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

• Statements are minimal and simple:
  ◦ Only one kind of loop: \textit{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)
• \textit{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.
Concurreny example

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
- 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 Gocclipse): https://code.google.com/p/gocclipse/
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


- Many optimization interfaces already:
  - 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

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