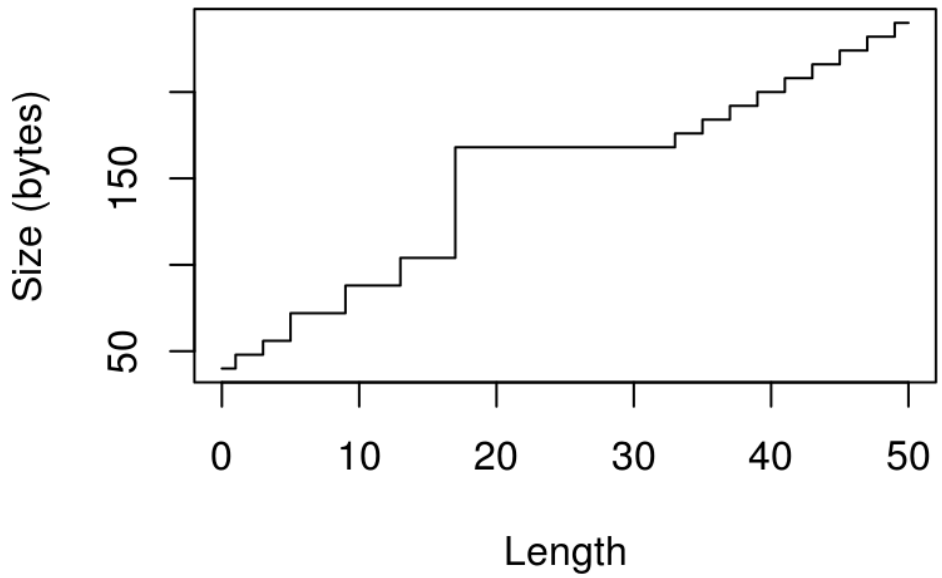
Large-scale data analysis in R

Peter Carbonetto

Research Computing Center and the Dept. of Human Genetics
University of Chicago



A brief intro

We will develop essential skills for large-scale data analysis in R.
In particular, we will learn how to:

1. Run R analyses *non-interactively*.
2. Determine memory needs.
3. Make efficient use of memory.
4. Speed up our analyses using. . .
   - ▷ Simple parallelization techniques.
   - ▷ Interfacing to C code (Rcpp).

A brief intro

- This is a *hands-on workshop*—you will get the most out of this workshop if you work through the exercises on your computer.
- All the examples are intended to run on the RCC cluster.

Software we will use today

1. R
2. Python3
3. Slurm
4. A few R packages: data.table, matrixStats, parallel, Rcpp

These are already installed on midway2.

The “large” data set

- RegMap data: genetic and ecological data on *Arabidopsis thaliana* in a range of climates.
- From Joy Bergelson’s lab at U. Chicago.
- See Hancock et al (2011) *Science* 334, 83–86.

Outline of workshop

- Preliminaries
- Programming challenges:
  1. Setting up your environment for large-scale data analysis.
  2. Importing a large data set into R.
  3. Automating analysis of a large data set.
  4. Speeding up operations on large matrices.
  5. Multithreaded computing with “parLapply”.
  6. Using Rcpp to improve performance.

Preliminaries

- WiFi.
- Power outlets.
- Computer clutter.
- Reading what I type.
- Pace & questions (e.g., keyboard shortcuts).
- Yubikeys.
- What to do if you get stuck.

Preliminaries

- The workshop packet is a repository on GitHub. Go to:
  - ▷ github.com/rcc-uchicago/R-large-scale
- Download the workshop packet onto your computer.
- Open the slides PDF.

What's in the workshop packet

- **slides.pdf:** These slides.
- **slides.Rmd:** R Markdown source used to create these slides.
- **.R** files: R scripts we will run in the examples.
- **.sbatch** files: Slurm scripts we will run to allocate resources for our analyses on the cluster.
- **scale.cpp**: Some C++ code we will use to speed up one of the analyses.
- **monitor_memory.py:** Python script used to assess memory usage.

Outline of workshop

1. **Setting up your environment for large-scale data analysis.**
2. Importing a large data set into R.
3. Automating analysis of a large data set.
4. Speeding up operations on large matrices.
5. Multithreaded computing with “parLapply”.
6. Using Rcpp to improve performance.

