Code Analysis and Parallelizing Vector Operations in R

Luke Tierney

Department of Statistics & Actuarial Science
University of Iowa

DSC 2007
February, 2007

Introduction

- R is a language for interactive data analysis and graphics.
- Some features intended to make interactive use easier:
  - Named arguments.
  - Partial matching of argument names.
  - Lazy evaluation of arguments and use of argument expressions.
- R is also
  - a powerful high level language
  - well suited to expressing complex statistical computations
- Two concerns:
  - correctness of code
  - performance

Improving Correctness of R Code

- The R package system provides an infrastructure for testing:
  - examples are run
  - code in a `tests` directory is run

  by R CMD check.
- Unit testing frameworks have been developed, e.g. `RUnit`.
- Testing is essential there are issues:
  - Most tests need to be created manually.
  - Complete coverage is hard to achieve.
- Static code analysis is a useful supplement.

Static Code Analysis

- Static code analysis examines source code without executing it.
- Analysis can look at
  - individual expressions
  - larger patterns of expressions
  - relationships among functions and modules
- For C, for example,
  - compilers carry out basic code analysis and report errors
  - more sophisticated tools have been developed recently
  - these have been used successfully on the Linux kernel
- Most code analysis involves approximations
  - not all issues can be detected (undecidable)
  - there are false positives
  - being able to tune specificity/sensitivity is helpful
  - methods of ranking possible issues are useful
  - statistical error ranking methods have been studies (Engler et al.)

Code Analysis for R

The R language presents some unusual challenges:

- Whether a variable is global or local may depend on data.
- Functions can create new variables in their callers.
    - used in glm.fit with family$initialize
- Functions can remove variables from their callers.
- Some functions use nonstandard evaluation of some arguments.
    - library, curve, link functions
- Evaluation context may not be statically available
    - with function

- `codetools` analyzes expressions in the context of visible definitions.
- Some of the things it can detect:
  - Calls not consistent with visible function definitions.
  - Bad assignment expressions.
  - Improper use of ..., next, or break.
  - Undefined functions or variables used.
  - Calls with no visible function definition.
  - Local variables assigned but not used.
  - Parameters changed by assignment.
  - Multiple incompatible definitions of a local function.

The codetools Package
Usage

- The two main functions are
  - checkUsage for checking individual R functions.
  - checkUsagePackage for checking a (loaded) package.
- A range of arguments are provided select classes of warnings:
  - all: enable all warnings.
  - suppressLocal: suppress all local variable warnings.
  - suppressParamUnused: suppress warnings about unused parameters.
  - suppressLocalUnused: suppress warnings about unused local variables
  - suppressUndefined: suppress warnings about undefined variables.
  - ...
- A better approach to managing which warnings to show is needed.

A function definition with some possible errors:

```
g<-function(x, exp = TRUE) {
    if (exp)
        exp(x+3) + ext(z-3)
    else
        log(x, bace=2)
}
```

The code analysis:

```
> checkUsage(g, name = "g")

g: no visible global function definition for 'ext'
g: no visible binding for global variable 'z'
g: possible error in log(x, bace = 2): unused argument(s) (bace = 2)
```

The codetools Package
Some Examples

Running checkUsagePackage on base in R 2.4.1 produces

```
> checkUsagePackage("base")

...
substring: local variable 'x' assigned but may not be used
...
```

The definition of substring is

```
substring <- function (text, first, last = 1e+06)
{
    if (!is.character(text))
        x <- as.character(text)
    n <- max(lt <- length(text), length(first), length(last))
    if (lt && lt < n)
        text <- rep(text, length.out = n)
    substr(text, first, last)
}
```

An obscure bug, but a bug nonetheless.

- Currently `codetools` is being used
  - by a number of programmers for checking their packages
  - for screening of CRAN submissions
  - in the `weaver` package
- Current version is available at
  `http://www.stat.uiowa.edu/~luke/R/codetools`
- May be made available via CRAN soon.
- Should eventually be integrated into R and be available within R CMD check.

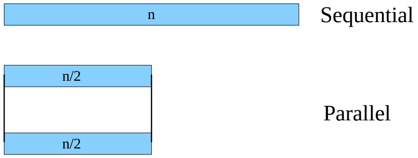
The `codetools` Package
Future Directions

- Develop a framework for adding rules, checks.
- Look at larger units than expressions.
- Explore allowing declarations to clarify ambiguities, intent?
- Identify common idioms that
  - are often errors (e.g. FAQ 7.31)
  - represent common inefficiencies
- Interactive features, such as
  - call graph display
  - editor integration support
- ...

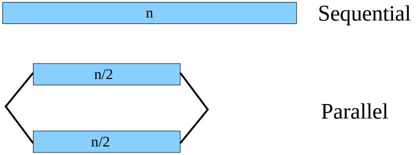
Parallelizing Vector Operations

- Multi-core processors are becoming increasingly common:
    - Many laptops have dual core processors.
    - Quad core workstations are available and affordable.
- In principle this allows speedups by a factor of 2 or 4.
    - This is attractive if "free," but
    - maybe not enough to justify extra user programming.
    - Useful if it can be activated automatically.
- Already possible in linear algebra by using a threaded BLAS.
- Possible candidates for automatic parallelization:
    - Vectorized arithmetic operations.
    - Some uses of apply family and sweep.

- Basic idea for computing f(x[1:n]) on a two-processor system:
  - Run two worker threads.
  - Place half the computation on each thread.
- Ideally this would produce a two-fold speed up.



- Reality is a bit different:



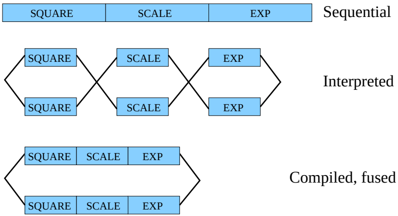
- There is synchronization overhead.
- Parallelizing will only pay off if n is large enough.
  - For some functions, e.g. qbeta, $n \approx 10$ may be large enough.
  - For some, e.g. qnorm, $n \approx 1000$ is needed.
  - For basic arithmetic operations $n \approx 30000$ may be needed.
- Careful tuning to insure improvement will be needed.

The Connection: Compilation

- Developing a byte code compiler for R is an ongoing project.
- The current `codetools` implementation is a by-product.
- Compilation will also be useful for parallelizing vector operations:
  - Many vector operations occur in compound expressions, like

    ```
    exp(-0.5*x^2)
    ```

  - A compiler may be able to fuse these operations:



- Compilation may also allow many simple uses of apply functions and sweep to be parallelized.

Challenges

- Tuning issues:
  - Hardware/OS may play a role.
  - Competing system usage may be important.
  - Performance may vary with inputs.
  - Load balancing may be useful.
- Error handling and user interrupts.
- Parallelization interface for package use.
- Extensible byte code for package use.
- Generic functions and non-default methods.
- Declarations may be useful.

Summary

- Two concerns:
  - correctness
  - performance
- There is a strong synergy:
  - Code analysis tools help with automated performance improvement.
  - They may also be able to suggest opportunities for rewriting.
- Work on code analysis has progressed, but much more can be done.
- Parallelization work is just starting but seems promising.
- Hopefully there will be significant progress in the near future.