Parallel Computing with R

Le Yan

HPC @ LSU

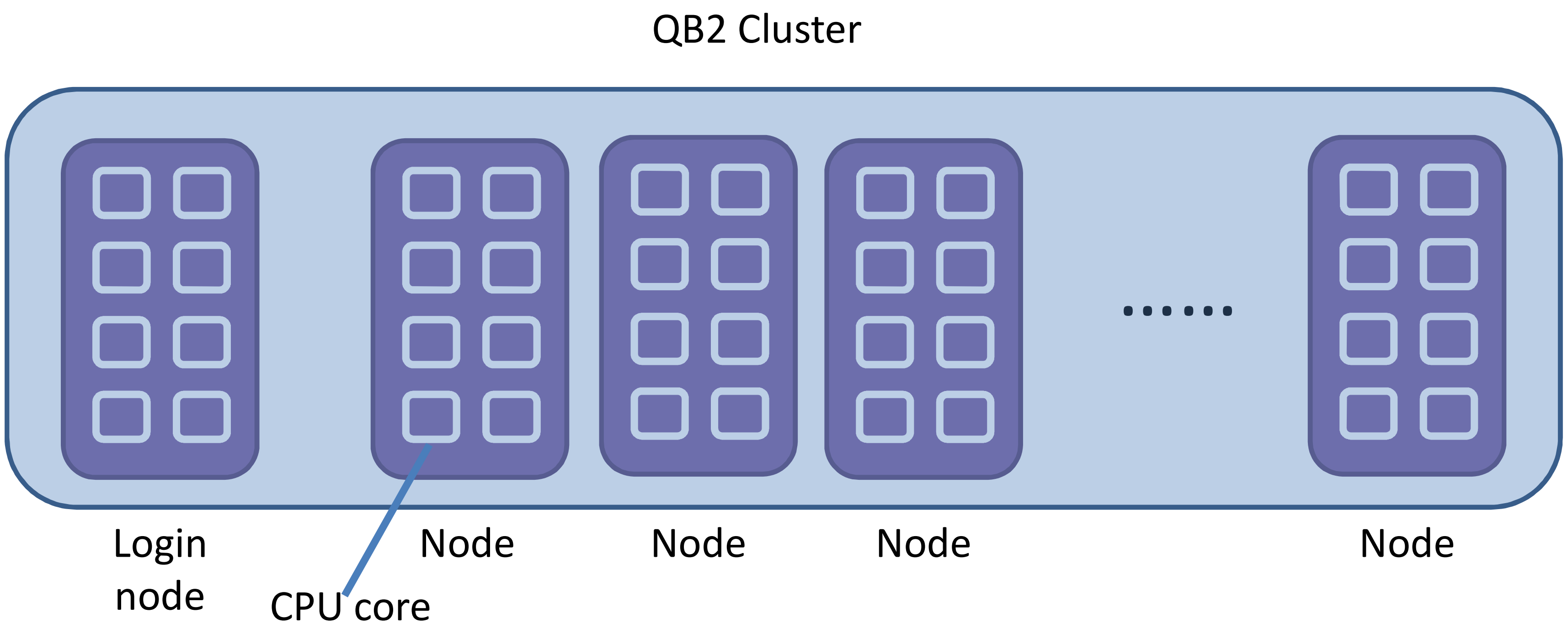
Outline

- Parallel computing primers
- Parallel computing with R
  - Implicit parallelism
  - Explicit parallelism
- R with GPU

