New Parallel Programming Languages for Optimization Research

John W. Chinneck, Stephane Ernst

Systems and Computer Engineering

Carleton University, Ottawa, Canada

Motivation

- Challenges for optimization algorithms:
  - *Always*: faster solutions for bigger problems
  - *New*: massive scale up to handle big data
- Hardware has evolved:
  - *Multiple processors are everywhere*
  - Even *phones* have quad core processors!
  - Recent purchase: 16-core machine for $2000
- Conclusion:
  - *New optimization algorithms **must** be parallel*
  - Must handle big data problems
  - Must take advantage of parallel hardware

Language Selection Criteria

- Shortest distance between *idea* and *implementation*
  - *I'm an algorithms guy, not a programming specialist*
  - Easy to learn and program
  - Parallelism (concurrency) built-in and easy to use
- Fast execution
  - Needed for comparisons to commercial solvers
  - *Compiled* language execution speed
- Nice to have:
  - Multi-platform (Windows, linux, Apple)
  - Fast compilation
  - Integrated Development Environment (IDE)
  - Low cost / free
  - Active user community (especially optimizers)

Go Language: Design Criteria

- Language specification simple enough to hold in a programmer's head.
- Built-in concurrency
- Others
  - Automatic garbage collection
  - Fast compilation and execution
  - Simple system for dependencies
    - *I hate header files*

Helpful Features of Go

- Simplicity
  - No header files!
  - Simple scoping. E.g. externally visible package-level variable: just capitalize the first letter
  - No type inheritance
  - No method or operator overloading
  - No circular dependencies among packages
  - No pointer arithmetic
- Very strict compiler prevents common errors
  - No mixed-type arithmetic: you must explicitly cast types.
- Enforced efficiency
  - Unused variables are an *error*
- Enforced common format
  - Just run *gofmt*: takes care of indenting etc. in a standard way
- Call C code directly
  - Use *cgo* or *gccgo*
- Debugger

Packages

```go
package solver
// Controls the solution process

import (
	"fmt"
	"lp"
	"math"
	"math/rand"
	"sort"
	"strconv"
	"time"
)
```

These are the names of Go Packages, some built-in, some I created. Each can expose variables and routines.

```go
// Package global variables
var PrintLevel int     // controls the level of printing. Setting it equal to zero turns printing off
var FinalBox int       // Captures the last box commenced so it can be printed out

// Structures needed for sorting the impact list
type IMPACT struct {
	Row   int
	Sum   int
}

func Solve(AlphaIn float64, BetaIn float64, MaxItnsIn int, MaxSwarmPtsIn int, plinfyIn float64, ...
…
```

External Reference to a Package Variable:

```go
solver.PrintLevel = 0     // PrintLevel = 0 turns off the printing so you can run through a set of files
```

External Reference to a package routine:

```go
Point, Status = solver.Solve(Alpha, Beta, MaxItns, MaxSwarmPts, plinfy, featol)
```

Language Elements

- Statements are minimal and simple:
  - Only one kind of loop: *for*
    - Index over a range, or over the length of a vector
    - Can act like a while loop
  - If-then-else
  - Select / Case
  - Etc.
- General data structures
- Arrays and "slices" (vectors)
- *Generally simple and intuitive*

Functions

```
//===========================================================================================
// Given an input point at which some of the variables may violate their bounds, this
// routine returns an output point in which all of the variables have been reset onto their
// closest bound, if necessary.

func EnforceBounds(PtIn []float64) (PtOut []float64) {
	PtOut = make([]float64, len(PtIn))
	for j:=0; j<lp.NumCols; j++ {
		if PtIn[j] < lp.LP.Cols[j].BndLo {
			PtOut[j] = lp.LP.Cols[j].BndLo
			continue
		}
		if PtIn[j] > lp.LP.Cols[j].BndUp {
			PtOut[j] = lp.LP.Cols[j].BndUp
			continue
		}
		PtOut[j] = PtIn[j]
	}
	return
}
```

Concurrency

- Make any routine concurrent by the *go* keyword
  - Spawns a new asynchronous thread
- Communication via *channels*
  - Channels have defined types
    - Could be a structure holding many items
  - Return results via channels
- Channels allow:
  - Blocking to wait for something to be received
  - Receive something from one of several channels
  - Etc.
- There is also a *sync* package
  - Mutex, lock, wait, etc.

