

Parallel Computing in R

Xie Chao

xiechao@nus.edu.sg

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

Different Scales in R

- Multi-core computer
 - multicore
- Simple network of several computers
 - snow
- Linux cluster with many computers
 - RMpi, snow

(<http://trg.apbionet.org/euasiagrid/docs/parallelR.notes.pdf>)

Example

- Find palindrome regions in *C. elegans* chromosomes
- “Super” *C. elegans* with 10X larger genome

Install

```
# multicore package
install.packages("multicore")

# example data
source("http://bioconductor.org/biocLite.R")
biocLite("BSgenome.Celegans.UCSC.ce2")
```

Requirements for this workshop:
Linux or Mac,
(multicore) laptop or server,
no Windows

```
# load C elegans genome
library(BSgenome.Celegans.UCSC.ce2)

# overview of the genome
Celegans

# chromosomes in celegans
seqnames(Celegans)

# creating the super bug, with 10X larger genome
chrs <- rep(seqnames(Celegans), 10)
chrs
elegan <- getSeq(Celegans, chrs, as.character = F)
elegan

# convert to a list of chromosomes
elegan <- as.list(elegan)
elegan

# individual chromosomes
elegan[[1]]
elegan[[2]]
elegan[[10]]
```

Create Super Celegans

Find Palindromes

```
findPalindromes(elegan[[1]])  
findPalindromes(elegan[[2]])  
findPalindromes(elegan[[3]])  
# a “for” loop ???
```

the R way: lapply

```
> 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

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

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

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

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

serial computing

```
> system.time(pld2 <- findPalindromes(elegan[[2]]))
  user  system elapsed
0.960   0.010   0.971
> pld2

> system.time(pld20 <- findPalindromes(elegan[[20]]))
  user  system elapsed
1.130   0.030   1.153
> pld20

> system.time(plds <- lapply(elegan, findPalindromes))
  user  system elapsed
86.290   0.190  86.696
> plds[[2]]
> plds[[20]]
```

```
1 [|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||100.0%]
2 [                                                                0.0%]
3 [                                                                0.0%]
4 [                                                                0.0%]
5 [|||                                                             1.9%]
6 [                                                                0.0%]
7 [                                                                0.0%]
8 [                                                                0.0%]
Mem[||||| ||||||||||||||||||||||||||||||||||||||||||||||2003/24161MB]
Swp[|                                                                155/19459MB]

Tasks: 229 total, 2 running
Load average: 0.22 0.21 0.16
Uptime: 7 days, 23:11:04
```

PID	USER	PRI	NI	VIRT	RES	SHR	S	CPU%	MEM%	TIME+	Command
25638	xiechao	20	0	949M	766M	5456	R	106.	3.2	1:56.96	/usr/lib64/R/bin/exec/R
26089	xiechao	20	0	19892	1764	1104	R	2.0	0.0	0:00.34	htop
1	root	20	0	23836	1620	1028	S	0.0	0.0	0:03.04	/sbin/init
483	root	20	0	17036	768	564	S	0.0	0.0	0:00.08	upstart-udev-bridge --daemon
488	root	16	-4	17028	876	384	S	0.0	0.0	0:00.06	udevd --daemon

But only one core is utilized

Multicore Computing

- multicore package by Simon Urbanek
- Linux and Mac OS

multicore computing

```
> system.time(plds <- lapply(elegan, findPalindromes))
  user  system elapsed
86.290   0.190  86.696
> plds[[2]]
> plds[[20]]

> library(multicore)
> system.time(pldmc <- mclapply(elegan, findPalindromes))
  user  system elapsed
85.160   4.140  20.902
> pldmc[[2]]
> pldmc[[20]]
```

in case you have too many processor cores for your RAM:
option(cores = 2)

```
1 [|||||||||||||||||||||||||||||||||||||||||||||||||||||||||100.0%]
2 [|||||||||||||||||||||||||||||||||||||||||||||||||||||||||100.0%]
3 [|||||||||||||||||||||||||||||||||||||||||||||||||||||||||100.0%]
4 [|||||||||||||||||||||||||||||||||||||||||||||||||||||||||100.0%]
5 [|||||||||||||||||||||||||||||||||||||||||||||||||||||||||100.0%]
6 [|||||||||||||||||||||||||||||||||||||||||||||||||||||||||100.0%]
7 [|||||||||||||||||||||||||||||||||||||||||||||||||||||||||100.0%]
8 [|||||||||||||||||||||||||||||||||||||||||||||||||||||||||100.0%]
Mem[|||||||||||||||||||||||||||||||||||||||||||||||||||||6340/24161MB]
Swp[|155/19459MB]
```

Tasks: 235 total, 8 running
Load average: 2.05 0.71 0.33
Uptime: 7 days, 23:13:27

PID	USER	PRI	NI	VIRT	RES	SHR	S	CPU%	MEM%	TIME+	Command
26097	xiechao	20	0	1113M	924M	1328	R	100.	3.8	0:14.60	/usr/lib64/R/bin/exec/R
26098	xiechao	20	0	1113M	924M	1328	R	100.	3.8	0:14.58	/usr/lib64/R/bin/exec/R
26094	xiechao	20	0	1113M	924M	1328	R	100.	3.8	0:14.67	/usr/lib64/R/bin/exec/R
26095	xiechao	20	0	1113M	924M	1328	R	100.	3.8	0:14.66	/usr/lib64/R/bin/exec/R
26096	xiechao	20	0	1113M	924M	1328	R	100.	3.8	0:14.49	/usr/lib64/R/bin/exec/R
26100	xiechao	20	0	1113M	924M	1328	R	98.0	3.8	0:14.26	/usr/lib64/R/bin/exec/R
26093	xiechao	20	0	1113M	924M	1328	R	97.0	3.8	0:14.40	/usr/lib64/R/bin/exec/R
25638	xiechao	20	0	1238M	1053M	5472	S	19.0	4.4	3:09.34	/usr/lib64/R/bin/exec/R
26089	xiechao	20	0	19892	1768	1104	R	1.0	0.0	0:02.20	htop
1776	xiechao	20	0	338M	17420	4600	S	0.0	0.1	1:28.65	/usr/bin/cli /usr/lib/gnome-do/Do.exe
1	root	20	0	23836	1620	1028	S	0.0	0.0	0:03.04	/sbin/init

All cores are being utilized

(note: I have only 4 cores, not 8)

one more example

