Big data analysis with R

Myeong-Hun Jeong

CyberGIS Center for Advanced Digital and Spatial Studies
National Center for Supercomputing Applications (NCSA)
University of Illinois at Urbana-Champaign
mhjeong@illinois.edu

March, 15, 2016

Contents

- Labs:
  - Lab 1: Investigate resource pressures between serial and parallel approaches.
  - Lab 2: Parallel spatial analysis -- Parallel spatial autocorrelation analysis (Moran's I)

+

=

What to do if the computation is too big for a single desktop

- Break big computation with multiple job submission
  - RHadoop is a collection of R packages that enables users to process and analyze big data with Hadoop.
  - Rmr2 and Rhdfs packages.
- Implement code using parallel packages
  - Run R with a couple of parallel packages.

Lab 1: Running parallel R

- Objective:
  - Learn how to run parallel R
- Successful outcome:
  - Investigate resource pressures between serial and parallel approaches.
- Before you begin:
  - Environmental variable setting
    - vi ./bashrc -- open bashrc file
    - module load R -- put this line and exit the file after saving.
  - Module load
    - module load binutils/2.25 openblas/0.2.14 R/3.2.1-openmpi geos
  - Install parallel R packages

Running parallel R

- Perform the following steps:
- Step 1: Install packages for this course.
  - Start R
  - $ R
  - Install packages for this course.
    - > install.packages(“snow”)
    - > install.packages(“Rmpi”)
    - > install.packages(“foreach”)
    - > install.packages(“doSNOW”)
    - > install.packages(“parallel”)
    - > install.packages(“boot”)
    - > install.packages(“maptools”)
    - > install.packages(“spdep”)
    - You select a CRAN mirror for installing packages (e.g., USA(IN)).
    - Or > install.packages(c("snow", "Rmpi", "foreach", "doSNOW", "parallel", "boot", "maptools", "spdep”))

Running parallel R

- Step 2: Define a simple R function.

```
myProc <- function(size=10000000) {
  #Load a large vector
  vec <- rnorm(size)
   #Now sleep on it
  Sys.sleep(2)
  #Now sum the vec values
  total  <- 0
  for(v in vec) {
    total  <- total + v
  }
}
```

Running parallel R

- Step 3: To run myProc() 10 times using apply.

```
ptm <- proc.time()
#sapply converts results into a vector or array of appropriate size
result <- sapply(1:10, function(i) myProc())
proc.time() – ptm
=>  user  system elapsed
 14.929   0.232  35.179
```

Running parallel R

- Step 3: To run myProc() 10 times using a parallel package.

```
require(parallel)
ptm <- proc.time()
result <- mclapply(1:10, function(i) myProc(), mc.cores=10)
proc.time() – ptm
=>  user  system elapsed
  0.012   0.025  4.616
```

Running parallel R

- Step 4: To run myProc() 10 times using a SNOW package.

```
require(snow)
#Create a cluster object
hostnames <- rep('localhost', 10)
cluster <- makeSOCKcluster(hostnames)
#Assigns the values on the master of the variables named in list to variables of the same names in the global environments of each node.
clusterExport(cluster, list('myProc'))
ptm <- proc.time()
#Call function on the first cluster node with arguments seq[[1]] and ..., on the second node with seq[[2]] and ..., and so on. If the length of seq is greater than the number of nodes in the cluster then cluster nodes are recycled.
result <- clusterApply(cluster, 1:10, function(i) myProc())
proc.time() - ptm
=>  user  system elapsed
  0.006   0.001   4.053
#To shut down the cluster
stopCluster(cluster)
```

Running parallel R

- Step 5: To run myProc() 10 times using foreach + SNOW packages.

```
require(foreach)
require(doSNOW)
hostnames <- rep('localhost', 10)
cluster <- makeSOCKcluster(hostnames)
#register the SNOW parallel backend with the foreach package.
registerDoSNOW(cluster)
ptm <- proc.time()
#'c' is useful for concatenating the results into a vector.
result <- foreach(i=1:10, .combine=c) %dopar% {
myProc()
}
proc.time() - ptm
⇒ user  system elapsed
0.023   0.010   4.983
stopCluster(cluster)
```