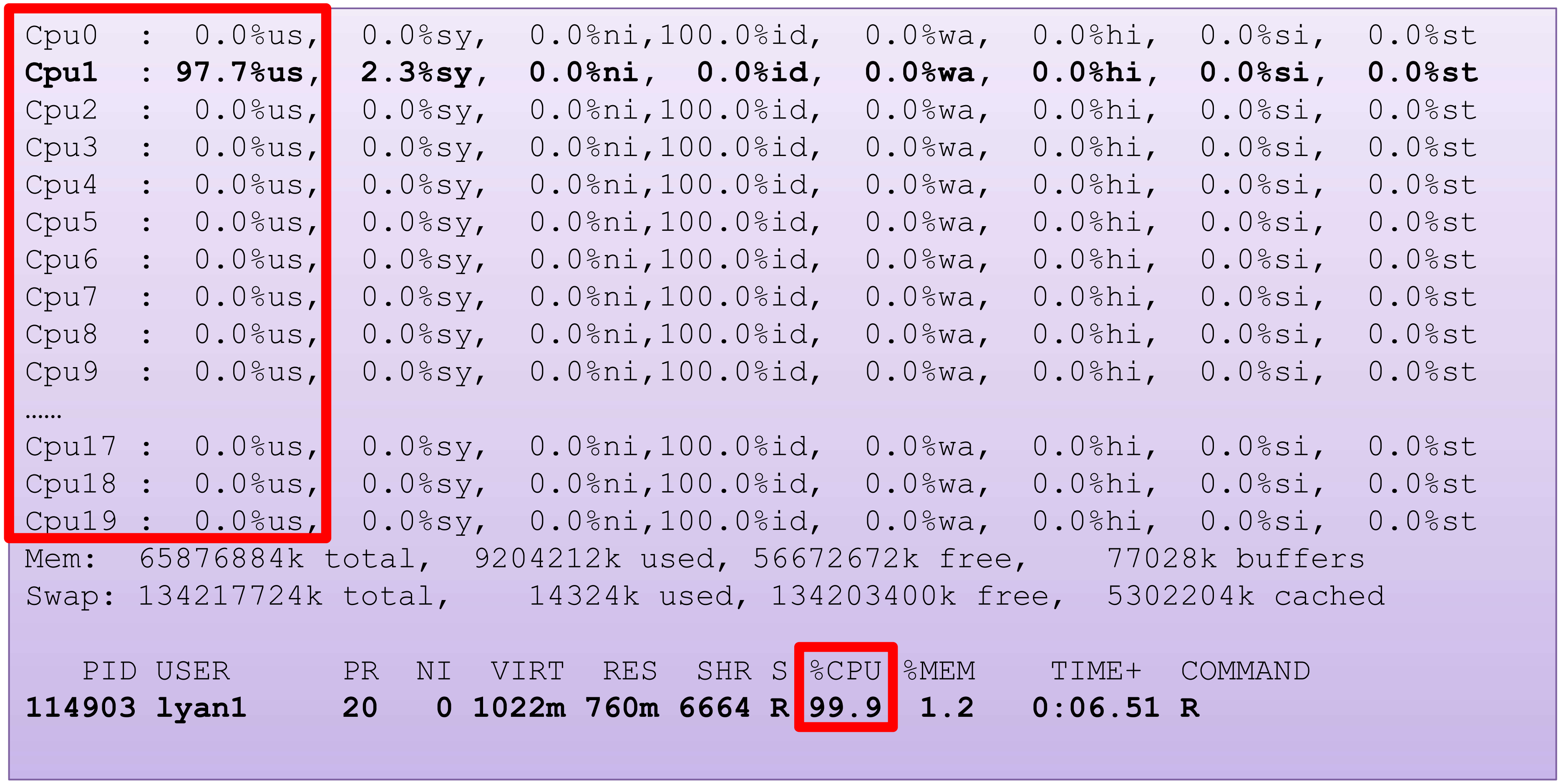
R Is Not Parallel

- Modern computers are equipped with more than one CPU core
  - Your laptop may have 4 or 8 or more
  - HPC clusters may have millions
- R is single-threaded
  - Regardless how many cores are available, R can only use one of them

Cluster Architecture



Cluster = multiple nodes (servers) x multiple cores per node


```
Cpu0   :   0.0%us,  0.0%sy,  0.0%ni,100.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu1   : 97.7%us,  2.3%sy,  0.0%ni,  0.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu2   :   0.0%us,  0.0%sy,  0.0%ni,100.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu3   :   0.0%us,  0.0%sy,  0.0%ni,100.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu4   :   0.0%us,  0.0%sy,  0.0%ni,100.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu5   :   0.0%us,  0.0%sy,  0.0%ni,100.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu6   :   0.0%us,  0.0%sy,  0.0%ni,100.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu7   :   0.0%us,  0.0%sy,  0.0%ni,100.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu8   :   0.0%us,  0.0%sy,  0.0%ni,100.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu9   :   0.0%us,  0.0%sy,  0.0%ni,100.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
……
Cpu17  :   0.0%us,  0.0%sy,  0.0%ni,100.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu18  :   0.0%us,  0.0%sy,  0.0%ni,100.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu19  :   0.0%us,  0.0%sy,  0.0%ni,100.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:   65876884k total,  9204212k used, 56672672k free,    77028k buffers
Swap: 134217724k total,    14324k used, 134203400k free,  5302204k cached

   PID USER       PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
114903 lyan1      20   0 1022m 760m 6664 R 99.9  1.2   0:06.51 R
```

R running on one node of the QB2 cluster:
20 cores total, 1 busy, 19 idle

Why Parallel Computing

- Speed
  - Running with more cores **may** speed up the time to reach solution
- Problem size
  - Running with more nodes **may** allow you to use more memory than that available on a single node

How to Achieve Parallelization

- Set up a group of workers
  - For sake of simplicity we will use worker/process/thread interchangeably
- Divide the workload into chunks
- Assign one chunk or a number of chunks to each worker

Caveats of Parallel Computing

- Using more workers does not always make your program run faster
- **Efficiency** of parallel programs
  - Defined as speedup divided by number of workers
    - Example: 4 workers, 3x speedup, efficiency = 75%; 8 workers, 4x speedup, efficiency = 50%
  - Usually decrease with increasing number of workers

Is Parallel Computing for You?

- Before parallelizing your R code, need answers to these questions:
  - Does your code run slow?
    - If no, then do not bother, e.g. it is not wise to spend weeks on parallelizing a program that finished in 30 seconds;
  - Is it parallelizable?
    - If no, then do not bother, e.g. not much we can do in R if the target R function is written in C or Fortran;
- First step in parallelization: performance analysis
  - Purpose: locate the "hotspot" first
  - Two most frequent used methods in R
    - `system.time()`
    - `Rprof()` and `summaryRprof()`

System.time()

```
## Output from system.time() function
## User: time spent in user-mode
## System: time spent in kernel (I/O etc.)
## Elapsed: wall clock time

## Usage: system.time(<code segment>)

> system.time({
+ A <- matrix(rnorm(10000*10000),10000,10000)
+ Ainv <- solve(A)
+ })
   user  system elapsed
156.582   0.948  16.325
```

Rprof() and summaryRprof()

```
## Usage:
## Profile a code segment: Rprof(); <code segment>; Rprof(NULL)
## Print profiling results: summaryRprof()

> Rprof()
> A <- matrix(rnorm(10000*10000),10000,10000)
> Ainv <- solve(A)
> Rprof(NULL)
> summaryRprof()
$by.self
                self.time self.pct total.time total.pct
"solve.default"    149.10    94.91     149.34     95.06
".External"          6.72     4.28       6.72      4.28
"matrix"             1.04     0.66       7.76      4.94
"diag"               0.24     0.15       0.24      0.15

$by.total
                total.time total.pct self.time self.pct
"solve.default"     149.34     95.06    149.10    94.91
"solve"             149.34     95.06      0.00     0.00
"matrix"              7.76      4.94      1.04     0.66
".External"           6.72      4.28      6.72     4.28
"rnorm"               6.72      4.28      0.00     0.00
"diag"                0.24      0.15      0.24     0.15
```

Outline

- Parallel computing primers
- Parallel computing with R
  - Implicit parallelism
  - Explicit parallelism
- R with GPU