Challenge #1: Setting up your HPC environment

- Aim: Configure your HPC environment for the next programming challenges.
- Steps:
  1. Connect to midway2.
  2. Download workshop packet onto midway2.
  3. Retrieve data set.
  4. Allocate a midway2 compute node.
  5. Launch R.
  6. Set up your R environment.
  7. Open another midway2 connection.

Connect to midway2

- **If you have an RCC account:** I'm assuming you already know how to connect to midway2. Use your preferred method. See: `https://rcc.uchicago.edu/docs/connecting`
- **If you do not have an RCC account:** I will provide you with a Yubikey. This will give you guest access (see the next slide).

Using the Yubikeys

- Prerequisites:
  1. SSH client
  2. USB-A port
- Steps:
  1. Insert Yubikey into USB port.
  2. Note your userid: `rccguestXXXX`, where `XXXX` is the last four digits shown on Yubikey.
  3. Follow instructions to connect to midway2 via SSH, replacing the cnetid with your `rccguestXXXX` user name: `https://rcc.uchicago.edu/docs/connecting`
  4. When prompted for password, press lightly on metal disc.

Please return the Yubikey at the end of the workshop.

Download workshop packet onto midway2

Once you have connected to a midway2 login node, download the workshop packet to your home directory on the cluster (**note:** there are no spaces in the URL below):

```
cd $HOME
git clone https://github.com/rcc-uchicago/
  R-large-scale.git
```

Retrieve the data set

Copy and decompress the data to your home directory:

```
cd $HOME/R-large-scale
cp ~pcarbo/share/regmap.tar.gz .
tar zxvf regmap.tar.gz
```

After taking these steps, this command should list two CSV files:

```
ls *.csv
```

Connect to a compute node

Set up an interactive session on a midway2 compute node with 8 CPUs and 18 GB of memory:

```
screen -S workshop
sinteractive --partition=broadwl \
  --reservation=workshop --cpus-per-task=8 \
  --mem=18G --time=3:00:00
echo $HOSTNAME
```

Launch R

Start up an interactive R session:

```
module load R/3.6.1
which R
R --no-save
```

Check your R environment

Check that you are running R 3.6.1:

```
sessionInfo()
```

Check that you are starting with an empty environment:

```
ls()
```

Check that you have the correct working directory—it should be set to the “R-large-scale” repository:

```
getwd()
```

Open another connection to midway2

- Open a *second* SSH connection, following the same steps as before.
- This second connection will be used to monitor your computations on the cluster.
- *At this point, you have completed the initial setup. You are now ready to move on to the next programming challenge.*

Outline of workshop

1. Setting up your environment for large-scale data analysis.
2. **Importing a large data set into R.**
3. Automating analysis of a large data set.
4. Speeding up operations on large matrices.
5. Multithreaded computing with “parLapply”.
6. Using Rcpp to improve performance.

Challenge #2: Importing a large data set into R

- Aim: Use the “data.table” R package to quickly read a large data set into R.
- Steps:
  1. Try importing the data using “read.csv”.
  2. Import data using “fread” from the data.table package.
  3. Time how long it takes to import using fread.

Import data using `read.csv`

Our first aim is a simple one: read the RegMap genotype data into R. First, try this using the "read.csv" function:

```
geno <- read.csv("geno.csv",check.names = FALSE)
```

Note: You can tell R to stop running the code at any time by typing "Control-C". (If "Control-C" doesn't work, I will give you an alternative.)

Import data using data.table package

Try again using the **data.table** package:

```
library(data.table)
geno <- fread("geno.csv",sep = ",",header = TRUE)
class(geno) <- "data.frame"
```

Timing the data import step

How long does it take to run “fread” on the RegMap data?

```
timing <- system.time(
  geno <- fread("geno.csv",sep = ",",
                header = TRUE))
class(geno) <- "data.frame"
```

Outline of workshop

1. Setting up your environment for large-scale data analysis.
2. Importing a large data set into R.
3. **Automating analysis of a large data set.**
4. Speeding up operations on large matrices.
5. Multithreaded computing with “parLapply”.
6. Using Rcpp to improve performance.

