Lecture 16 - Fork Parallelism Part 1

DSE 512

Drew Schmidt
2022-03-24

From Last Time

- Homework 3 posted tonight some time
- Questions?

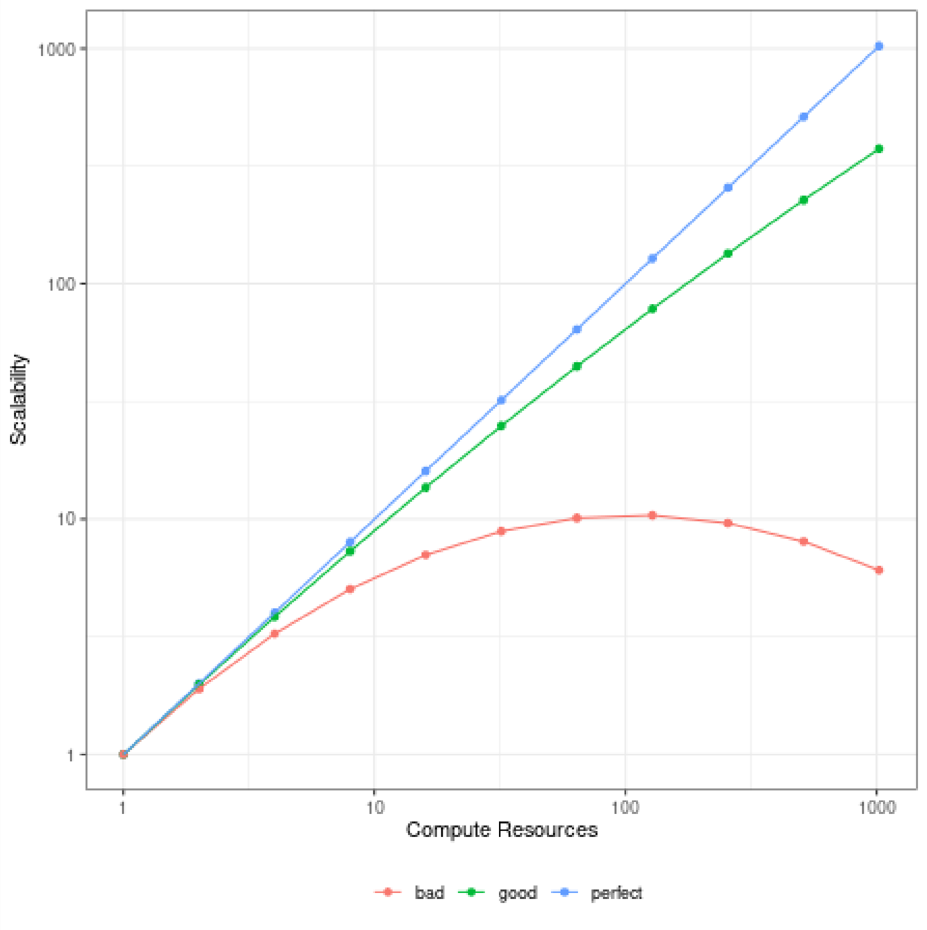
Parallelism Basics

Vocabulary

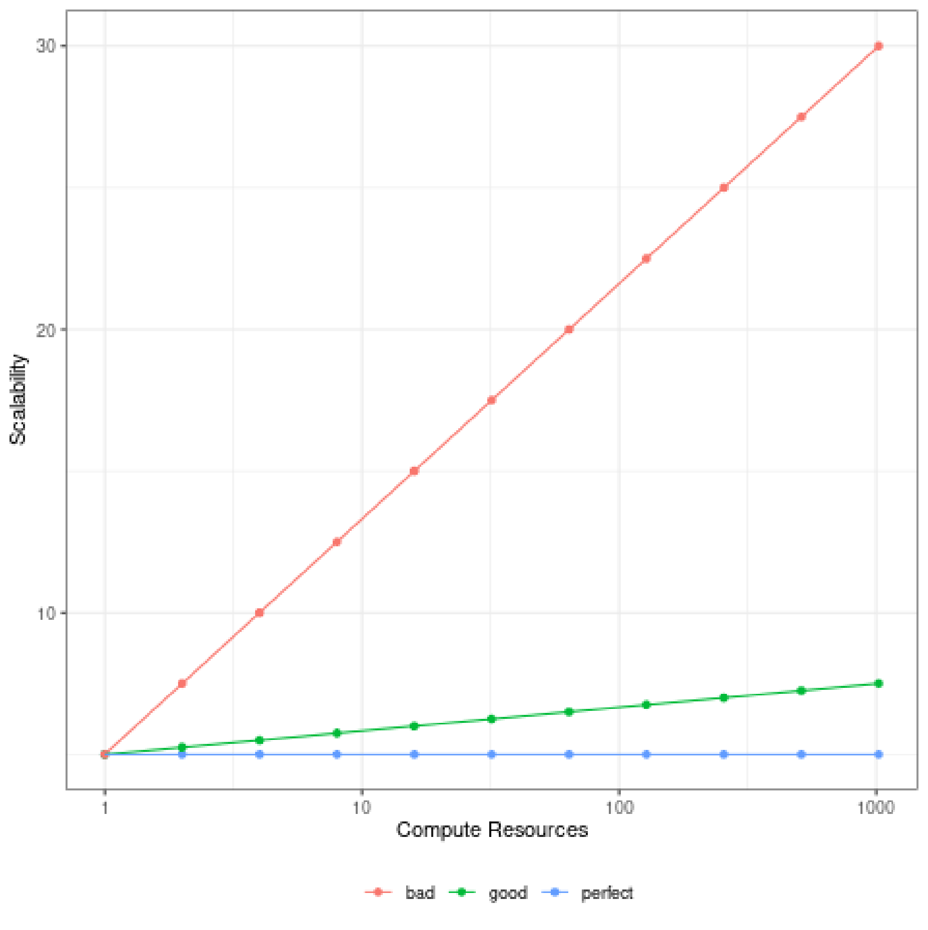
- *Implicit parallelism* - Parallel details hidden from user
  - Example: Using multi-threaded BLAS
- *Explicit parallelism* - Some assembly required
  - Example: Using the `mclapply()` from the parallel package
- *Embarrassingly Parallel* aka *naturally parallel* aka *loosely coupled* Obvious how to make parallel; lots of independence in computations
  - Example: Fit two independent models
- *Tightly Coupled* - Opposite of embarrassingly parallel; lots of dependence in computations
  - Example: Speed up model fitting for one model

Scalability

Strong Scaling



Weak Scaling



Speedup

- *Wallclock Time* - Time of the clock on the wall from start to finish
- *Speedup* - unitless measure of improvement; more is better.

$$S_{n_1,n_2} = \frac{\text{Time for } n_1 \text{ cores}}{\text{Time for } n_2 \text{ cores}}$$

  - n_1 is often taken to be 1
  - In this case, comparing parallel algorithm to serial algorithm

Independence

- Parallelism requires *independence*
- Ability to do different work at the same time
- Single problem split into many sub-problems
- Algorithm vs Implementation
  - Sometimes the algorithm is the hard part
  - Sometimes the parallel interface is
  - Sometimes it's both

Independence Thought Experiment

- I have 10 laborers each with a shovel
- What if I need 10 holes dug?
- 5 holes?
- 20 holes?
- One really big hole?

Independence in Practice

- R/Python function evals almost always independent
  - I/O may break this
  - Some HPC packages break this in memory (e.g. fmlr)
  - Other languages violate this all the time (e.g. C, Fortran, ...)
- Some examples by area:
  - Bio: split by gene
  - Chemistry/Materials: split by atom
  - Stats: split loglik sum
  - Bayesian: MCMC with multiple chains
  - ML: fit multiple models at once

Overhead

- ***ALL*** parllel methods have overhead
- Sometimes it is undetectable
- Sometimes it is very noticeable
- Sometimes it is worse than serial!

Fork

- Kernel-level parallelism mechanism
- Inexpensive process clone mechanism
- Only supported on Linux and MacOS
- Data is copy on modify!