Running parallel R

- Step 6: Bootstrap calculations.
  - Serial implementation.

```
random.data <- matrix(rnorm(1000000), ncol = 1000)
bmed <- function(d, n) median(d[n])
library(boot)
sapply(1:100, function(n) {sd(boot(random.data[, n], bmed, R = 10000)$t)})
=> user  system elapsed
103.305   0.828 224.822
```

  - Parallel implementation

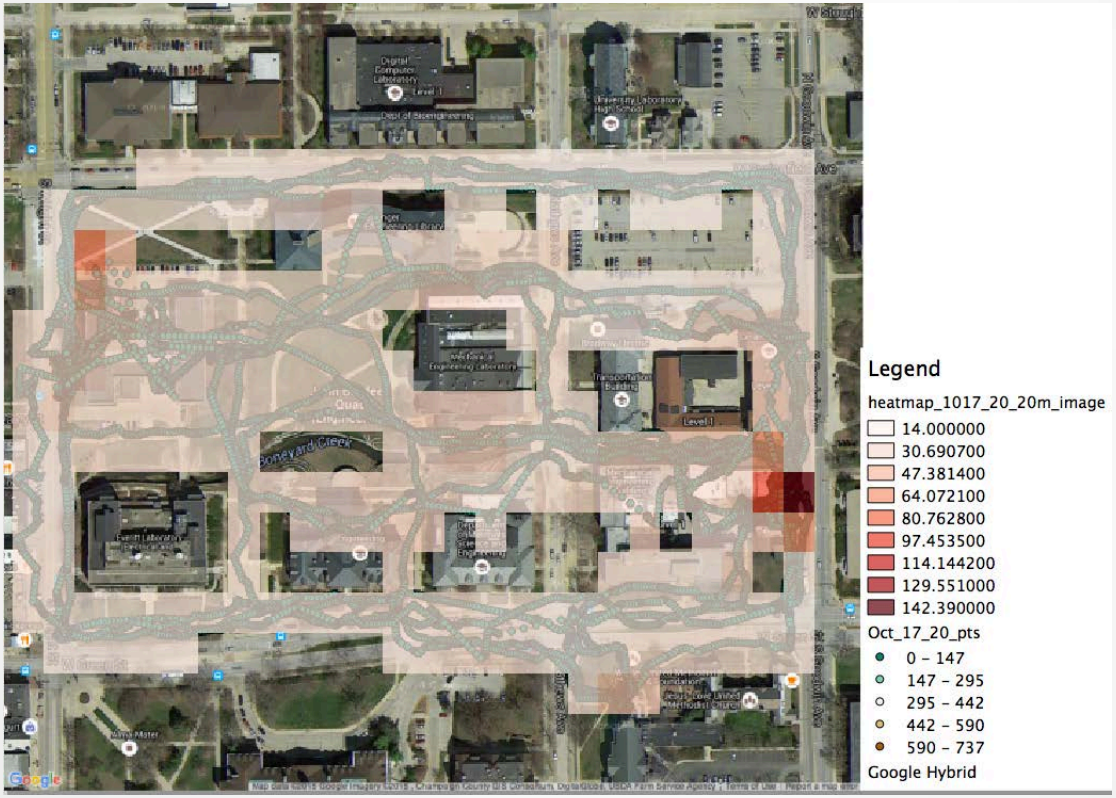
```
library(doSNOW)
cluster = makeCluster(10, type = "SOCK")
clusterExport(cluster, c("random.data", "bmed"))
results = foreach(n = 1:100, .combine = c) %dopar% {
    library(boot); sd(boot(random.data[, n], bmed, R = 10000)$t)
}
stopCluster(cluster)
 ⇒ user  system elapsed
0.340   0.069  21.037
```

Lab 2: Parallel spatial analysis

- Objective:
  - Learn how to calculate parallel spatial autocorrelation (Moran's I)
- Location of files:
  - /gpfs_scratch/geog479/lab9
- Successful outcome:
  - Your will calculate a spatial autocorrelation index, based on serial and parallel approaches.
- Before you begin:
  - SSH into the ROGER system.
  - Make a data directory and copy shape files into the data directory
    - cp /gpfs_scratch/geog479/lab9/*.* dataset

Lab 2: Radiation spatial autocorrelation

- Radiation measurements on UIUC campus.



