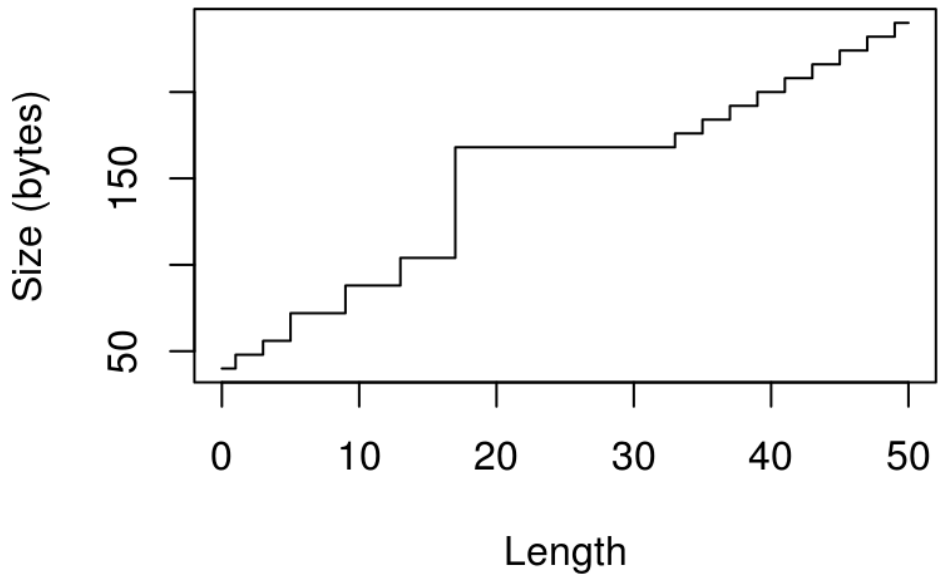
Large-scale data analysis in R

Peter Carbonetto

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



Workshop aims

Develop **essential skills** for large-scale data analyis in R, and apply these skills to a large(ish) data set. In particular:

1. Running R analyses *non-interactively* within a high-performance computing (HPC) environment.
2. Quantifying memory needs.
3. Making efficient use of memory.
4. Speeding up your analyses using...
   - ▷ Simple parallelization techniques.
   - ▷ Interfacing to C code (Rcpp).

Workshop aims

- 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. R packages: data.table, matrixStats, parallel and Rcpp

These are already installed on the RCC cluster.

The “large” data set

- RegMap data: genetic and ecological data on *Arabidopsis thaliana* in a range of climates.
- Developed by Joy Bergelson’s lab at the University of 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.
  3. Automating an 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.
- 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 to your computer.

What's included 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.
- **set_slurm_env.sh:** Shell commands to configure Slurm.

Outline of workshop

1. **Setting up your environment for large-scale data analysis.**
2. Importing a large data set.
3. Automating an 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.
  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: `https://rcc.uchicago.edu/docs/connecting`
  4. When prompted for password, press lightly on metal disc.

Download workshop packet

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 midway2 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/3.5.1
which R
R --no-save
```

Check your R environment

Check that you are running R 3.5.1:

```
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

- Follow the same steps as before to connect to a midway2 login node.
- 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.**
3. Automating an 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

- Aim: Use an R package specifically designed for reading data from large text files.
- Steps:
  1. Try importing 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".

Import data using data.table package

c 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.
3. **Automating an 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 the analysis and configuration of the environment.
- Steps:
  1. Run data analysis interactively.
  2. Automate the data analysis using Rscript.
  3. Automate the environment setup using sbatch.
  4. Make the analysis scripts more flexible using command-line arguments.

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. Since the RegMap genotypes are encoded as allele counts, the allele frequencies are easily computed by taking the mean ofeach column:

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

Now summarize the minor allele frequencies:

```
summary(maf)
```

Automate the data analysis using Rscript

Quit R, and re-run the analysis using the provided script:

```
Rscript summarize_regmap_mafs.R
```

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 command will do 1 & 2, then run the analysis:

```
sbatch summarize_regmap_mafs.sbatch
```

Check the status of your analysis while it is running:

```
source set_slurm_env.sh
squeue --user=<cnetid>
```

Run this command to check the status upon termination:

```
sacct --user=<cnetid>
```

Automate the analysis for many data sets

Suppose you want to repeat your analysis several data sets. One disadvantage of this script is that it will only work for one data set. See “summarize_mafs.R” for a similar script that is more *flexible*: it takes the name of the genotype data file as a command-line argument:

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

Likewise, we can implement an sbatch script that takes a command-line argument:

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

Outline of workshop

1. Setting up your environment for large-scale data analysis.
2. Importing a large data set.
3. Automating an 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. 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 3.5.1 on midway2 use the OpenBLAS library. This is a *multithreaded* library, meaning that it can take advantage of multiple CPUs to accelerate the computations. Re-run the kinship computations as before, but using Rscript this time:

```
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.
3. Automating an 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 association analysis (associations between genetic variants and measured traits) using simple multithreading (parallel computing) techniques.
- Steps:
  1. Run association analysis without multithreading.
  2. Set up R for multithreading.
  3. Run association analysis 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 the "map_temp_assoc.R" script.

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

Although we have data on over 200,000 genetic variants (SNPs), we limited the association analysis to only 10,000 SNPs because computing all 200,000+ *p*-values will take too long (several minutes). Let's use "parLapply" to speed up this computation.

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:p)
```

Next, tell R which functions we will use in the mulithreaded 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,p)
pvalues[unlist(cols)] <- unlist(out)
```

Check that the results 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.
3. Automating an 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: In this final example, we will show that even simple computations on large data sets in R can be very slow and use an excessive amount of memory.
- *Implementing the most intensive computations in C can sometimes give large (10–1000x) speedups.*
- Steps:

Center & scale the genotype matrix

For many analyses of genotype data (e.g., PCA), it is important to “center” and “scale” the columns of the genotype matrix so that each column has zero mean and standard deviation of 1. This computation is implemented in the “scale_geno.R” script:

```
Rscript scale_geno.R
```

Assessing memory usage of “scale”

To get a feel for the memory usage of “scale”, let’s compare against our earlier analysis of the minor allele frequencies. To measure memory usage accurately, let’s use the “monitor_memory” Python script:

```
module load python/3.5.2
export MEM_CHECK_INTERVAL=0.01
python3 monitor_memory.py \
  Rscript summarize_regmap_mafs.R
```

Now compare to the centering & scaling analysis:

```
python3 monitor_memory.py Rscript scale_geno.R
```

Does centering & scaling require more memory than computing MAFs?

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 a C++ function *that modifies the input matrix directly.*
- See files **scale.cpp** and **scale.geno.rcpp.R** for how this is implemented using the **Rcpp** package.

Now try monitoring memory usage in the script that uses the Rcpp implementation:

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

Does the C++ code improve the runtime and memory usage?

Recap

Some techniques we used today:

1. Automating analyses using R and sbatch scripts.
2. The **data.table** package for reading large data sets.
3. The **parallel** package for parallelizing computations.
4. Speeding up matrix operations using multithreading.
5. Interfacing to C code using the **Rcpp** package.