Large-scale data analysis in R

November 13 & 14, 2017 / Peter Carbonetto / Research Computing Center / University of Chicago

The main aim of this workshop is give you a solid footing for transitioning from *interactive* (exploratory) data analysis in R to *non-interactive* (automated) large-scale data analysis within a high-performance computing environment. We will use the RCC's newest and most powerful cluster, *midway2*. This will involve:

1. making most effective use of computational resources on the RCC cluster (managing memory, using multiple CPUs simultaneously);
2. using R in combination with the SLURM job engine to run hundreds of data analyses simultaneously.

During this workshop, we will work with the *RegMap* data set—this data set contains genetic and climate variable data on *Arabidopsis thaliana* collected by Joy Bergelson's lab at the University of Chicago.

Part 1: Warm-up with principal components analysis of the RegMap data

Summary: In the first part, we will interactively compute and visualize the top principal components (PCs) in the RegMap data. During our explorations, we will develop R and SLURM scripts to automate the principal components analysis (PCA) of this data set. We will use SLURM and command-line tools to assess resource needs (time & memory).

Materials:

- `pca.R`: Script to compute and visualize the top principal components (PCs) in the RegMap data.
- `pca.sbatch`: Script for submitting PCA analysis script to SLURM job engine.

Instructions: Create two sessions on midway2: one for loading up an R environment, and a second for developing code (e.g., using emacs). In one of these sessions, create a SLURM interactive session for 2 hours on midway2. It is useful to create a screen session in case you lose your connection at any point. (Motivate use of sinteractive by running htop on login node.)

```
screen -S workshop
cd R-large-scale/code
sinteractive --partition=broadwl --time=2:00:00
module load R/3.4.1
R
getwd()
```

Note: You may need to add flag `--mem` to the `sinteractive` command to request sufficient memory for the PCA analysis. Try `--mem=12G`.

Interactively step through the code in `pca.R`. A few useful commands for inspecting the RegMap data:

```
ls()
class(regmap.geno)
dim(regmap.geno)
regmap.geno[1:10,1:10]
mode(regmap.geno)
storage.mode(regmap.geno)
format(object.size(regmap.geno),units = "GB")
```

Next, in the same session, restart R, and test the script with

```
source("pca.R")
```

While rpca is running, use `htop --user=<cnetid>` in a separate console to profile memory usage (see RES column, and sort by this column by typing "M". Another useful keystroke in htop: "p".

Now, let's try non-interactive computing in R. Submit a SLURM job using our sbatch script. While it is running, here are a few of the things we can do to monitor progress of the script:

```
squeue --user=<cnetid> | less -S
ssh midway2-xxxx
htop --user=<cnetid>
less ../output/pca_err.txt
less ../output/pca_out.txt
```

Once the job has completed, here are some things we can do to assess resource usage:

```
less ../output/pca_err.txt
less ../output/pca_out.txt
sacct --user=<cnetid> --units=G | less -S
```

Exercise: Based on the `htop` results, and the output from `sacct`, what is the minimum amount of memory needed to run the PCA analysis of the RegMap data? Do `htop` and `sacct` agree? If not, which estimates more memory usage? Please test your estimate by modifying the requested memory in the SLURM script, and re-running it.

Part 2: Implementing multithreaded computation in R for analysis of genetic adaptation to climate

Summary: In the first part, we used PCA to study structure in the *A. thaliana* genetic data. In this part of the lesson, we will study the genetic variance of the climate variables ("phenotypes"). This can be loosely interpreted as an indicator of adaptation to climate; higher genetic variance estimates indicates greater genetic adaptation.

Materials:

- `climate.R`: R ccript that estimates the genetic variance of a selected climate variable.
- `climate.sbatch`: sbatch script that submits a SLURM job on the broadwl partition, in which one of the R script parameters is specified in the command line.
- `run.all.climate.vars.R`: This R script runs the genetic variance analysis (`climate.R`) for all climate variables in the RegMap data set.
- `summarize.climate.R`: This R script summarizes the genetic variance analyses of all the climate variables.