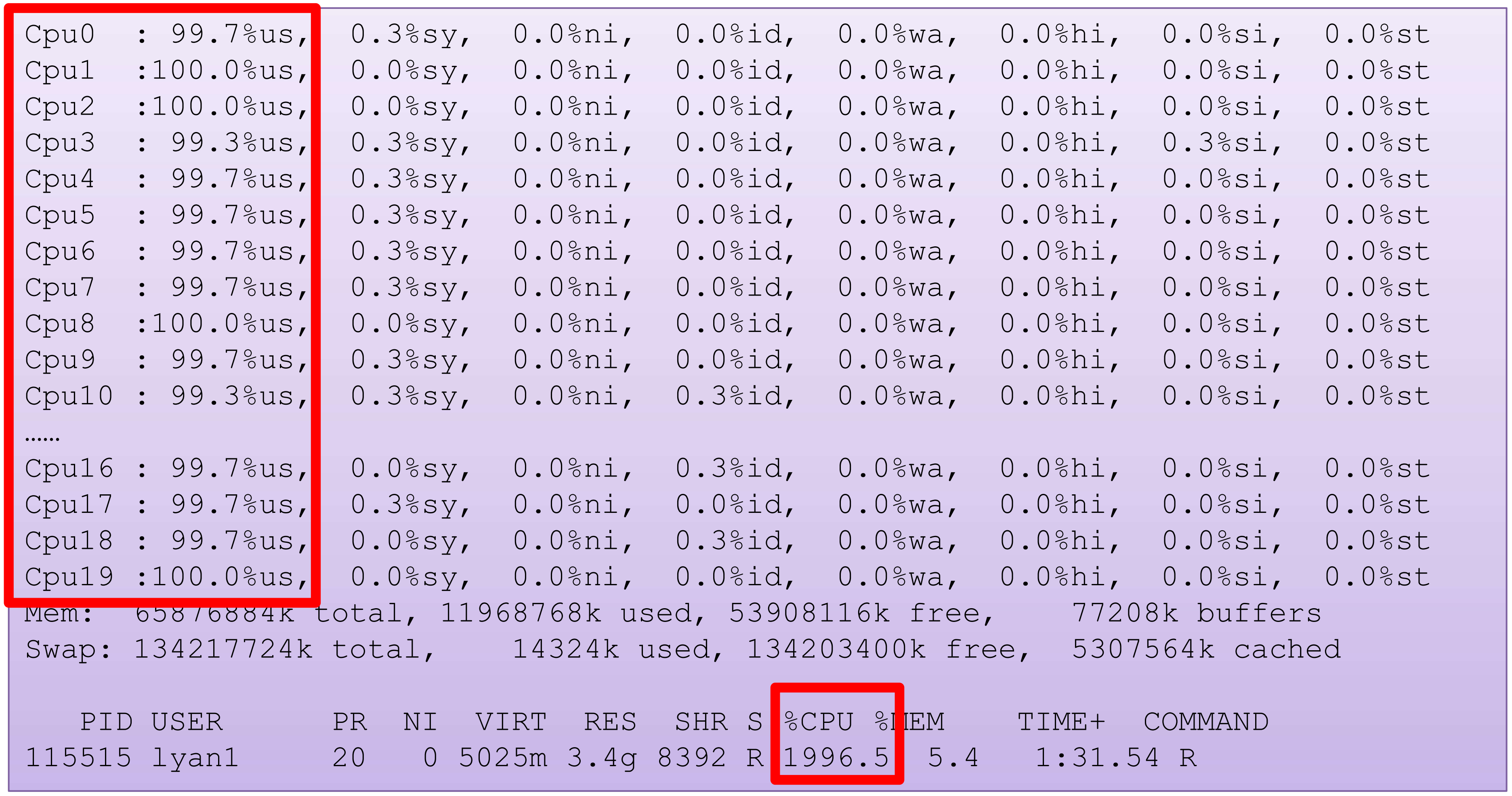
Forms of Parallelism in R

- Implicit parallelism
  - Use parallel libraries
- Explicit parallelism
  - Use parallel packages in R
  - We will focus on the `parallel` package

Implicit Parallelism

- Some functions in R can call parallel numerical libraries
  - On LONI and LSU HPC clusters this is the multi-threaded Intel MKL library
  - Mostly linear algebraic and related functions
    - Example: linear regression, matrix decomposition, computing inverse and determinant of a matrix

```
# on QB2 R is built with Intel MKL library, which is multi-threaded
> system("ldd /usr/local/packages/r/3.1.0/INTEL-14.0.2/lib64/R/lib/libR.so")
        linux-vdso.so.1 =>  (0x00007fff1e17c000)
        libmkl_intel_lp64.so =>
/usr/local/compilers/Intel/cluster_studio_xe_2013.1.046/composer_xe_2013_sp1.2.14
4/mkl/lib/intel64/libmkl_intel_lp64.so (0x00002b27f7895000)
        libmkl_intel_thread.so =>
/usr/local/compilers/Intel/cluster_studio_xe_2013.1.046/composer_xe_2013_sp1.2.14
4/mkl/lib/intel64/libmkl_intel_thread.so (0x00002b27f7fdc000)
        libmkl_core.so =>
/usr/local/compilers/Intel/cluster_studio_xe_2013.1.046/composer_xe_2013_sp1.2.14
4/mkl/lib/intel64/libmkl_core.so (0x00002b27f8fc6000)
……
```



```
Cpu0  : 99.7%us,  0.3%sy,  0.0%ni,  0.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu1  :100.0%us,  0.0%sy,  0.0%ni,  0.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu2  :100.0%us,  0.0%sy,  0.0%ni,  0.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu3  : 99.3%us,  0.3%sy,  0.0%ni,  0.0%id,  0.0%wa,  0.0%hi,  0.3%si,  0.0%st
Cpu4  : 99.7%us,  0.3%sy,  0.0%ni,  0.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu5  : 99.7%us,  0.3%sy,  0.0%ni,  0.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu6  : 99.7%us,  0.3%sy,  0.0%ni,  0.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu7  : 99.7%us,  0.3%sy,  0.0%ni,  0.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu8  :100.0%us,  0.0%sy,  0.0%ni,  0.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu9  : 99.7%us,  0.3%sy,  0.0%ni,  0.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu10 : 99.3%us,  0.3%sy,  0.0%ni,  0.3%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
……
Cpu16 : 99.7%us,  0.0%sy,  0.0%ni,  0.3%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu17 : 99.7%us,  0.3%sy,  0.0%ni,  0.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu18 : 99.7%us,  0.0%sy,  0.0%ni,  0.3%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu19 :100.0%us,  0.0%sy,  0.0%ni,  0.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:   65876884k total, 11968768k used, 53908116k free,    77208k buffers
Swap: 134217724k total,    14324k used, 134203400k free,  5307564k cached

   PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
115515 lyan1     20   0 5025m 3.4g 8392 R 1996.5  5.4   1:31.54 R
```

```
# Matrix inverse is implicitly parallel on QB2
A <- matrix(rnorm(10000*10000),10000,10000)
Ainv <- solve(A)
```