- Acknowledgement of data use
  - Prof. Clair, the department of NPRE shares her group data for the CyberGIS workshop.

Radiation spatial autocorrelation

- Step 1: Moran's I serial implementation

```
require(maptools)
require(spdep)
#an absolute filepath representing the current working directory of the R proces
getwd()
#set the working directory
setwd("~/")

#Get shapefile header information in the radiation data
getinfo.shape("dataset/Oct_17_20_proj.shp")
#Reads data from a points shapefile
radiation<-readShapePoints ("dataset/Oct_17_20_proj.shp")

#Retrieve spatial coordinates from a Spatial object
coords<-coordinates(radiation)
IDs<-row.names(as(radiation, "data.frame"))

#Neighbourhood contiguity by distance
radiation_nei<-dnearneigh(coords, d1=0, d2=20, row.names=IDs)

#Spatial weights for neighbours lists
radiation_nbq_wb<-nb2listw(radiation_nei, style="W")

#Moran's I test for spatial autocorrelation
moran.test(radiation$field_5, listw=radiation_nbq_wb)
```

Radiation spatial autocorrelation

- Step 1: Moran's I serial implementation

```
#Moran's I test for spatial autocorrelation
moran.test(radiation$field_5, listw=radiation_nbq_wb)
```

```
Moran I statistic standard deviate = 335.63, p-value < 2.2e-16
alternative hypothesis: greater
sample estimates:
Moran I statistic       Expectation          Variance
     4.310875e-01     -1.076079e-04      1.650584e-06
```

Radiation spatial autocorrelation

- Step 2: Moran's I parallel implementation

```
gamma <- radiation$field_5
listw <- radiation_nbq_wb
nsim <- 999
require(foreach)
require(doSNOW)
n <- length(listw$neighbours)
S0 <- Szero(listw)
cluster = makeCluster(10, type = "SOCK")
registerDoSNOW(cluster)
clusterExport(cluster, c("gamma", "listw","n","S0"))
results = foreach(n = 1:nsim, .combine = c) %dopar% {
 library(spdep); moran(sample(gamma), listw, n, S0,zero.policy=NULL)$I
}
```

Radiation spatial autocorrelation

- Step 2: Parallel Moran's I calculation

```
paMoran <- function(res, x, listw, nsim,zero.policy=NULL,alternative="greater") {
  n <- length(listw$neighbours)
  S0 <- Szero(listw)

  res[nsim+1] <- moran(x, listw, n, S0, zero.policy)$I
  rankres <- rank(res)
  xrank <- rankres[length(res)]
        diff <- nsim - xrank
        diff <- ifelse(diff > 0, diff, 0)

  if (alternative == "less")
        pval <- punif((diff + 1)/(nsim + 1), lower.tail=FALSE)
        else if (alternative == "greater")
        pval <- punif((diff + 1)/(nsim + 1))

        statistic <- res[nsim+1]
        names(statistic) <- "statistic"
        parameter <- xrank
        names(parameter) <- "observed rank"
        method <- "Parallel Monte-Carlo simulation of Moran's I"
        lres <- list(statistic=statistic, parameter=parameter,
           p.value=pval, alternative=alternative, method=method,res=res)
        lres
}
```

Radiation spatial autocorrelation

- Step 2: Parallel Moran's I calculation

```
mtest <- paMoran(results,gamma,listw,nsim)
mtest$method      => Parallel Monte-Carlo simulation of Moran's I
mtest$statistic   => 0.43109
mtest$parameter   => observed rank 1000
mtest$p.value     =>  0.001
```