Parallel Computing in R

Xie Chao & Tan Tin Wee

Why Parallel Computing?

- Data
  - Large amount of data
  - Large amount of computation
- Computer
  - Multicore processor
  - Many multicore processors
- Many data analysis procedures are embarrassingly parallel

Different Scales

- Multi-core computer
- Simple network of several computers
- Linux cluster with many computers
- Grid or Cloud

Example

- geneData: 500 genes x 26 samples
- Correlation between every pair of genes

Demo

Your Turn: The Data

```
R                                   # start R

> library(Biobase)                  # load package
> data(geneData)                    # load data
> ?geneData                         # details about the data
> dim(geneData)                     # dimensions of geneData
> head(geneData)                    # first few genes

> ?cor
> cor(geneData[12, ], geneData[13, ])
                                    # correlation between gene 12 & 13
> ?pair
> pair <- combn(1:nrow(geneData), 2, simplify = F)
                                    # generate pairs
> length(pair)                      # same as choose(500, 2)
> head(pair)
> tail(pair)
```

```
> mylist <- list(1:5, 1:10)

> mylist[[1]]
[1] 1 2 3 4 5

> mylist
[[1]]
[1] 1 2 3 4 5

[[2]]
[1]  1  2  3  4  5  6  7  8  9 10

> myfun <- function(x) mean(x)

> myfun(mylist[[1]])
[1] 3
> myfun(mylist[[2]])
[1] 5.5

> lapply(mylist, myfun)
[[1]]
[1] 3

[[2]]
[1] 5.5
```

lapply

- lapply(list, FUN)
- apply the function FUN to each element of the list

Serial Computing

```
> geneCor <- function(x, gene = geneData) {
+     cor(gene[x[1], ], gene[x[2], ])          # declare a function
+ }
> geneCor(c(12, 13))                           # cor between 12 & 13
> out <- lapply(pair[1:3], geneCor)            # first 3 pairs
> out

( start another console, run “htop”, monitor CPU usage )

> system.time(out <- lapply(pair, geneCor))    # for all pairs
( record elapsed time )
> head(out)

( optional: visualize correlation network )
> corm <- cbind(do.call(rbind, pair), unlist(out))
> corm <- corm[abs(corm[,3]) >= 0.86, ]     # remove low cor pairs
> library(network); library(sna)
> net <- network(corm, directed = F)        # the network
> cd <- component.dist(net)                 # component analysis
> delete.vertices(net, which(cd$csize[cd$membership] == 1))
                          # delete genes not connected with others
> plot(net)
```

Multicore Computing

- multicore package by Simon Urbanek
- Linux and Mac OS
- Benchmark
  - `serial:     14.44 seconds`
  - `quad-core:   3.89 seconds`

Multicore Computing

```
( in another console, monitor CPU usage )

> library(multicore)
> system.time(out <- mclapply(pair, geneCor))

( compare elapsed time with lapply )

> head(out)
```

Something Heavier

```
fakeData <- cbind(geneData, geneData, geneData, geneData)

library(boot)
geneCor2 <- function(x, gene = fakeData) {
    mydata <- cbind(gene[x[1], ], gene[x[2], ])
    mycor <- function(x, i) cor(x[i,1], x[i,2])
    boot.out <- boot(mydata, mycor, 1000)
    boot.ci(boot.out, type = 'bca')$bca[4:5]
}

geneCor2(c(12, 13))        # for gene 12 and 13

system.time(out <- lapply(pair[1:10], geneCor2))
( first 10 pairs, )
( how many hours do you need for all pairs? )

pair2 <- sample(pair, 300)   # let's do 300 pairs
system.time(out <- lapply(pair2, geneCor2))    # serial
system.time(out <- mclapply(pair2, geneCor2))  # multicore
( record elapsed time )
```

larger data

heavier computing: 95% confidence interval by 1000 bootstrapping

Multiple Machines

- Simple Network Of Workstations (snow)
- snow package by Luke Tierney, A. J. Rossini, Na Li, H. Sevcikova
- Requirement:
  - SSH access to worker nodes
  - R package snow installed on all nodes

On the other two computers,
find out the IP address

```
nusnet-75-6 ~ # ifconfig
eth1      Link encap:Ethernet  HWaddr 00:21:97:a2:13:69
          inet addr:137.132.75.6  Bcast:137.132.75.255  Ma
          UP BROADCAST NOTRAILERS RUNNING MULTICAST  MTU:1
          RX packets:216 errors:0 dropped:0 overruns:0 fra
          TX packets:8 errors:0 dropped:0 overruns:0 carri
          collisions:0 txqueuelen:100
          RX bytes:24859 (24.2 KiB)  TX bytes:1579 (1.5 Ki
          Memory:f9fc0000-f9fe0000

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
```

(Then, on your main computer)

```
$ ssh-keygen          # press Enter, Enter, ...
$ ssh-copy-id -i /root/.ssh/id_rsa.pub 137.132.xxx.xxx
                      # type “yes”, then password: bioslax
$ ssh-copy-id -i /root/.ssh/id_rsa.pub 137.132.yyy.yyy

( password-less login )
$ ssh 137.132.xxx.xxx
( successful ? )
```