Caution with Implicit Parallelism

- Do not run many R instances if they use parallel libraries
- Example: Running 10 R instances with each using 20 workers will spawn 200 workers

Outline

- Parallel computing primers
- Parallel computing with R
  - Implicit parallelism
  - Explicit parallelism
- R with GPU

Explicit Parallelism - `parallel` Package

- Introduced in R 2.14.1
- Integrated previous `multicore` and `snow` packages
- Coarse-grained parallelization
  - Suit for the chunks of computation are unrelated and do not need to communicate
- Two ways of using `parallel` packages
  - `mc*apply` function
  - `for` loop with `%dopar%`
    - Need `foreach` and `doParallel` packages

Function `mclapply`

- Parallelized version of the `lapply` function
  - Similar syntax

```
mclapply(X, FUN, mc.cores = <number of cores>, …)
```

  - Return a list of the same length as X, each element of which is the result of applying ‘FUN’ to the corresponding element of X
- Can use all cores on one node
  - But not on multiple nodes

```r
# Quadratic Equation: a*x^2 + b*x + c = 0
solve.quad.eq <- function(a, b, c)
{
 # Return solutions
 x.delta <- sqrt(b*b - 4*a*c)
 x1 <- (-b + x.delta)/(2*a)
 x2 <- (-b - x.delta)/(2*a)

 return(c(x1, x2))
}

len <- 1e7
a <- runif(len, -10, 10); b <- runif(len, -10, 10); c <- runif(len, -10, 10)

#Serial: lapply
res1.s <- lapply(1:len, FUN = function(x) { solve.quad.eq(a[x], b[x], c[x])})

#Parallel: mclapply with 4 cores
library(parallel)
res1.p <- mclapply(1:len,
                   FUN = function(x) { solve.quad.eq(a[x], b[x], c[x]) },
                   mc.cores = 4)
```

```
# Quadratic Equation: a*x^2 + b*x + c = 0
solve.quad.eq <- function(a, b, c)
{
 # Return solutions
 x.delta <- sqrt(b*b - 4*a*c)
 x1 <- (-b + x.delta)/(2*a)
```

```
> system.time(
+res1.s <- lapply(1:len, FUN = function(x) { solve.quad.eq(a[x], b[x], c[x])})
)
   user  system elapsed
358.878   0.375 359.046
> system.time(
+ res1.p <- mclapply(1:len,
+                     FUN = function(x) { solve.quad.eq(a[x], b[x], c[x]) },
+                     mc.cores = 4)
+ )
   user  system elapsed
 11.098   0.342  81.581
```

```
library(parallel)
res1.p <- mclapply(1:len,
                   FUN = function(x) { solve.quad.eq(a[x], b[x], c[x]) },
                   mc.cores = 4)
```

%dopar%

- From `doParallel` package
  - On top of packages `parallel`, `foreach`, `iterator`
- Purpose: parallelize a `for` loop
- Can run on multiple nodes
- Steps
  - Create a cluster of workers
  - Register the cluster
  - Process the for loop in parallel
  - Stop the cluster

%dopar%: On A Single Node

```
# Workload:
# Create 1,000 random samples, each with
# 1,000,000 observations from a standard
# normal distribution, then take a #
summary for each sample.

iters <- 1000

# Sequential version
for (i in 1:iters) {
  to.ls <- rnorm(1e6)
  to.ls <- summary(to.ls)
}
```

```
# Parallel version with %dopar%

# Step 1: Create a cluster of 4 workers
cl <- makeCluster(4)

# Step 2: Register the cluster
registerDoParallel(cl)

# Step 3: Process the loop
ls<-foreach(icount(iters)) %dopar% {
  to.ls<-rnorm(1e6)
  to.ls<-summary(to.ls)
}

# Step 4: Stop the cluster
stopCluster(cl)
```

%dopar%: On A Single Node

```
# Workload:
# Create 1,000 rando
# 1,000,000 observat
# normal distributio
summary for each samp

iters <- 1000

# Sequential version
for (i in 1:iters) {
  to.ls <- rnorm(1e6)
  to.ls <- summary(t
}
```

```
# Sequential
> system.time(
+ for (i in 1:iters) {
+   to.ls <- rnorm(1e6)
+   to.ls <- summary(to.ls)
+ }
+ )
   user  system elapsed
 60.249   3.499  63.739

# Parallel with 4 cores
> system.time({
+ cl <- makeCluster(4)
+ registerDoParallel(cl)
+ ls<-foreach(icount(iters)) %dopar% {
+   to.ls<-rnorm(1e6)
+   to.ls<-summary(to.ls)
+ }
+ stopCluster(cl)
+ })
   user  system elapsed
  0.232   0.032  17.738
```

```
ith %dopar%

luster of 4 workers


he cluster
l)

e loop
ters)) %dopar% {

ls)


luster
```

