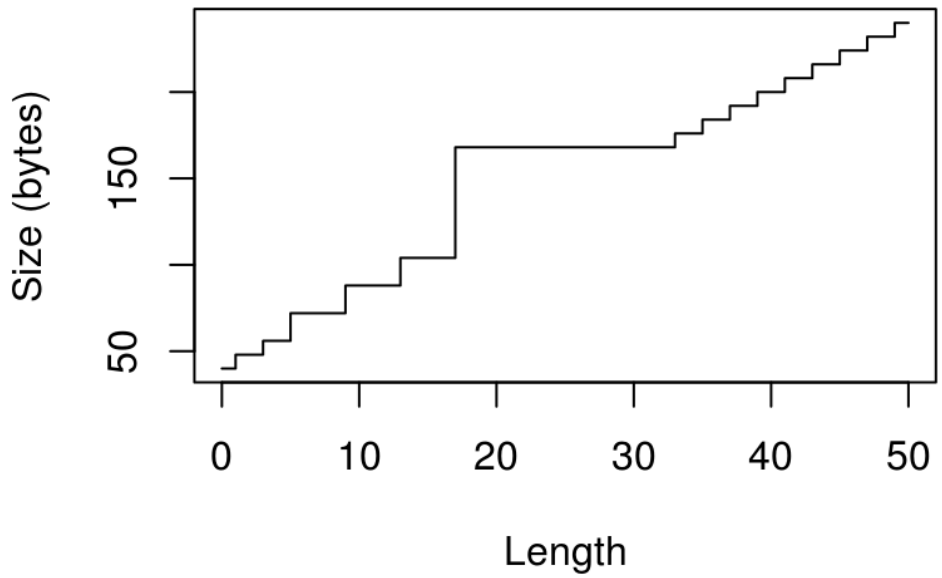
Large-scale data analysis in R

Peter Carbonetto

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



Workshop aims

1. Develop essential skills for large-scale data analyis in R, and apply these skills to a large-ish data set.
2. Learn how to use R **non-interactively** within a high-performance computing (HPC) environment.
3. Assess memory needs.
4. Make efficient use of memory.
5. Speed up your R analyses by...
   - ▷ using simple parallelization techniques.
   - ▷ interfacing to C code (Rcpp).
6. Learn through **live coding**—this includes learning from our mistakes!

It's your choice

You may choose to...

- Use R on the RCC cluster.
- Use R on your laptop.
- Pair up with your neighbour.
- Follow what I do on the projector.

Please note:

- Please use R, not RStudio.
- Some examples will only work on the RCC cluster.

Software we will use today

1. R
2. Python 3.x (for assessing memory usage).
3. Slurm (job scheduler on RCC cluster).
4. R packages **data.table**, **matrixStats**, **parallel** and **Rcpp**.

These software and R packages are already installed on the RCC cluster.

The “large-ish” data set

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

Outline of workshop

1. Initial setup.
2. Vignettes:
   - ▷ Vignette #1: Importing a large data set.
   - ▷ Vignette #2: Automating an analysis of a large data set.
   - ▷ Vignette #3: Speeding up operations on large matrices.
   - ▷ Vignette #4: Multithreaded computing with “parLapply”.
   - ▷ Vignette #5: Using Rcpp to improve performance.

Preliminaries

- WiFi.
- Power outlets.
- Reading what I type.
- Pace & questions (e.g., keyboard shortcuts).
- Getting stuck.

Download workshop packet

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

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

Optional: Download the repository to your laptop.

What's included in the workshop packet

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

Retrieve the data

Copy and decompress the data to your home directory:

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

After taking these steps, you should have two CSV files:

```
geno.csv
pheno.csv
```

Connect to a midway2 compute node

Set up an interactive session on a midway2 (“Broadwell”) 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 the R interactive environment:

```
module load R/3.5.1
R
```

Check your R environment

Check that your R environment is set up correctly:

```
sessionInfo()
ls()
getwd()
```

Establish another connection to midway2

This second connection will be used to monitor our computations on the cluster.

- *At this point, we have completed the initial setup. We are now ready to begin the first example.*

Outline of workshop

1. Initial setup.
2. Vignettes:
   - **Vignette #1: Importing a large data set.**
   - Vignette #2: Automating an analysis of a large data set.
   - Vignette #3: Speeding up operations on large matrices.
   - Vignette #4: Multithreaded computing with “parLapply”.
   - Vignette #5: Using Rcpp to improve performance.

Vignette #1: Importing a large data set

Our first aim is a simple one: read the RegMap genotype data into R. Let's try this using `read.csv`:

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

Import genotype data using `data.table`

Let's try again using the **data.table** package:

```
# install.packages("data.table")
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. Initial setup.
2. Vignettes:
   - Vignette #1: Importing a large data set.
   - **Vignette #2: Automating an analysis of a large data set.**
   - Vignette #3: Speeding up operations on large matrices.
   - Vignette #4: Multithreaded computing with “parLapply”.
   - Vignette #5: Using Rcpp to improve performance.

Vignette #2: Automating an analysis of a large data set

You should have a data frame with 948 rows and 214,051 columns containing the genotypes of the *A. thaliana* samples:

```
class(geno)
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, allele frequencies are column means:

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

Now summarize the minor allele frequencies:

```
summary(maf)
```

Automating the analysis using Rscript

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

```
Rscript summarize_regmap_mafs.R
```

Automating 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
```

Run these commands to 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>
```

Automating the analysis for many data sets

Suppose you want to repeat your analysis for several other (similar) genetic data sets. To reduce effort, you could re-write your R script to make it more *generic*. See “summarize_mafs.R” for a similar script that 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. Initial setup.
2. Vignettes:
   - ▷ Vignette #1: Importing a large data set.
   - ▷ Vignette #2: Automating an analysis of a large data set.
   - ▷ **Vignette #3: Speeding up operations on large matrices.**
   - ▷ Vignette #4: Multithreaded computing with “parLapply”.
   - ▷ Vignette #5: Using Rcpp to improve performance.

Vignette #3: Speeding up operations on large matrices

At this point, you have quit R, so re-launch R, and load the RegMap genotype data again:

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

This time, we convert the genotypes to a matrix of floating-point numbers:

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

Computing the genotype matrix cross-product

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))
```

Exploiting 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. First, re-run the kinship computations as before using Rscript:

```
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. Initial setup.
2. Vignettes:
   - Vignette #1: Importing a large data set.
   - Vignette #2: Automating an analysis of a large data set.
   - Vignette #3: Speeding up operations on large matrices.
   - **Vignette #4: Multithreaded computing with "parLapply".**
   - Vignette #5: Using Rcpp to improve performance.

Vignette #4: Multithreaded computing with "parLapply"

Often in genetic studies, we seek to identify associations between genetic variants and measured traits. Although there are very good software for running an association analysis (e.g., PLINK), here we will implement this analysis in R to illustrate simple multithreading (parallel computing) techniques.

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 your R environment for multithreading

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

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

Next, we need to 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 we 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))
```

We aren’t quite done—the output is a list object, and we need to combine the individual list elements into a single vector of p-values.

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

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. Initial setup.
2. Vignettes:
   - Vignette #1: Importing a large data set.
   - Vignette #2: Automating an analysis of a large data set.
   - Vignette #3: Speeding up operations on large matrices.
   - Vignette #4: Multithreaded computing with “parLapply”.
   - **Vignette #5: Using Rcpp to improve performance.**

Vignette #5: Using Rcpp to improve performance

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.*

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.