R Code Optimization and Package Creation

Matthew T. Pratola
Dept. of Statistics and Actuarial Science
Simon Fraser University
Fall, 2003.

Outline

- R Code Optimization
  - How and why
  - Results
- R Package Creation
  - Basic steps
  - Other...

R Code Optimization

- R is an interpreted language
- Easy to author code to reflect an “idea” due to friendliness of the language
- Worry about efficiency later...

Why might it be slow?

- Computers operate on a few basic datatypes... integers, floats, chars to name the main types.
- R doesn't directly operate with these underlying datatypes, instead it works on “abstract representations” of these (and more complex) types.
- Why? So strange things like NA's, data frames, etc, can be easily created, accessed and manipulated by the end user.

Why might it be slow?

- Read “abstract representations” => think “more memory used to represent a given unit of data”
- This means that for ever operation you do on your data, your computer must spend extra time moving around all this extra memory, much of which doesn't directly have anything to do with the value of interest.

How to make it faster?

- Standard approaches
  - Investigate algorithmic design of your program
  - Optimize memory access patterns**
  - Others....

Results for RecodeHaplos

- Numbers for example_dat file and 4loci.txt
- Run on Athlon K7 500MHz w/ 128MB PC100 memory – ie much slower than stat-db
- Used repeating test loop with n=10 iterations
- example_dat numSNPs=2
- 4loci.txt numSNPs=4

Results for RecodeHaplos

example_dat – time in seconds					
Orig	Ver.2	Ver.3	Ver.4	Ver.5	Ver.6 (w/missing)
118.62	76.92	62.72	....13.58.....7.95...	8.14	8.01
100.00%	-35.15%	-47.13%		-93.14%	-93.25%

4loci.txt – time in seconds					
Orig	Ver.2	Ver.3	Ver.4	Ver.5 (w/missing)	Ver.6 (w/missing)
182.12	21.75		13.29	193.27	15.66
100.00%	-88.06%		-92.70%	6.12%	-91.40%

How?

- Avoid using dataframes within your code
- Simple example:
  - m=matrix(nrow=2000,ncol=1)
  - for(i in (1:2000)){m[i,1]=1}
  - m=as.data.frame(m)
  - for(i in (1:2000)){m[i,1]=1}
- How much slower do you think it is?

How?

- 0.05s vs. 5.47s !!!
- Fix: Only convert your data to a dataframe at the end of the program – ie – for the user.
- Other good trick – avoid the use of rbind/cbind if possible during a long (intensive) loop.
- Instead, pre-allocate the entire (or, if not possible, a very large) array initialized to NA's, then directly assign the values by using a row (col) counter.

How?

- Eg:
  - x=c(1,1,1,1)
  - for(i in 1:5000){x=rbind(x,c(i,i,i,i))}
  - x=matrix(nrow=5000,ncol=4)
  - for(i in 1:5000){x[i,]=c(i,i,i,i)}
- 0.15s vs. 12.45s!!!

How?

- Another trick – often it may be the case that you have a main loop in your code, perhaps updating many matrices
- But, it could be that there is no interdependence amongst the matrices you are updating
- Consider:
  - for(...) {
  - Blah[]=...
  - Blah2[]=...
  - ... }

How?

- Versus:
  - for(...){
  - Blah[]=...
  - }
  - for(...){
  - Blah2[]=...
  - }
  - ...

How?

- This second approach can often yield a reasonable gain in a very long, intensive loop.
- Real compilers (ie C, Fortran, ...) do this automatically, but R does not.

How?

- A final hint – avoid excessive use of string manipulation functions, eg:
  - as.numeric(unlist(strsplit(heteroStates1,split="")))
- Working around this provided decent improvement in this particular case

How?

- Other, more detailed optimizations can yield some improvement, although past these simple approaches discussed, the percentage improvement will quickly fall to single digit levels.
- Could take a look at the differences in the original and revised versions for more insight...

How? - Profiling

- Use Rprof to gain insight into where your code is slow
  - Rprof(“Rprof.dat”)
  - <code>
  - Rprof(NULL)
  - R CMD Rprof Rprof.dat > profile.txt

Conclusion

- Speedup is possible in R using a few tricks
- Sometimes these are easy to implement, sometimes they are not
- Try to achieve a balance between code that conveys an idea and code that runs quickly
- Try simple approaches that are within your comfort zone

R Package Creation

- Distribute your own statistical procedure(s) along with help files, examples, demo's, etc.
- Package Structure
- Documentation
- Building & Checking

Package Structure

- Root directory – ie hapassoc/
- Should contain files:
  - DESCRIPTION
  - INDEX
- Possible subdirectories:
  - R*, data*, demo, exec, inst, man*, src, tests

“ DESCRIPTION” File

- Contains basic information about the package
- Eg:
  - The package name
  - Version, date, license, URL, depends
  - Title, author(s), maintainer
  - Description

“ INDEX” File

- Is optional
- Contains one line containing name and description of each important object in the package.
- Eg:
  - RecodeHaplos Recode SNP data into...
  - CheckHaplos Estimation of Haplotype...

Subdirectories

- R – contains files ending in .R, .S, .q, .r or .s
- man – contains documentation files with the extension .Rd
- data – contains data files which can be loaded with the data() command. Useful for example code placed in Rd files
  - Can be plain .R (.r) code, tables (.tab, .txt, .csv) or saved images (.Rdata, .rda)

Documentation

- Rd is a simple markup format, using LaTeX as its backend
- The main parts to a .Rd documentation file:
  - Header section
  - Body
  - Footer section
- The header and footer are mandatory

Header

- Information about name of file, title, description and usuage information
  - \name{...}
  - \alias{...}
  - \title{...}
  - \description{...}
  - \usage{...}

Body

- Arguments, examples, links to related functions
  - \arguments{...}
  - \value{...}
  - \examples{...}
  - \seealso{...}

Footer

- Contains keyword information
  - \keyword{...}
- Keywords are standardized
- There must be one keyword specified, but can be more than one if the functionality of the procedure falls into more than one class
- Keywords listed in R_HOME/doc/KEYWORDS

A few more points...

- A rough Rd file can be generated by prompt(foo) for some function foo
- Datasets should be documented in a similar fashion
- There are further markup functions available which may sometimes be useful... (see docs)
- You can check the output of your Rd file by running:
  - R CMD Rd2dvi blah.Rd OR Rd2txt blah.RD

Building and Checking

- Build package:
  - R CMD build hapassoc/
- This will create the file hapassoc_{version}.tar.gz

Building and Checking

- Check package:
  - R CMD check hapassoc
- A lot of things are checked in this process
- If there was a problem, you can look in the [pkgname].Rcheck subdirectory for clues.
  - ie, above command will create hapassoc.Rcheck

R Package Submission

- Upload your .tar.gz to:
  - ftp://ftp.ci.tuwien.ac.at/incoming
- Send a message to cran@r-project.org about it