`makeCluster()`

- On the same node:

```
cl <- makeCluster(<number of workers>)
```

- On multiple nodes:

```
cl <- makeCluster(<list of hostnames>)
```

  – Example: create 4 workers, 2 on `qb101` and 2 on `qb102`

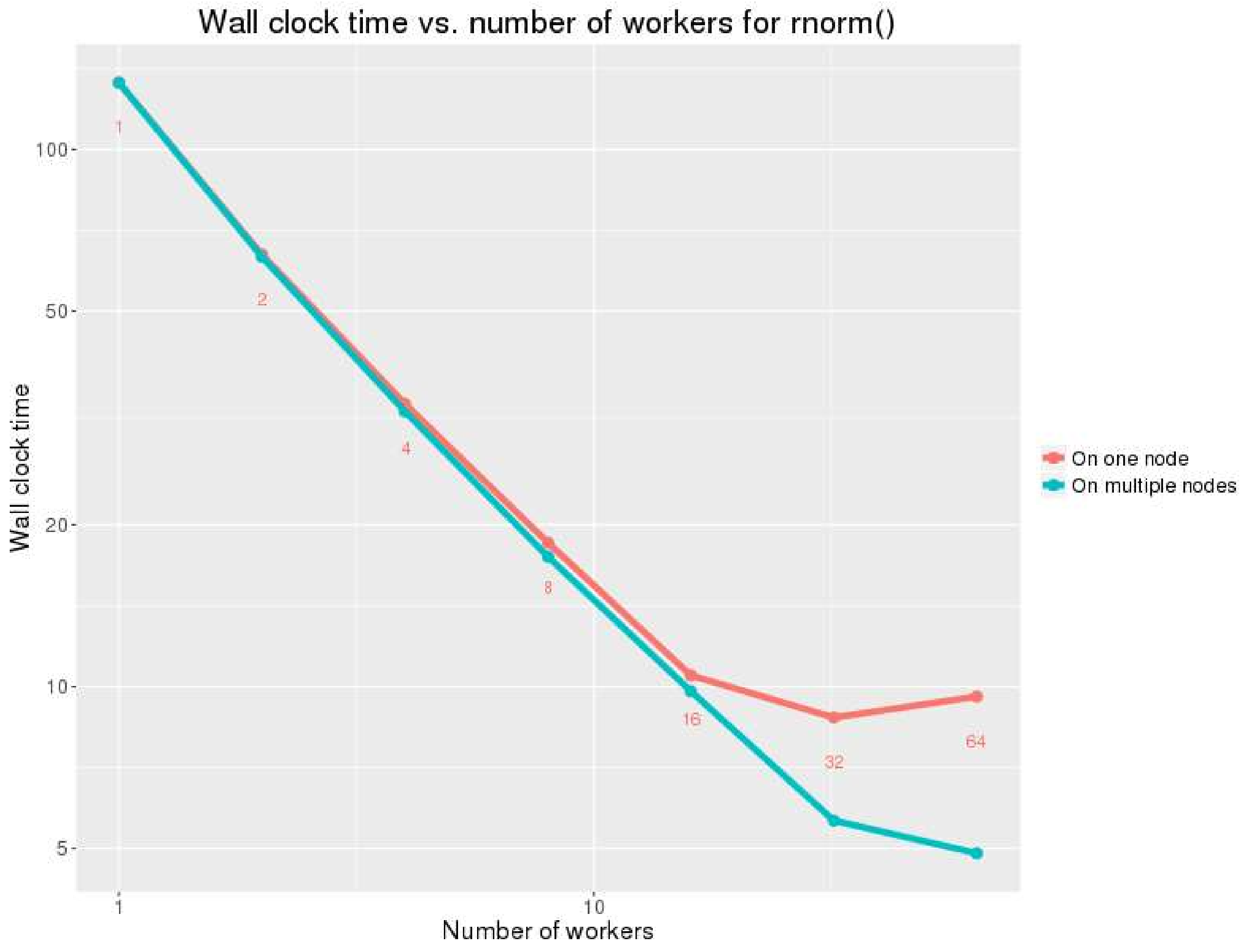
```
cl <- makeCluster(c("qb101","qb101","qb102","qb102"))
```

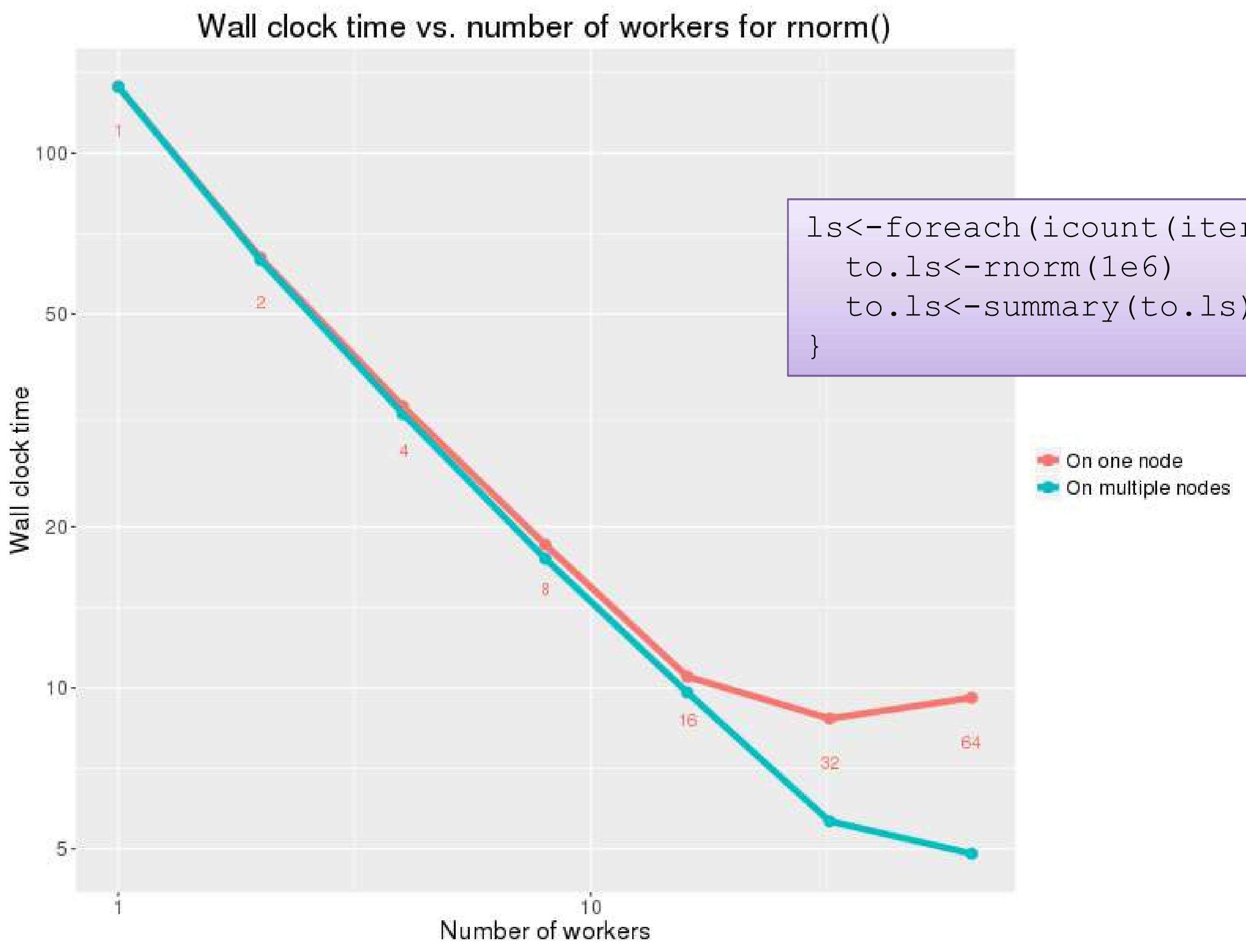
`%dopar%`: On Multiple Nodes

```
# Read all host names
hosts <-
as.vector(unique(read.table(Sys.getenv("PBS_NODEFILE")
,stringsAsFactors=F))[,1])
# Count number of hosts
nh <- length(hosts)
# Use 4 workers
nc <- 4

# Make a cluster on multiple nodes
cl <- makeCluster(rep(hosts , each = nc/nh))
registerDoParallel(cl)

ls<-foreach(icount(iters)) %dopar% {

  to.ls<-rnorm(1e6)
  to.ls<-summary(to.ls)

}

stopCluster(cl)
```