Challenge #3: Automating analysis of a large data set

- Aim: Develop scripts to automate (1) the data analysis in R and (2) configuration of the environment.
- Steps:
  1. Run data analysis interactively.
  2. Automate the data analysis using a script.
  3. Make the script more flexible using command-line arguments.
  4. Automate environment setup and resource allocation using sbatch.

Run the analysis interactively

At this point, you should have a data frame with 948 rows and 214,051 columns containing the *A. thaliana* genotypes.

```
nrow(geno)
ncol(geno)
```

A common step in genetic analysis is to examine the distribution of minor allele frequencies. This involves taking the mean of each column:

```
maf <- sapply(geno,mean)
maf <- pmin(maf,1 - maf)
```

Now summarize the minor allele frequencies:

```
summary(maf)
```

Automate the data analysis using Rscript

Let's quit R:

```
quit()
```

Now re-run the RegMap allele frequency analysis using the script provided in the repository:

```
Rscript summarize_regmap_mafs.R
```

Automate the analysis for several data sets

Suppose you need to repeat your allele frequency analysis several data sets. The `summarize_regmap_mafs.R` script only works for one data set. So we create a new script "summarize_mafs.R" that is more flexible; it takes the name of the genotype data file as a command-line argument:

```
Rscript summarize_mafs.R geno.csv
```

Automate environment setup and resource allocation

Rscript automates the steps *within the R environment*, but it does not automate the steps taken before running R code. Typically, before running the R code you will need to:

1. Run bash commands to set up your (shell) environment.
2. Run Slurm commands to allocate computing resources.

This will do 1 & 2, then run the analysis in R:

```
sbatch summarize_regmap_mafs.sbatch
```

Check the status of your analysis while it is running:

```
source set_slurm_env.sh
squeue -u cnetid
```

Run this command to check jobs after they are completed:

```
sacct -u cnetid
```

Automate environment setup and resource allocation

We can also implement an sbatch script that takes a command-line argument:

```
sbatch summarize_mafs.sbatch geno.csv
```

Followup challenge

Suppose you had to re-run this allele frequency analysis for 1,000 large data sets. How would you design your R and sbatch scripts to implement this analysis efficiently?

Outline of workshop

1. Setting up your environment for large-scale data analysis.
2. Importing a large data set into R.
3. Automating analysis of a large data set.
4. **Speeding up operations on large matrices.**
5. Multithreaded computing with “parLapply”.
6. Using Rcpp to improve performance.

Challenge #4: Speeding up operations on large matrices

- Aim: Take advantage of multithreaded routines in the OpenBLAS library to speed up your matrix computations.
- Steps:
  1. Import data into R.
  2. Compute “kinship” matrix.
  3. Compute “kinship” matrix again using multithreading.

Import genotype data as a matrix

Re-launch R, and load the RegMap genotype data again:

```
library(data.table)
geno <- fread("geno.csv",sep = ",",header = TRUE)
```

Convert the genotypes to a matrix:

```
geno <- as.matrix(geno)
storage.mode(geno) <- "double"
```

Compute kinship matrix

Another common task in genetic analysis is to compute the "kinship" matrix from the genotypes. (For non-quantitative geneticists, this is effectively the covariance matrix.) This can be done by computing the matrix cross-product:

```
K <- tcrossprod(geno)
```

How long does it take to compute this matrix?

```
timing <- system.time(K <- tcrossprod(geno))
```

Exploit multithreaded OpenBLAS

Most matrix operations in R on midway2 use OpenBLAS. This is a *multithreaded* library, meaning that it can take advantage of multiple CPUs to accelerate the computations. Re-run the kinship computations using the script:

```
Rscript compute_regmap_kinship.R
```

Now tell OpenBLAS to use 2 CPUs, and run the computations again:

```
export OPENBLAS_NUM_THREADS=2
Rscript compute_regmap_kinship.R
```

Do you get additional performance improvements with 4 threads and 8 threads?

