R: A Dynamic Language for Statistical Computing

Luke Tierney

Department of Statistics & Actuarial Science
University of Iowa

September 3, 2010

Introduction

- R is a language for data analysis and graphics.
- Originally developed by Ross Ihaka and Robert Gentleman at University of Auckland, New Zealand.
- Now developed and maintained by a distributed group of 19 people.
- R is based on the S language developed by John Chambers and others at Bell Labs.
- R is widely used in the field of statistics and beyond, especially in university environments.
- R has become the primary framework for developing and making available new statistical methodology.
- Most extension packages are available through CRAN or similar repositories.

The R Language

- R is a dynamic language.
- Lazy evaluation is used for function arguments.
  - everything is a function, including flow control
  - sometimes used to capture argument expressions and evaluate in non-standard ways
- Managing data is an important part of the language.
- Typical usage is initially interactive
  - read some data into variables
  - make some plots
  - compute some summaries
  - more sophisticated modeling steps
  - develop simple functions to replicate analysis
  - ...

The R Language (cont.)

- R is a vector/array language
  - similar in some ways to MATLAB, APL
  - if x is a vector of data then

    ```
    (x - mean(x)) / sd(x)
    ```

    produces a standardized version.
- Explicit looping is often unnecessary.
- Writing loops can be necessary/convenient at times.
- The current interpreter is rather slow, making explicit loops using scalar-sized values slower than in should be.
- R packages can include code written in C or FORTRAN
  - to improve performance
  - to allow use of existing code implementations

Parallel Computing in R

- R is single-threaded.
- Two approaches have been used to add parallel computation:
  - explicit parallel computing by creating separate communicating R processes (e.g. snow, Rmpi)
  - implicit approaches, including
    - using a multi-threaded BLAS
    - parallelizing vectorized operations and matrix/array operations using OpenMPI (e.g. pnmath)
- There is also work on using GPU-based parallel computing within R packages.

Some Directions for the R Engine

Some directions I hope to work on in the next 12 to 18 months:

- Adding parallelized versions of for vectorized operations and simple matrix operations to the core distribution.
- Byte code compilation of R code.
- Increasing the limit on the size of vector data objects.

Parallelizing Vector and Matrix Operations

- Conceptually, vectorized math functions are easy to parallelize.
- Parallelizing loops for short vectors will often slow the code down.
- Break-even points vary with hardware/operating system.
- A strategy for determining and using break-even points is needed.
- A preliminary implementation is available as the pnmath package.
- Basic issues carry over to simple matrix operations, like colSums, and operations producing matrix results from vectors, like dist.
- Being able to easily turn off parallel computation may be important to avoid contention in explicit parallelization contexts.

Byte Code Compilation

- The current R implementation
  - parses code into a *parse tree* when the code is read
  - evaluates code by interpreting the parse trees.
- Compiling to byte code for a suitable virtual machine should
  - improve performance
  - help enable further improvements
- Efforts to add byte code compilation to R have been underway for some time.
- Current R implementations include a byte code interpreter, and a preliminary compiler is available from my web page.
- The current compiler and virtual machine produce good improvements in a number of cases.
- However, better results should be possible with a new virtual machine design.
- This redesign is currently in progress.

Byte Code Compilation (cont.)

- The new virtual machine will support
  - avoiding the allocation of intermediate values when possible
  - more efficient variable lookup mechanisms
  - more efficient function calls
  - possibly improved handling of lazy evaluation
- Other directions to explore include
  - opcode fusing for parallelization
  - declarations (sealing, scalars, types, strictness)
  - advice to programmer on possible inefficiencies
  - machine code generation using LLVM or other toolkits
  - replacing the interpreter entirely

Byte Code Compilation (cont.)

- A simple, artificial, example:

```
p1 <- function(x) {
    for (i in seq_along(x))
        x[i] <- x[i] + 1
    x
}
```

  In R this is essentially equivalent to `x + 1`.

- Some timings for `x <- rep(1, 1e7)` on an `x86_64` Ubuntu laptop:

Method	Time	Speedup
Interpreted	32.730	1.0
Byte compiled	9.530	3.4
Experimental	1.128	29.0
x+1	0.119	275.0

Increasing the Limit on the Size of Vector Objects

- Currently The total number of elements in a vector cannot exceed $2^{31} - 1 = 2,147,483,647$
- This is fairly large, but is becoming an issue with larger data sets with many variables on 64-bit platforms.
- Can this limit be raised without
  - breaking too many existing packages
  - requiring the rewriting of too much C code?
  - breaking compatibility with external software, such as BLAS
  - breaking ability to handle saved work spaces across platforms
- Possible directions:
  - changing the integer data type
  - adding a long integer data type
  - allowing floating point numbers to be used for length and index calculations.