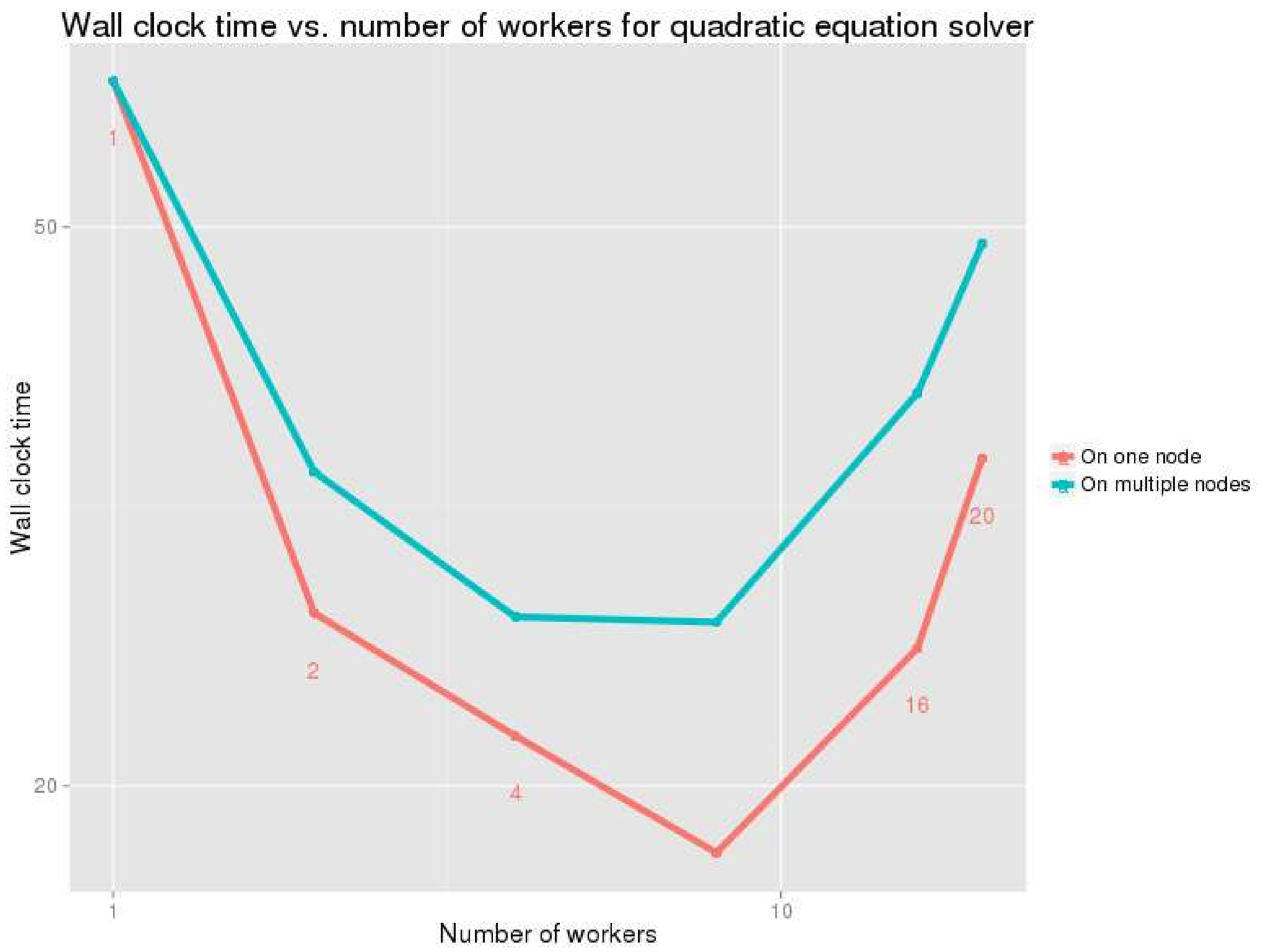
The Million Second Question

- How many workers should one use?
- What should be the standard?


Wall clock time vs. number of workers for rnorm()
Wall clock time
100
50
20
10
5
Number of workers
1
10
1
2
4
8
16
32
64
On one node
On multiple nodes