```
> system.time(di.a <- lapply(elegan, dinucleotideFrequency))
  user  system elapsed
 5.660   0.000   5.684
> di.a[[2]]
> di.a[[20]]

> library(multicore)
> system.time(di.b <- mclapply(elegan, dinucleotideFrequency))
  user  system elapsed
65.120   2.610   1.824
> di.b[[2]]
> di.b[[20]]
```


with non-default parameters

```
> system.time(plds <- lapply(elegan, findPalindromes,  
min.armlength = 6, max.looplength = 5))  
  user  system elapsed  
67.070  0.300  67.541  
> plds[[2]]  
> plds[[20]]  
  
> library(multicore)  
> system.time(pldmc <- mclapply(elegan, findPalindromes,  
min.armlength = 6, max.looplength = 5))  
  user  system elapsed  
132.930  6.570  16.283  
> pldmc[[2]]  
> pldmc[[20]]
```

Longest palindrome region?

```
# define our own function
> longest.palindrome <- function(s) {
  pld <- findPalindromes(s);
  pld.width <- width(pld);
  longest <- which.max(pld.width);
  pld[longest]
}

> system.time(pldmc <- mclapply(elegan, longest.palindrome))
  user  system elapsed
135.000   7.360  19.827

> pldmc
```

```

> pldmc
[[1]]
  Views on a 15080483-letter DNAString subject
subject: GCCTAAGCCTAAGCCTAAGCCTAAGCCTAAGCCT...AGGCTTAGGCTTAGGCTTAGGTTTAGGCTTAGGC
views:
      start      end width
[1] 9391035 9391223   189 [GTGTGTGTGTGTGTGTGTGTGTGTGTGTG...TGTGTGTGTGTGTGTGTGTGTGTGTGTG]

[[2]]
  Views on a 15279308-letter DNAString subject
subject: CCTAAGCCTAAGCCTAAGCCTAAGCCTAAGCCTA...TAGGCTGAGACTTAGGCTTAGGCTTAGGCTTAGT
views:
      start      end width
[1] 503958 504632   675 [ATATATATATATATATATATATATATATAT...ATATATATATATATATATATATATA]

[[3]]
  Views on a 13783313-letter DNAString subject
subject: CCTAAGCCTAAGCCTAAGCCTAAGCCTAAGCCTA...TAGGCTTAGGCTTAGGCTTAGGCTTAGGCTTAGG
views:
      start      end width
[1] 3334130 3334302   173 [AAAAATAAAAATAAAAATAAAAATA...TAAAATAAAAATAAAAATAAAA]

[[4]]
  Views on a 17493791-letter DNAString subject
subject: CCTAAGCCTAAGCCTAAGCCTAAGCCTAAGCCTA...TAGGCTTAGGCTTAGGCTTAGGCTTAGGCTTAGG
views:

```

```
> data.frame(  
  chr = chrs,  
  start = sapply(pldmc, start),  
  end = sapply(pldmc, end),  
  width = sapply(pldmc, width)  
)
```

	chr	start	end	width
1	chrI	9391035	9391223	189
2	chrII	503958	504632	675
3	chrIII	3334130	3334302	173
4	chrIV	13116111	13116283	173
5	chrV	6946625	6947010	386
6	chrX	2011047	2011239	193
7	chrM	13362	13400	39
8	chrI	9391035	9391223	189
9	chrII	503958	504632	675
10	chrIII	3334130	3334302	173
11	chrIV	13116111	13116283	173
12	chrV	6946625	6947010	386
13	chrX	2011047	2011239	193

Beyond Multicore...

Example

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

Data & Compute

```
# the data
```

```
library(Biobase)          # load package
data(geneData)           # load data
fakeData <- cbind(geneData, geneData, geneData, geneData)
```

```
# the compute
```

```
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]
}
pair <- combn(1:nrow(geneData), 2, simplify = F)
pair2 <- sample(pair, 300) # let's do 300 pairs
```

Serial vs Multicore

```
# serial
```

```
> system.time(out <- lapply(pair2, geneCor2))  
  user  system elapsed  
71.450   0.290  71.997
```

```
# multicore
```

```
> system.time(out <- mclapply(pair2, geneCor2))  
  user  system elapsed  
108.650   1.060  15.969
```

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
 - Enable password-less login (public key)
 - R package snow installed on all nodes

parallel computing with snow

```
> library(snow)

> hosts <- c("localhost", "137.132.x.x", "137.132.y.y")
> 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))

> 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

| Master + 3 | 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: 72
quad-core: 16

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 Analytics
 - open source
- Unified interface for packages:
 - multicore => doMC
 - snow => doSNOW
 - Rmpi => doMPI

The Serial 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

Multicore with 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

snow with 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

MPI cluster with 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](http://cran.r-project.org/web/views/HighPerformanceComputing.html)