Fork in R and Python

- R: parallel package
- Python: multiprocessing package
- Both packages have other interfaces and backends (e.g. sockets)
- You can use that if you want (Windows), but we won't focus on it

Fork Parallelism in R

`lapply()` becomes `mclapply()`:

```
mclapply(X, FUN, ..., mc.preschedule = TRUE, mc.set.seed = TRUE,
    mc.silent = FALSE, mc.cores = getOption("mc.cores", 2L),
    mc.cleanup = TRUE, mc.allow.recursive = TRUE, affinity.list = NULL)
```

Basic Example in R

```
x = lapply(1:5, sqrt)
unlist(x)
```

```
## [1] 1.000000 1.414214 1.732051 2.000000 2.236068
```

```
ncores = parallel::detectCores()
x_mc = parallel::mclapply(1:5, sqrt, mc.cores=ncores)
all.equal(x, x_mc)
```

```
## [1] TRUE
```

Fork Parallelism in Python

- A Bit more complicated…
- We'll try to keep it simple

Basic Example in Python

```python
import math
import multiprocessing as mp

list(map(math.sqrt, range(6)))
```

```
## [0.0, 1.0, 1.4142135623730951, 1.7320508075688772, 2.0, 2.23606797749979]
```

```python
ncores = mp.cpu_count()
pool = mp.Pool(ncores)
pool.map(math.sqrt, range(6))
```

```
## [0.0, 1.0, 1.4142135623730951, 1.7320508075688772, 2.0, 2.23606797749979]
```

"Portability"

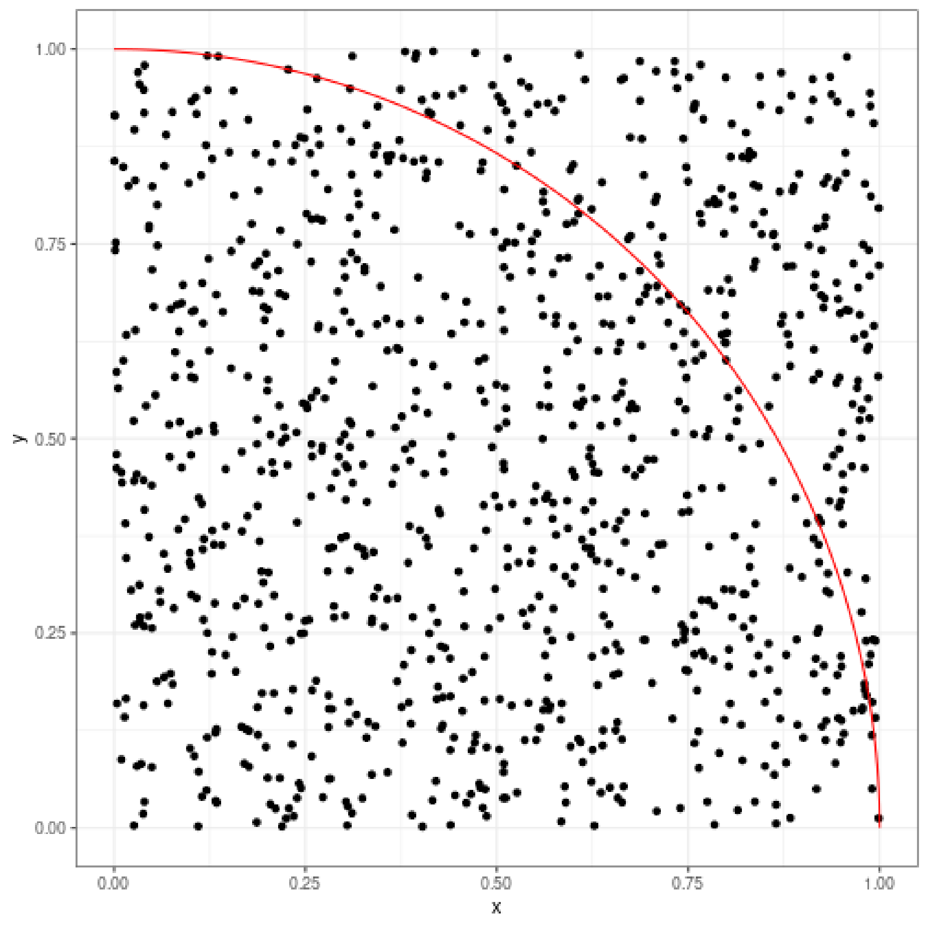
- Not all OS's supported
- No fork on Windows
- Many API's try to hide this
- R and Python take different strategies
    - `mclapply()` errors; instead use `makeCluster()`
    - `Pool()` has a `Spawn()` method for Windows
    - ***DO NOT DESTROY YOUR WORKER POOL UNTIL YOU ARE DONE***