Public Key Authentication

let it snow

```
> library(snow)
> hosts <- c(“localhost”, “137.132.x.x”, “137.132.y.y”)
> cl <- makeCluster(hosts)

> clusterCall(cl, date)
                # call date() on each worker
                # synchronized ?
> clusterCall(cl, Sys.info)
                # call Sys.info() on each worker

> stopCluster(cl)
```

back to work

```
> cl <- makeCluster(rep(hosts, 2))
                # two worker per machine

> clusterExport(cl, “fakeData”)
                      # export data to workers
> clusterEvalQ(cl, library(boot))
                      # load package on workers

> system.time(out <- clusterApply(cl, pair2, geneCor2))
( is it faster ? )

> stopCluster(cl)
```

Cluster

- multicore and multi-machine snow are too slow for you
- Large cluster with MPI support
  - NUS HPC cluster
  - 32 processor-cores per job

1 Master + 31 Workers

```
( submit job through MPI environment )

> cl <- makeCluster()
> clusterExport(cl, 'fakeData')
> clusterEvalQ(cl, library(boot))
> system.time(out2 <- clusterApply(cl, pair2, geneCor2))
   user  system elapsed
  2.373   0.001   2.374
```

```
single-core: 66.516
quad-core:   19.245
```

Problem

- How to maintain your code for
  - single core
  - multicore
  - snow
  - cluster with MPI support

"one ring to rule them all"

- The foreach framework
  - by REvolution Computing
- Unified interface for packages:
  - multicore => doMC
  - snow => doSNOW
  - Rmpi => doMPI

The Base Code

```
#!/usr/bin/env Rscript
library(foreach)

data(geneData, package = 'Biobase')
pair <- combn(1:nrow(geneData), 2, simplify = F)
fakeData <- cbind(geneData, geneData, geneData, geneData)
pair2 <- sample(pair, 300)

print(system.time(
    out <- foreach(p = pair2, .packages = 'boot', .combine = 'rbind') %dopar%
    {
        mydata <- cbind(fakeData[p[1],], fakeData[p[2], ])
        mycor <- function(x, i) cor(x[i,1], x[i,2])
        boot.out <- boot(mydata, mycor, 1000)
        ci <- boot.ci(boot.out, type = 'bca')$bca[4:5]
        c(p, ci)
    }
))
print(head(out))
```

Save as “base.r”
chmod 700 base.r

Run The Base Code

```
$ ./base.r

   user  system elapsed
 65.920   0.440  66.355

          [,1] [,2]       [,3]       [,4]
result.1  288  422 -0.3357138 -0.1266808
result.2  243  303 -0.1529784  0.1017721
result.3  234  386 -0.1561772  0.2161708
result.4    5  183  0.2505966  0.4794503
result.5  275  440  0.2420215  0.5738407
result.6   62  457  0.1558987  0.4712927
```

doMC

```
#!/usr/bin/env Rscript

library(doMC)
registerDoMC()

source("base.r")  # or paste content
```

Save as “domc.r”
`chmod 700 domc.r`

run domc.r

```
$ ./domc.r

   user  system elapsed
 56.410  56.290  19.114

          [,1] [,2]       [,3]        [,4]
result.1  164  172 -0.2668234  0.14122971
result.2   31   67  0.6607894  0.78171239
result.3   62  282 -0.3225335 -0.03370322
result.4   37  293  0.1073143  0.31629694
result.5   38  271 -0.1041704  0.28953027
result.6  211  287 -0.3495002  0.00608487
```

doSNOW

```
#!/usr/bin/env Rscript

library(doSNOW)
hosts <- c(
   'localhost', 'localhost', 'localhost', 'localhost',
   'variome', 'variome',
   'lams', 'lams',
   'bug'
)
cl <- makeCluster(hosts)
registerDoSNOW(cl)

source("base.r")    # or paste content

stopCluster(cl)
```

change to your own hosts

Save as “dosnow.r”
`chmod 700 dosnow.r`

run dosnow.r

```
$ ./dosnow.r

   user  system elapsed
  0.130   0.020  12.335

          [,1] [,2]       [,3]        [,4]
result.1   217  332  0.1224703  0.44764756
result.2    19  179 -0.6041616 -0.36109986
result.3   192  426  0.4824728  0.68836742
result.4    46  228 -0.2883579 -0.01718258
result.5   407  426  0.1998271  0.45557312
result.6   165  386 -0.6302590 -0.15638907
```

doMPI

```
#!/usr/bin/env Rscript


library(doMPI)
cl <- startMPIcluster()
registerDoMPI(cl)

source("base.r")   # or paste content

closeCluster(cl)
```

Save as “dompi.r”
chmod 700 dompi.r

run dompi.r

```
$ bsub -q atlas5_parallel -a openmpi -n 32
mpirun.lsf ./dompi.r

   user  system elapsed
  2.742   0.243   3.015

          [,1] [,2]        [,3]       [,4]
result.1   202  231  0.003461775  0.3608233
result.2   174  372  0.009071919  0.3583579
result.3   118  186  0.601903906  0.7885814
result.4   186  362 -0.336281370 -0.1353606
result.5   270  485 -0.491912888 -0.1121098
result.6   168  413 -0.147680243  0.1079481
```

Summary

- Parallel computing in R is easy
  - for embarrassingly parallel problems
- Different scales of parallel computing
  - multicore
  - snow
  - Rmpi
  - foreach + doMC / doSNOW / doMPI

More Information

http://cran.r-project.org/web/views/HighPerformanceComputing.html