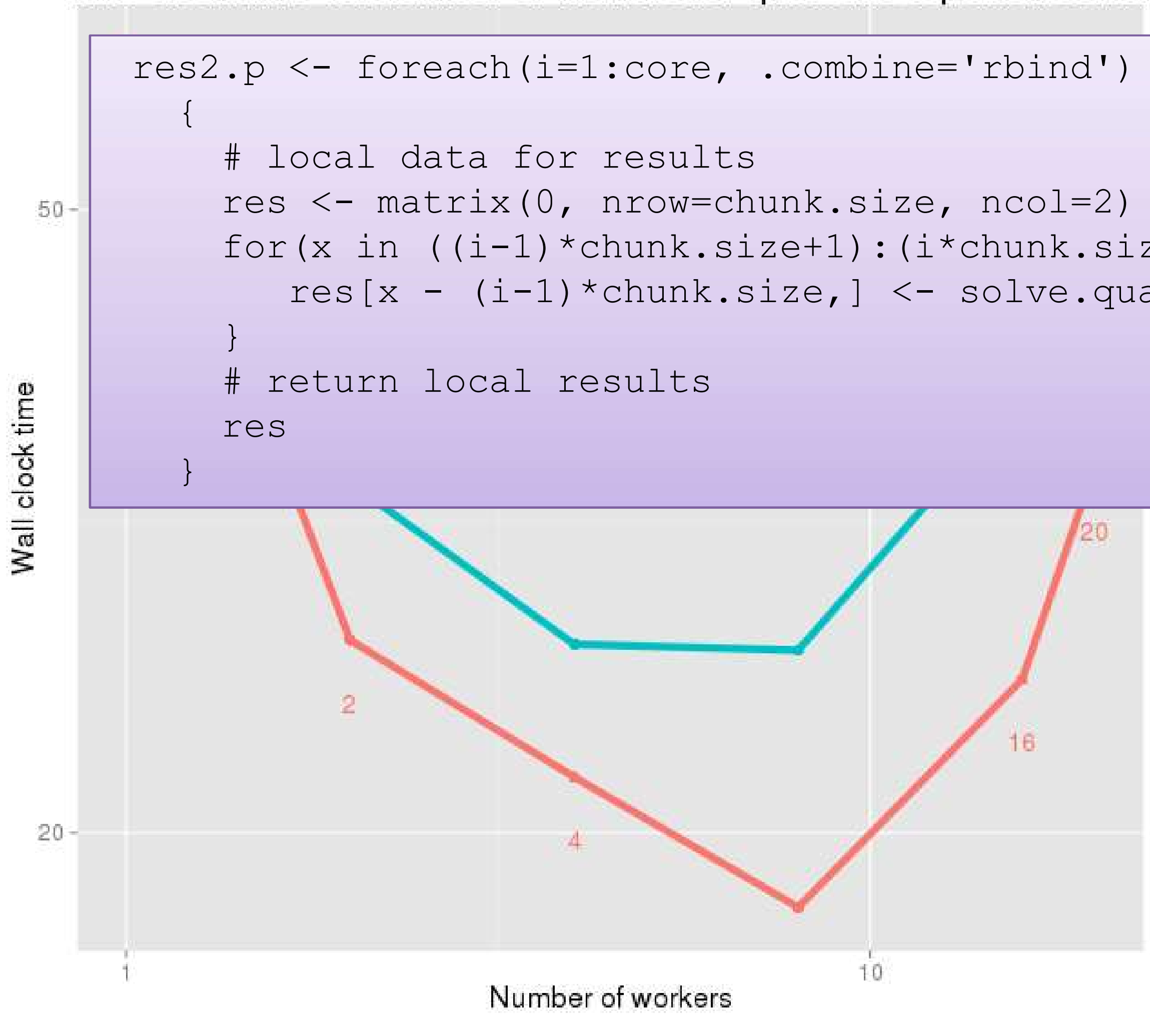


```
ls<-foreach(icount(iters)) %dopar% {
  to.ls<-rnorm(1e6)
  to.ls<-summary(to.ls)
}
```



Wall clock time vs. number of workers for quadratic equation solver
Wall clock time
50
20
1
10
Number of workers
1
2
4
16
20
On one node
On multiple nodes

Wall clock time vs. number of workers for quadratic equation solver



```
res2.p <- foreach(i=1:core, .combine='rbind') %dopar%
  {
    # local data for results
    res <- matrix(0, nrow=chunk.size, ncol=2)
    for(x in ((i-1)*chunk.size+1):(i*chunk.size)) {
       res[x - (i-1)*chunk.size,] <- solve.quad.eq(a[x], b[x], c[x])
    }
    # return local results
    res
  }
```

The Million Second Question

- Parallel programs have overhead
  - Extra time spent on coordinating workers, which cause efficiency to drop with increasing number of workers
- How many workers should one use?
  - It depends
- What should be the standard?