Concurrency example

```go
NumCPUs := runtime.NumCPU()
...
MaxPts := 2 * NumCPUs
...
chPoint := make(chan []float64)
...

for itn := 0; itn < MaxItns; itn++ {

      // Get new set of CC start points
      NewPoints(itn)

      // Run CC in parallel to improve each start point
      for i := 0; i < MaxPts; i++ {
            go CC(Point[i], chPoint, i)
      }

      // Retrieve the CC output points
      for i := 0; i < MaxPts; i++ {
            Point[i] = <-chPoint
      }

} // end of large iteration loop
```

Adding the *go* keyword before calling a routine spawns a concurrent goroutine

Concurrency:
hard lessons for a newbie

- Return order:
  - Routines return results in a different order than they were instantiated
  - Interruptions from other processes, etc.
- Reads and writes to common memory:
  - Unpredictable order of reads/writes
  - Best to communicate solely via channels where possible

Go Packages

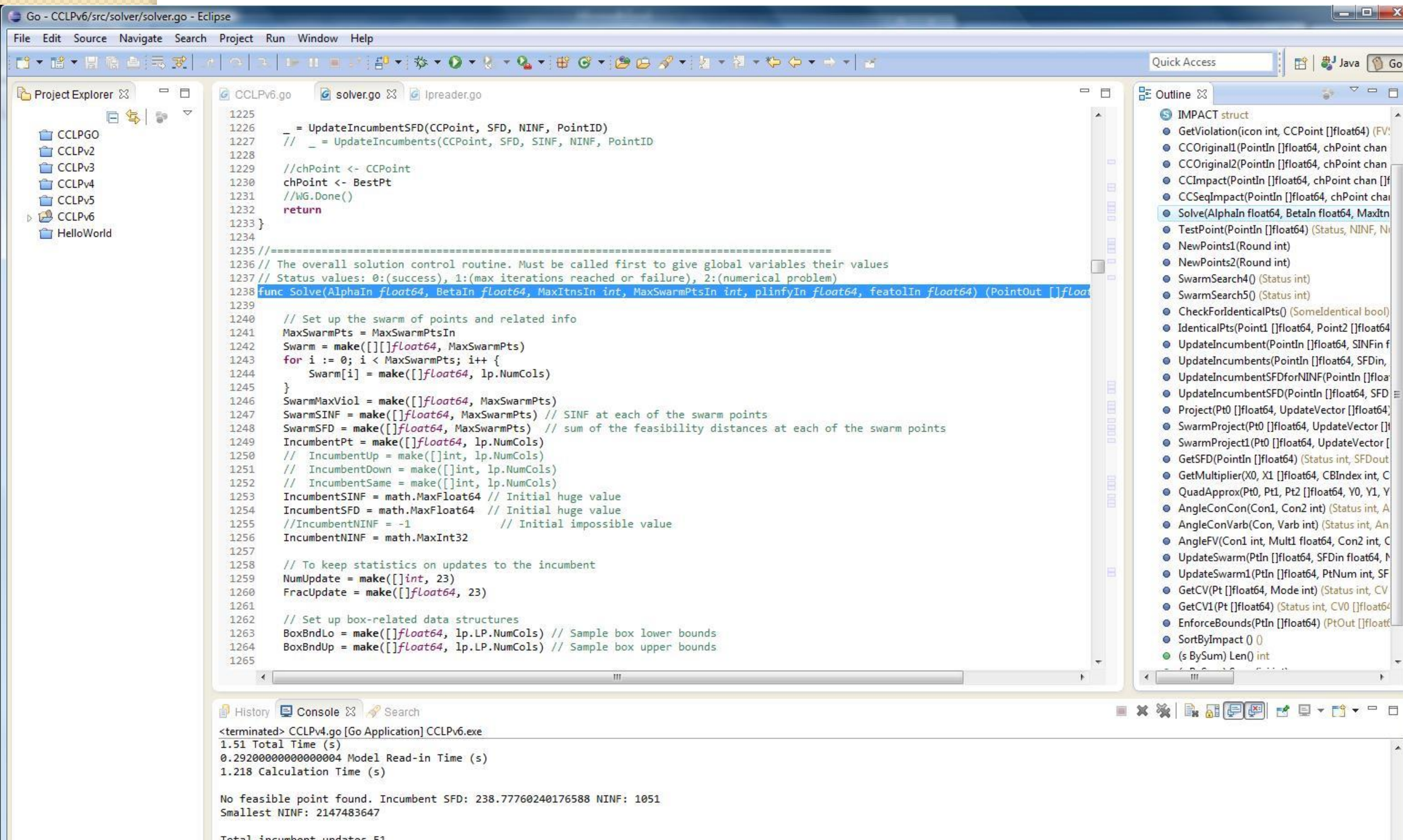
- Many built-in, see http://golang.org/pkg/
  - E.g. sorting, database, etc.
- External projects:
  - https://code.google.com/p/go-wiki/wiki/Projects
  - E.g. Mathematics, machine learning
  - CVX (ported from the CVX python package)
  - A particle swarm optimizer
  - Linear algebra routines, e.g. BLAS
  - Graph theory algorithms

