-code, using GCC-Go would be better

Conclusions

Can Go address the (non-) multicore problems of yesterday ?

- **yes:**
 - ▶ productivity (dev cycle of a scripting language)
 - ▶ build scalability (package system)
 - ▶ deployment (goinstall)
 - ▶ support for “legacy” C/C++/Fortran software (cgo+swig)

Can Go address the multicore issues of tomorrow ?

- **yes:**
 - ▶ **easier** to write concurrent code with the builtin abstractions (goroutines, channels)
 - ▶ **easier** to have efficient concurrent code (stack management)
 - ▶ still have to actually write efficient concurrent code, though...
 - ★ work partitioning, load balancing, ...
- **but:** no such thing as a magic wand for multicores/manycores

Prospects - what's missing ?

- better support for C++ libraries
 - ▶ building on ROOT C++ dictionary infrastructure
 - ★ now using GCC-Xml + a modified version of genreflex
 - ★ tomorrow using LLVM/CLang
 - ▶ automatically generate the Go bindings
- bind more HEP libraries ?
- provide a Go interpreter ?
 - ▶ bitbucket.org/binet/igo

Resources

- golang.org
- root.cern.ch
- swig.org
- godashboard.appspot.com
- bitbucket.org/binet/go-hepevt
- bitbucket.org/binet/go-herwig
- bitbucket.org/binet/go-croot
- [fwrap](#)
- [LLVM](#)
- [CLang](#)