Towards a New Statistical Computing System

Ross Ihaka & Brendan McArdle
University of Auckland
Outline

- So what’s wrong with R anyway?
- The design space and some choices we’ve made.
- Some consequences of these choices.
- Where things stand at the moment.
- Some other possibilities.
A Simplified Example

For each of $y_1, \ldots, y_m$, find the closest value in $x_1, \ldots, x_n$.

The solution should take the form of a function.

    nearest(x, y)

Here:

    x is a vector that contains the x values,
    y is a vector that contains the y values,
    nearest returns the vector of closest values.
Strategy One – R (Vectorised) Style

An experienced R programmer would produce the following type of solution.

```r
> nearest =
function(x, y)
x[apply(outer(y, x,
  function(y, x)
  abs(y - x)), 1,
  function(x)
  which(x == min(x))[1])]
```
Strategy One – Explanation

Compute the $m \times n$ matrix $D$ of distances between each $y$ value (row index) and each $x$ value (column index).

$$d_{ij} = |y_i - x_j|$$

Obtain the indices of the minimum value in each row. (The index of the closest $x$ value to each $y$ value.)

$$j_i = \arg\min_j d_{ij}$$

Return the vector of $x$ values corresponding to these values.

$$\{x_{ji} : i = 1, \ldots, m\}$$
Strategy Two – Naive Style

> nearest =
    function(x, y) {
        xmatch = numeric(length(y))
        for (i in seq(along = y)) {
            dist = Inf; xv = NA
            for(j in seq(along = x)) {
                ndist = abs(y[i] - x[j])
                if (ndist < dist) {
                    dist = ndist; xv = x[j]
                }
            }
            xmatch[i] = xv
        }
        xmatch
    }
Strategy Two – Explanation

The following pseudo code explains the function.

Allocate space for the computed values.
For each value $y$ in $y$,
determine the closest $x$ value in $x$ to $y$.
Return the vector of closest values.

The inner loop compares the current minimum distance with the distance between $x$ and $y$ and updates that minimum value and its associated $x$ value if a smaller value has been found.
Cost Evaluation

Strategy One

ONE array of size $m$ is allocated to contain the matches.

THREE temporary arrays of $m \times n$ elements are allocated during evaluation.

Looping over arrays takes place in C.

Strategy Two

ONE array of size $m$ is allocated to contain the matches.

No additional temporary space is allocated.

Looping over arrays takes place in R.
Problems with Current Systems

- Tree-walking interpreters.
  - Inherently slow.
  - No optimisation.

- Call-by-value semantics.
  - Produces vast amounts of data copying.
  - Prevents some useful programming techniques.

- No scalar data types.

- The problems go unnoticed because systems have gotten much faster and memory is cheap.
The Design Space

- Try to make existing systems run faster.
  - Refine the existing interpreter.
  - Jan Vitek et al, *trace compilation*.

- Use automatic translation of high-level descriptions into low level equivalents.
  - Sholz, Grelck et al, *Single Assignment C*.

- Develop new languages that are less hostile to compilation.
Some Ways to Avoid Current Problems

- Use machine resources to refine and optimise code.
  - Traditional compilation techniques.
  - Automagically rewrite specifications for solving problems in ways that are more efficient.

- Avoid unnecessary copying at all costs.
  - Use reference counting to avoid unnecessary copying.
  - Change language semantics to be call-by-reference.

- Introduce and use scalar data types.
Our Design Choices

• Full ahead-of-time compilation.
  – Initially, byte-coded virtual machine.
  – Later, machine code generation via LLVM or GCC.

• Call-by-reference semantics.

• Support for scalars.
  – Full numeric tower, including integers, floats of various sizes, bignums and rationals.

• Declarations
  – Mandatory scope declarations.
  – Optional type declarations.
(Hoped for) Consequences of Design Choices

- TWO to THREE orders of magnitude speedup for interpreted code (100 to 1000 times faster).
  - This kind of speedup should make it possible to do qualitatively different things.

- Much less copying.
  - Copying will be under programmer control.
  - Arguments can be overwritten.

- A particular goal is to be able to stream data rather than holding it in memory.
Progress

- We have an approach that lets us represent and reason about code at a high level.

- We still need to be able to generate machine instructions from this high-level representation. (LLVM and GCC will help.)

- Currently, there is no syntax. (Syntax is simultaneously both trivial and very important.)

- Unfortunately, there seems to be no appetite for funding this type of work. This makes progress slow.
Other Possibilities

- This is just one approach.

- It provides a quick way to side-step the problems apparent in R and similar software.

- Once the compiler framework is in place it should be possible to try other models.
  - E.g. Call-by-value with reference counting.

- We have yet to experiment with macros and object models.
Summary

• New computing environments for statistics are needed.

• They can be created by looking for incremental improvements to existing systems or by creating something new.

• Completely new systems offer the possibility of a quantum leap in performance.

• The effort has been constrained by lack of resources, but should show results in the next year or two.