Learning Go is easy

- Start at the tour of Go:
  http://tour.golang.org/#1
- Go documentation:
  http://golang.org/doc/
  includes video tours, docs, examples
- Online books:
  http://www.golang-book.com/
- The Go playground:
  http://play.golang.org/
- Go home:
  http://golang.org/
- Searching online for Go information:
  search on "golang"

IDEs for Go

- See http://geekmonkey.org/articles/20-comparison-of-ides-for-google-go
- I like Eclipse (called Goclipse): https://code.google.com/p/goclipse/


Go - CCLPv6/src/solver/solver.go - Eclipse
File Edit Source Navigate Search Project Run Window Help
Quick Access
Java
Go
Project Explorer
CCLPGO
CCLPv2
CCLPv3
CCLPv4
CCLPv5
CCLPv6
HelloWorld
CCLPv6.go
solver.go
lpreader.go
_ = UpdateIncumbentSFD(CCPoint, SFD, NINF, PointID)
// _ = UpdateIncumbents(CCPoint, SFD, SINF, NINF, PointID
//chPoint <- CCPoint
chPoint <- BestPt
//WG.Done()
return
}
//==
// The overall solution control routine. Must be called first to give global variables their values
// Status values: 0:(success), 1:(max iterations reached or failure), 2:(numerical problem)
func Solve(AlphaIn float64, BetaIn float64, MaxItnsIn int, MaxSwarmPtsIn int, plinfyIn float64, featolIn float64) (PointOut []float
// Set up the swarm of points and related info
MaxSwarmPts = MaxSwarmPtsIn
Swarm = make([][]float64, MaxSwarmPts)
for i := 0; i < MaxSwarmPts; i++ {
Swarm[i] = make([]float64, lp.NumCols)
}
SwarmMaxViol = make([]float64, MaxSwarmPts)
SwarmSINF = make([]float64, MaxSwarmPts) // SINF at each of the swarm points
SwarmSFD = make([]float64, MaxSwarmPts) // sum of the feasibility distances at each of the swarm points
IncumbentPt = make([]float64, lp.NumCols)
// IncumbentUp = make([]int, lp.NumCols)
// IncumbentDown = make([]int, lp.NumCols)
// IncumbentSame = make([]int, lp.NumCols)
IncumbentSINF = math.MaxFloat64 // Initial huge value
IncumbentSFD = math.MaxFloat64 // Initial huge value
//IncumbentNINF = -1 // Initial impossible value
IncumbentNINF = math.MaxInt32
// To keep statistics on updates to the incumbent
NumUpdate = make([]int, 23)
FracUpdate = make([]float64, 23)
// Set up box-related data structures
BoxBndLo = make([]float64, lp.LP.NumCols) // Sample box lower bounds
BoxBndUp = make([]float64, lp.LP.NumCols) // Sample box upper bounds
Outline
IMPACT struct
GetViolation(icon int, CCPoint []float64)
CCOriginal1(PointIn []float64, chPoint chan
CCOriginal2(PointIn []float64, chPoint chan
CCImpact(PointIn []float64, chPoint chan
CCSeqImpact(PointIn []float64, chPoint chan
Solve(AlphaIn float64, BetaIn float64, MaxItn
TestPoint(PointIn []float64) (Status, NINF, N
NewPoints1(Round int)
NewPoints2(Round int)
SwarmSearch4() (Status int)
SwarmSearch5() (Status int)
CheckForIdenticalPts() (SomeIdentical bool)
IdenticalPts(Point1 []float64, Point2 []float64
UpdateIncumbent(PointIn []float64, SINFin f
UpdateIncumbents(PointIn []float64, SFDin,
UpdateIncumbentSFDforNINF(PointIn []floa
UpdateIncumbentSFD(PointIn []float64, SFD
Project(Pt0 []float64, UpdateVector []float64)
SwarmProject(Pt0 []float64, UpdateVector []
SwarmProject1(Pt0 []float64, UpdateVector [
GetSFD(PointIn []float64) (Status int, SFDout
GetMultiplier(X0, X1 []float64, CBIndex int, C
QuadApprox(Pt0, Pt1, Pt2 []float64, Y0, Y1, Y
AngleConCon(Con1, Con2 int) (Status int, A
AngleConVarb(Con, Varb int) (Status int, An
AngleFV(Con1 int, Mult1 float64, Con2 int, C
UpdateSwarm(PtIn []float64, SFDin float64, N
UpdateSwarm1(PtIn []float64, PtNum int, SF
GetCV(Pt []float64, Mode int) (Status int, CV
GetCV1(Pt []float64) (Status int, CV0 []float64
EnforceBounds(PtIn []float64) (PtOut []float
SortByImpact () ()
(s BySum) Len() int
History
Console
Search
<terminated> CCLPv4.go [Go Application] CCLPv6.exe
1.51 Total Time (s)
0.29200000000000004 Model Read-in Time (s)
1.218 Calculation Time (s)
No feasible point found. Incumbent SFD: 238.77760240176588 NINF: 1051
Smallest NINF: 2147483647
Total incumbent updates 51