Outline of workshop

1. Setting up your environment for large-scale data analysis.
2. Importing a large data set into R.
3. Automating analysis of a large data set.
4. Speeding up operations on large matrices.
5. **Multithreaded computing with "parLapply".**
6. Using Rcpp to improve performance.

Challenge #5: Multithreaded computing with “parLapply”

- Aim: Speed up computation of association *p*-values (associations between genetic variants and measured traits) using simple multithreading techniques.
- Steps:
  1. Compute *p*-values without multithreading.
  2. Set up R for multithreading.
  3. Compute *p*-values with parLapply.

Run the association analysis

Begin by starting the R environment in your interactive session. An association analysis for one climate variable—"maximum temperature of warmest month"—is implemented in `map_temp_assoc.R`. Run this in R:

```
source("map_temp_assoc.R")
```

This runs the code in `map_temp_assoc.R`, and keeps the results in your environment. This is the most expensive step:

```
pvalues <- get.assoc.pvalues(geno,pheno)
```

It applies `get.assoc.pvalue` to each column of `geno`.

- **Note:** Since it will take too long to compute all 200,000+ *p*-values, here <we only compute 10,000 of them.

Set up R for multithreading

Set up R to use all 8 CPUs you requested, and distribute the computation (columns of the matrix) across the 8 “threads”:

```
library(parallel)
cl   <- makeCluster(8)
cols <- clusterSplit(cl,1:10000)
```

Next, tell R which functions we will use in the multithreaded computation:

```
clusterExport(cl,c("get.assoc.pvalue",
                   "get.assoc.pvalues"))
```

Compute *p*-values inside "parLapply"

Now you are ready to run the multithreaded computation of association *p*-values using "parLapply":

```
f <- function (i, geno, pheno)
  get.assoc.pvalues(geno[,i],pheno)
timing <- system.time(
  out <- parLapply(cl,cols,f,geno,pheno))
```

Not done yet—you need to combine the individual outputs into a single vector of *p*-values.

```
pvalues <- rep(0,10000)
pvalues[unlist(cols)] <- unlist(out)
```

Check that the result is the same as before:

```
quantile(pvalues,c(0,0.001,0.01,0.1,0.25,0.5,1))
```

Did parLapply speed up the p-value computation?

Halt the multithreaded computation

When you are done using parLapply, run “stopCluster”:

```
stopCluster(cl)
```

Outline of workshop

1. Setting up your environment for large-scale data analysis.
2. Importing a large data set into R.
3. Automating analysis of a large data set.
4. Speeding up operations on large matrices.
5. Multithreaded computing with “parLapply”.
6. **Using Rcpp to improve performance.**

Challenge #6: Using Rcpp to improve performance

- Aim: Implement slow R computations in C for better performance and less memory usage.
- Steps:
  1. Run centering & scaling without Rcpp, assessing runtime and memory usage.
  2. Re-run centering & scaling with Rcpp, assessing runtime and memory usage.

Center & scale the matrix

For some data analyses, is important to first “center” and “scale” the columns of the matrix so that each column has zero mean and standard deviation 1. This is implemented in `scale_geno.R`:

```
Rscript scale_geno.R
```

Assessing memory usage of “scale”

To measure memory usage accurately, use the provided Python script:

```
module load python/3.7.0
export MEM_CHECK_INTERVAL=0.01
python3 monitor_memory.py Rscript scale_geno.R
```

Center & scale the genotype matrix using Rcpp

R duplicates objects aggressively ("copy on modify"). This can be an issue with large objects.

- We can circumvent this by implementing the computations in C++.
- See files **scale.cpp** and **scale_geno_rcpp.R** for how this is implemented using **Rcpp**.

Now re-run the centering & scaling with the Rcpp implementation:

```
python3 monitor_memory.py \
  Rscript scale_geno_rcpp.R
```

Is the C++ implementation faster? Does it reduce memory usage?

Recap

Some basic techniques we used today:

1. We automated analyses using **Rscript** and **sbatch**.
2. We used the **data.table** package for reading large data sets.
3. We used the **parallel** package for parallelizing computations.
4. We sped up matrix operations using **OpenBLAS multithreading**.
5. We interfaced to C++ code using the **Rcpp** package.