Exercise 1: Here we will interactively explore multithreading of matrix operations (implemented in OpenBLAS) for computing the kinship matrix and the weights:

- Quit R, enter command `export OPENBLAS_NUM_THREADS=2` in the shell, then re-open R. Then run the analysis script: `source("climate.R")`. What happens? Why does the computation get slower?
- Next, quite R, and start up a new `sinteractive` session, requesting 9 CPUs with the additional flag `--cpus-per-task`: `sinteractive --partition=broadwl --time=2:00:00 --cpus-per-task=9 --mem=36G`. The multicore variant may also require more memory, so we request 36 GB to be safe. Then set `export OPENBLAS_NUM_THREADS=1`.
- How does the number of threads used in OpenBLAS affect compute time of the kinship matrix and the weights? What effect does OpenBLAS multithreading have on memory usage?

Some tips:

- Use `htop` in a separate session to examine memory usage and multithreading (look at the `CPU%` and `NLWP` columns). The `NLWP` is not included by default, and may need to be added.
- In htop, sort rows by `RES` column by typing "<" then selecting `M_RESIDENT`.
- For a more convenient way to set the number of OpenBLAS threads without having to restart your R session, use the provided R function `set.blas.num.threads`. To use this function, you will first need to build the `setblas` shared object (`.so`) library in the command-line shell, `R CMD SHLIB setblas.c`, then load the shared objects into R, `dyn.load("setblas.so")`. For example, to tell OpenBLAS to use 2 threads, type `set.blas.num.threads(2)` in R.

Exercise 2: In this exercise, we will interactively explore parallel computation of the weights.

- First set the number of OpenBLAS threads to 1 using the same command as above. Try different values for the `nc` argument in the `compute.log.weights.mclapply` call in `climate.R`. How does how does increasing the number of threads ("workers") affect compute time and memory? Use `htop` to assess this. What computational (memory and time) trade-off do you observe?
- Based on your findings, how would you suggest setting the number of OpenBLAS and mclapply threads to make most effective use of computing resources?

Exercise 3: In this exercise, we will automate the data analysis for all 48 climate variables using a combination of R scripts and SLURM scripts. This can be used as a prototype for automating R data analyses using the SLURM, which is the job engine used by the RCC cluster.

Here is an example of running this script using SLURM from the shell:

```
sbatch --job-name=climate --output=climate_out.txt \
  --error=climate_err.txt climate.sbatch aridity_fao
```

- First, run script `run.all.climate.vars.R`, which generated results for 3 climate variables: `bio1_meant`, `bio2_diur_rng` and `bio3_isotherm`.
- Next, modify the script `run.all.climate.vars.R` so that it submits SLURM jobs to analyze all 48 climate variables. Run this R script, and monitor its progress using `squeue --user=<cnetid>`.
- What is the smallest (mean) genetic variance estimate for all 48 climate variables? What is the largest? You may use script `summarize.climate.R` to generate a summary of the results.
- Suppose that you would like a member of your project team to reproduce your analysis on the RCC cluster. What instructions would you give to this person?

Exercise 4 (time permitting): In Exercise 2, we found that computing the weights in parallel using `mclapply` requires a lot more memory. In the `parallel` package, there is an alternative way to parallelize computations using the `parLapply` function. Unlike `mclapply`, this function does not rely on forking, so it is also available on Windows. `parLapply` is more difficult to use because variables must be exported manually. But an important benefit of `parLapply` is that the memory usage can be much better controlled. In this exercise, modify the `climate.R` script to use function `compute.log.weights.parlapply` (defined in `functions.R`) instead of `compute.log.weights.mclapply`, then experiment with different numbers of threads and amounts of requested memory in `climate.sbatch`. How much additional memory is required compared to single-threaded computing? You should discover that multithreaded computing using `parLapply` requires much less memory than `mclapply`.