Go: Conclusions

- Easy to learn
  - Mostly intuitive
  - Good online learning, reference, and practice tools
- Concurrency easy to program
  - Takes some practice if new to concurrency
- *Very* fast compilation, fast execution
- Multi-platform (Windows, linux, Apple)
- Good IDEs
- Free
- *But* relatively little supporting software for optimization (yet)
- Bottom line:
  - Good language for general coding of parallel algorithms for optimization
    - Supported by Google, so likely to be around for a while
- Potential alternative: Julia

Julia Language: Design Criteria

- Targets high-performance numerical and scientific computing
  - Large mathematical function library
- Dynamic language
- Parallel and distributed computing built-in
- Call Fortran/C libraries directly
  - Call other languages via libraries, e.g. Python
- Garbage collection

Helpful Features of Julia

- Matlab-like features:
  - Interactive shell
  - Define arrays simply
  - Plotting (via libraries)
- Runs very quickly (C speed)
  - Uses the LLVM JIT compiler
- Free and open source

Concurrent Programming in Julia

- Message-passing interface
- Remote reference
  - Used by any process to refer to an object stored on a particular process
- Remote call
  - Request by one process to call a function on another (or the same) process: spawns a concurrent call
  - Generates a remote reference
  - Can *wait* and *fetch* result
  - @spawn macro makes this easier

Coroutines: produce and consume

- Coroutines (tasks) are like goroutines
  - Lightweight interruptible threads
- Produce and consume data is like a channel

Julia Resources

- Julia info: http://julialang.org/ or http://istc-bigdata.org/index.php/open-big-data-computing-with-julia/
- Many optimization interfaces already:
  - JuliaOpt umbrella group for Julia-based optimization projects: http://www.juliaopt.org/
  - JuMP modelling language for math programs: https://jump.readthedocs.org/en/release-0.4/jump.html
  - Connections to many solvers: COIN Cbc/CLP, Cplex, Gurobi, IPOPT, Knitro, etc.

Comparing Go and Julia

	Go	Julia
Writing concurrent programs	-Easy for multi-core - not obvious for distributed systems	-Syntax more convoluted - built-in support for distributed systems
Matlab-like features	None	-Arrays -Interactive system
Syntax	-Simple, unambiguous, clear -Simple dependency system	A little more convoluted
Optimization libraries, tools, community	Small	Extensive, links to solvers, modelling language, active community
Compilation speed	Blazing. Like working with a scripted language	Just-in-time compiler is fast
Execution speed	Like C or Fortran	Like C or Fortran
Calling other languages	- C via libraries	-Directly call C, Fortran -Call Python via libraries

Conclusions

- Go and Julia are good choices for concurrent programming
- Go is simpler, but has less uptake in the optimization community
- Julia has good support in the optimization community

Looking for a good post-doc

- Topic: concurrent optimization
- About Ottawa, Canada:
  - Canada's capital
  - Many fine museums, outdoor festivals
  - Canoeing, kayaking, hiking, camping, skiing
  - Close(ish) to Montreal
  - English/French bilingual
- *Must like snow*