2. A brief intro

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

1. Run R analyses *non-interactively* on the cluster.

2. Determine memory needs.

3. Make efficient use of memory.

4. Speed up our analyses using:
   ▷ Simple parallelization techniques.
   ▷ Interfacing to C code (Rcpp).

3. 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 examples are intended to run on midway3.

4. Software we will use today

   1. R
   2. Python (optional)
   3. Slurm
   4. R packages: data.table, matrixStats, parallel, Rcpp

These are already installed on midway3.

5. The large (but not too 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.

6. 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 "mclapply".
    6. Using Rcpp to improve performance.

7. Preliminaries

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

8. 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.

9. 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.

10. 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 "mclapply".
6. Using Rcpp to improve performance.

11. Challenge #1: Setting up your HPC environment

- Aim: Configure your HPC environment for the next programming challenges.
- Steps:
    1. Connect to midway3.
    2. Download workshop packet onto midway3.
    3. Retrieve data set.
    4. Allocate a midway3 compute node.
    5. Launch R.
    6. Set up your R environment.
    7. Open a second midway3 connection.

12. Connect to midway3

- **If you have an RCC account:** I'm assuming you already know how to connect to midway3. 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).

13. 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 midway3 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.

14. Download workshop packet onto midway3

Once you have connected to a midway3 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
```

15. 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
```

16. Connect to a compute node

Set up an interactive session on a midway3 compute node with 8 CPUs and 18 GB of memory (replace `pi-hsimpson` with an account you have access to):

```
screen -S r_workshop
sinteractive -p caslake \
  --reservation=r_workshop \
  -c 8 --mem=18G -t 3:00:00 \
  --account=pi-hsimpson
echo $HOSTNAME
```

17. Launch R

Start up an interactive R session:

```
module load R/4.2.0
which R
R
```

18. Check your R environment

Check that you are running R 4.2.0:

```
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()
```

19. Open another connection to midway3

- Open a *second* SSH connection to midway3, 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.*

20. 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 "mclapply".
6. Using Rcpp to improve performance.

21. 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.

22. 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.)

23. 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"
```

24. Timing the data import step

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

```
t0 <- proc.time()
geno <- fread("geno.csv",sep = ",",header = TRUE)
class(geno) <- "data.frame"
t1 <- proc.time()
print(t1 - t0)
```

25. 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 "mclapply".
6. Using Rcpp to improve performance.

26. Challenge #3: Automating analysis of a large data set

- Aim: Develop scripts to automate (1) the data analysis in R and (2) configuring the computing 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.

27. 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)
```

28. Automate the data analysis using Rscript

Let's now attempt to automate this calculation. First, quit R:

```
quit()
```

Next, re-run the RegMap allele frequency analysis using the script provided in the git repository:

```
Rscript summarize_regmap_mafs.R
```

29. Automate the analysis for several data sets

Now imagine you need to repeat your allele frequency analysis several data sets. As it is written, summarize_regmap_mafs.R script only works for one data set. So we use another 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
```

30. 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 Slurm commands to allocate computing resources.
2. Run shell commands to set up your shell environment.

This script will perform steps 1 and 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 one of these commands to the check job after it has completed:

```
sacct -u cnetid -j jobid
sacct -u cnetid
```

31. Automate environment setup and resource allocation

We can also develop an sbatch script that takes command-line arguments:

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

32. Followup challenge

Suppose you had to re-run this allele frequency analysis for 100 large data sets. How would you design your R and sbatch scripts to implement this?

33. 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 "mclapply".
6. Using Rcpp to improve performance.

34. Challenge #4: Speeding up operations on large matrices

- Aim: Leverage multithreaded OpenBLAS routines speed up your matrix computations.
- Steps:
    1. Import data into R.
    2. Compute kinship matrix without multithreading.
    3. Compute kinship matrix using multithreading.

35. 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"
```

36. Compute kinship matrix

Another common task in genetic analysis is to compute the "kinship" matrix from the genotypes. (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 the kinship matrix?

```
t0 <- proc.time()
K  <- tcrossprod(geno)
t1 <- proc.time()
print(t1 - t0)
```

37. Exploit multithreaded OpenBLAS

Most matrix operations in R on midway3 use OpenBLAS. This is a *multithreaded* library, meaning that it can take advantage of multiple processors 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 it again:

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

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

Before continuing, let's revert to the original setting:

```
export OMP_NUM_THREADS=1
```

38. 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 "mclapply".**
6. Using Rcpp to improve performance.

39. Challenge #5: Multithreaded computing with "mclapply"

- 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 mclapply.

40. Run the analysis without multithreading

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`. Launch R again, and run this code 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 the `geno`
data frame. (To shorten the example, we are only computing
10,000 *p*-values instead of all 200,000 of them.)

41. Split computation

First, split the columns of the data frame into smaller subsets:

```
library("parallel")
cols <- splitIndices(10000,8)
```

42. Compute the *p*-values inside "mclapply"

Now we are ready to run the multithreaded computation of association *p*-values using "mclapply". Let's try first with 2 CPUs:

```
f <- function (i)
  get.assoc.pvalues(geno[,i],pheno)
t0  <- proc.time()
out <- mclapply(cols,f,mc.cores = 2)
t1  <- proc.time()
print(t1 - t0)
```

43. Combine mclapply outputs

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

```
pvalues2 <- rep(0,10000)
pvalues2[unlist(cols)] <- unlist(out)
```

Check that the result is the same as before:

```
range(pvalues - pvalues2)
```

Did mclapply speed up the p-value computation? Do you get further speedups with 4 or 8 (or even 160) CPUs?

44. 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.**

45. 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.

46. 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. Centering and scaling a matrix can be performed easily in R using the scale function. Unfortunately, it can be slow and memory-intensive. The scale_geno.R script illustrates this with the *A. thaliana* genotype data:

```
Rscript scale_geno.R
```

47. Assessing memory usage of "scale" (optional)

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

```
module load python/anaconda-2022.05
source activate base
export MEM_CHECK_INTERVAL=0.01
python3 monitor_memory.py Rscript scale_geno.R
```

48. 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 the scaling and centering iis implemented in C++ using **Rcpp**.

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

```
Rscript scale_geno_rcpp.R
```

Optionally, check memory usage of the Rcpp implementation:

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

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

49. Recap

Some basic techniques we used today:

1. We automated analyses using **Rscript** and **sbatch**.
2. We used **data.table** to quickly read large data sets.
3. We sped up matrix operations using multithreaded matrix operations (**OpenBLAS**).
4. We used **parallel** to parallelize computations.
5. We interfaced to faster C++ code using **Rcpp**.