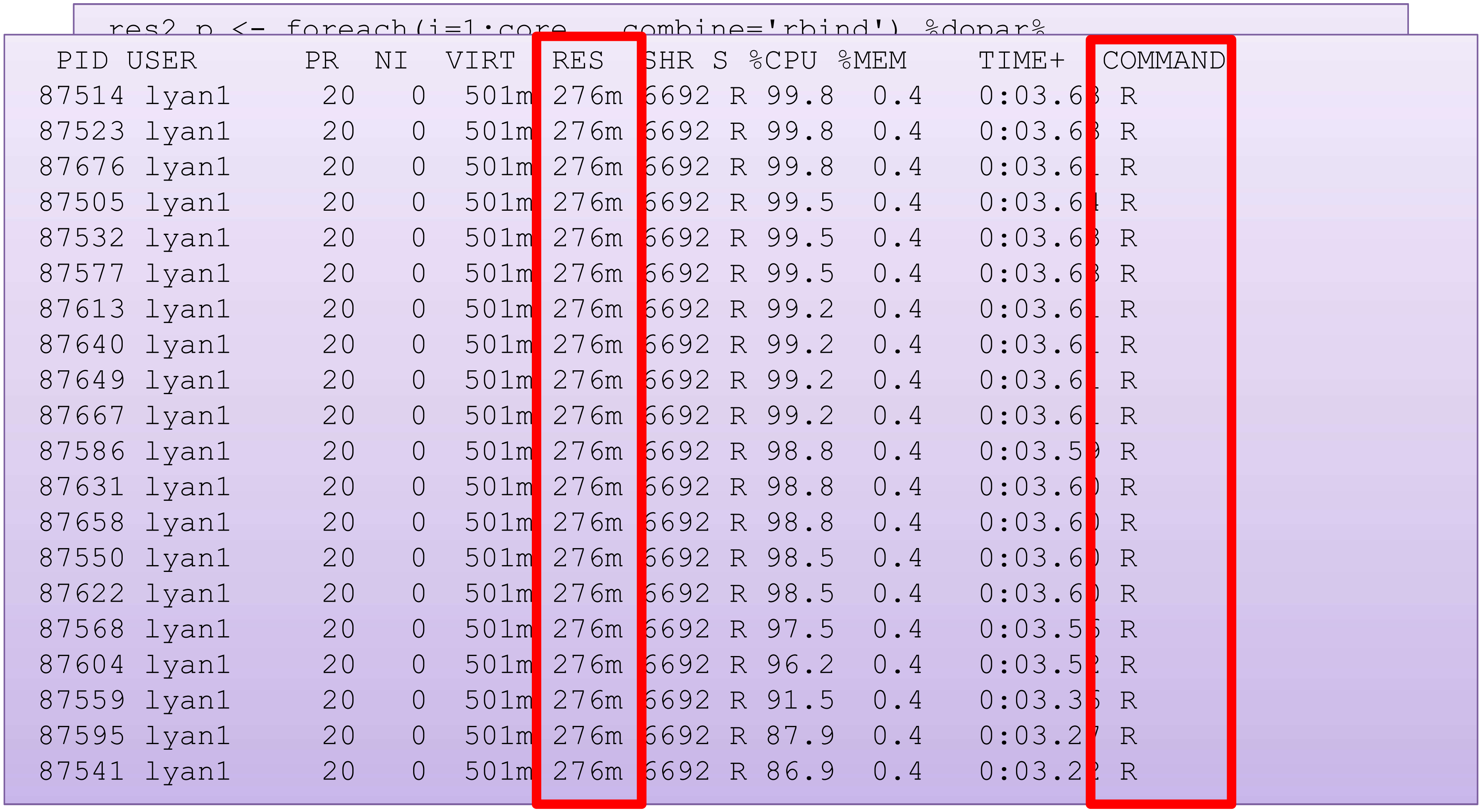
Memory Management

- Replica of data objects could be generated for every worker
  – Memory usage would increase with the number of workers
- R does not necessarily clean them up even if you close the cluster

```
res2.p <- foreach(i=1:core, .combine='rbind') %dopar%
  {
    # local data for results
    res <- matrix(0, nrow=chunk.size, ncol=2)
    for(x in ((i-1)*chunk.size+1):(i*chunk.size)) {
       res[x - (i-1)*chunk.size,] <- solve.quad.eq(a[x], b[x], c[x])
    }
    # return local results
    res
  }
```

```
   PID USER       PR  NI   VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 87483 lyan1      20   0   539m 314m 5692 R 100.0  0.5   0:02.0  R
 87492 lyan1      20   0   539m 314m 5692 R 100.0  0.5   0:02.0  R
 87465 lyan1      20   0   539m 314m 5692 R  99.4  0.5   0:02.0  R
 87474 lyan1      20   0   539m 314m 5692 R  99.4  0.5   0:02.0  R
```

4 workers



```
res2.p <- foreach(i=1:core, .combine='rbind') %dopar%
```

```
  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
87514 lyan1     20   0  501m 276m 6692 R 99.8  0.4   0:03.63 R
87523 lyan1     20   0  501m 276m 6692 R 99.8  0.4   0:03.63 R
87676 lyan1     20   0  501m 276m 6692 R 99.8  0.4   0:03.61 R
87505 lyan1     20   0  501m 276m 6692 R 99.5  0.4   0:03.64 R
87532 lyan1     20   0  501m 276m 6692 R 99.5  0.4   0:03.63 R
87577 lyan1     20   0  501m 276m 6692 R 99.5  0.4   0:03.63 R
87613 lyan1     20   0  501m 276m 6692 R 99.2  0.4   0:03.61 R
87640 lyan1     20   0  501m 276m 6692 R 99.2  0.4   0:03.61 R
87649 lyan1     20   0  501m 276m 6692 R 99.2  0.4   0:03.61 R
87667 lyan1     20   0  501m 276m 6692 R 99.2  0.4   0:03.61 R
87586 lyan1     20   0  501m 276m 6692 R 98.8  0.4   0:03.59 R
87631 lyan1     20   0  501m 276m 6692 R 98.8  0.4   0:03.60 R
87658 lyan1     20   0  501m 276m 6692 R 98.8  0.4   0:03.60 R
87550 lyan1     20   0  501m 276m 6692 R 98.5  0.4   0:03.60 R
87622 lyan1     20   0  501m 276m 6692 R 98.5  0.4   0:03.60 R
87568 lyan1     20   0  501m 276m 6692 R 97.5  0.4   0:03.55 R
87604 lyan1     20   0  501m 276m 6692 R 96.2  0.4   0:03.52 R
87559 lyan1     20   0  501m 276m 6692 R 91.5  0.4   0:03.35 R
87595 lyan1     20   0  501m 276m 6692 R 87.9  0.4   0:03.27 R
87541 lyan1     20   0  501m 276m 6692 R 86.9  0.4   0:03.22 R
```

20 workers

Outline

- Parallel computing primers
- Parallel computing with R
  - Implicit parallelism
  - Explicit parallelism
- R with GPU

R with GPU

- GPU stands for Graphic Processing Unit
  - Can accelerate certain types of computation
  - All nodes on LONI QB2 cluster are equipped with GPU
- Package `gpuR` brings the processing power of GPU to R

```
> ORDER <- 8192
> A = matrix(rnorm(ORDER^2), nrow=ORDER)
> B = matrix(rnorm(ORDER^2), nrow=ORDER)
> vclA = vclMatrix(rnorm(ORDER^2), nrow=ORDER, ncol=ORDER)
> vclB = vclMatrix(rnorm(ORDER^2), nrow=ORDER, ncol=ORDER)
> system.time(C <- A %*% B)
   user  system elapsed
 55.375   0.189   2.901
> system.time(vclC <- vclA %*% vclB)
   user  system elapsed
  0.011   0.002   0.047
```

Deep Learning in R

- Since 2012, Deep Neural Network (DNN) has gained great popularity in applications such as
  - Image and pattern recognition
  - Natural language processing
- There are a few R packages that support DNN
  - MXNet (multiple nodes with GPU support)
  - H2o (multiple nodes)
  - Darch
  - Deepnet
  - Rpud

References

- ParallelR (www.parallelr.com)
  - Code: https://github.com/PatricZhao/ParallelR
- R Documentation for packages mentioned in this tutorial

Training Next Week

- March 29th: Intermediate Python Programming
  - Focus on popular Python modules numpy, matplotlib and scipy to get users familiar with building quick Python real world computing solutions.
  - Serve as a quick crash training on Python for the upcoming tutorial on Machine Learning.
- March 28th: Agave Platform: Running Jobs on HPC Without the Command Line
  - The Agave Platform was developed to make it easy for scientists to take their existing code and "webify" it, or enable it for use as a Gateway.

INFORMATION TECHNOLOGY SERVICES

Thank you!

CENTER FOR COMPUTATION & TECHNOLOGY