Estimating π in Python

Monte Carlo π Simulation

$\text{Area} = \pi r^2$

```
set.seed(1234)
n = 1000
x = runif(n)
y = runif(n)

circ_pts = seq(0, pi/2,length.out=100)
library(ggplot2)
g = ggplot(data.frame(x=x, y=y), aes(x, y
  theme_bw() +
  geom_point() +
  annotate("path",
   x = 0 + 1*cos(circ_pts),
   y = 0 + 1*sin(circ_pts),
   color="red"
  )
```



Serial Implementation

```python
import random

def mc_pi(n):
  acc = 0
  for i in range(n):
    x = random.random()
    y = random.random()
    if x**2 + y**2 < 1:
      acc += 1
  return 4 * acc / n

random.seed(1234)
mc_pi(1000)
```

```
## 3.108
```

Parallel Implementation

```python
import multiprocessing as mp
import numpy as np
import random

def mc_pi_count(n):
  acc = 0
  for i in range(n):
    x = random.random()
    y = random.random()
    if x**2 + y**2 < 1:
      acc += 1
  return acc

def mc_pi_par(n, seed, ncores):
  random.seed(seed)
  pool = mp.Pool(ncores)
  acc = pool.map(mc_pi_count,
    np.repeat(round(n/ncores),
    ncores)
  )
  return sum(acc) * 4 / n
```

```python
n = 10000000
for nc in [1, 2, 4]:
  t.start()
  pi_est = mc_pi_par(n, 1234, nc)
  t.stop()
```

```
## 4.66501244995743
## 2.4230600469745696
## 1.2336212289519608
```

What About Numba?

```python
from numba import jit

@jit(nopython = True)
def mc_pi_count(n):
  acc = 0
  for i in range(int(n)):
    x = random.random()
    y = random.random()
    if x**2 + y**2 < 1:
      acc += 1
  return acc
```

```python
warmup = mc_pi_count(10)
n = 10000000
for nc in [1, 2, 4]:
  t.start()
  pi_est = mc_pi_par(n, 1234, nc)
  t.stop()
```

```
## 0.2649029800668359
## 0.09507097897585481
## 0.07486061903182417
```

"Random" Numbers

- ~~RNG~~ PRNG
- Streams can overlap
- Be careful!
- Like all things:
  - Easy in R
  - Painful in Python

RNG in Parallel

- *Random Number Generation* by Pierre L'Ecuyer
  https://www.iro.umontreal.ca/~lecuyer/myftp/papers/handstat.pdf
- R
  - It just works
  - Read `?parallel::mcparallel` if you want
- Python
  - Depends *a lot* on the generator
  - Hard to do correctly
  - Better than nothing strategy: pass additional process-unique seed arg and use `starmap`
  - You're on your own

Simulating Monty Hall in R

The Monty Hall Problem

- You are shown 3 doors
- Behind 1 is a prize
- The other 2 have nothing/junk
- You choose a door
- One of the remaining 2 doors is shown to be junk
- You can switch to the remaining door if you choose
- What is the probability of winning if you switch?

Simulating One Game

```r
monty_hall = function(.)
{
  doors = 1:3
  prize_door = sample(doors, size=1)
  choice_door = sample(doors, size=1)
  # Always switch
  if (choice_door == prize_door)
    0 # lose
  else
    1 # win
}
```

```r
set.seed(1234)
monty_hall()
```

```
## [1] 0
```

```r
monty_hall()
```

```
## [1] 1
```

```r
monty_hall()
```

```
## [1] 0
```

```r
monty_hall()
```

```
## [1] 0
```

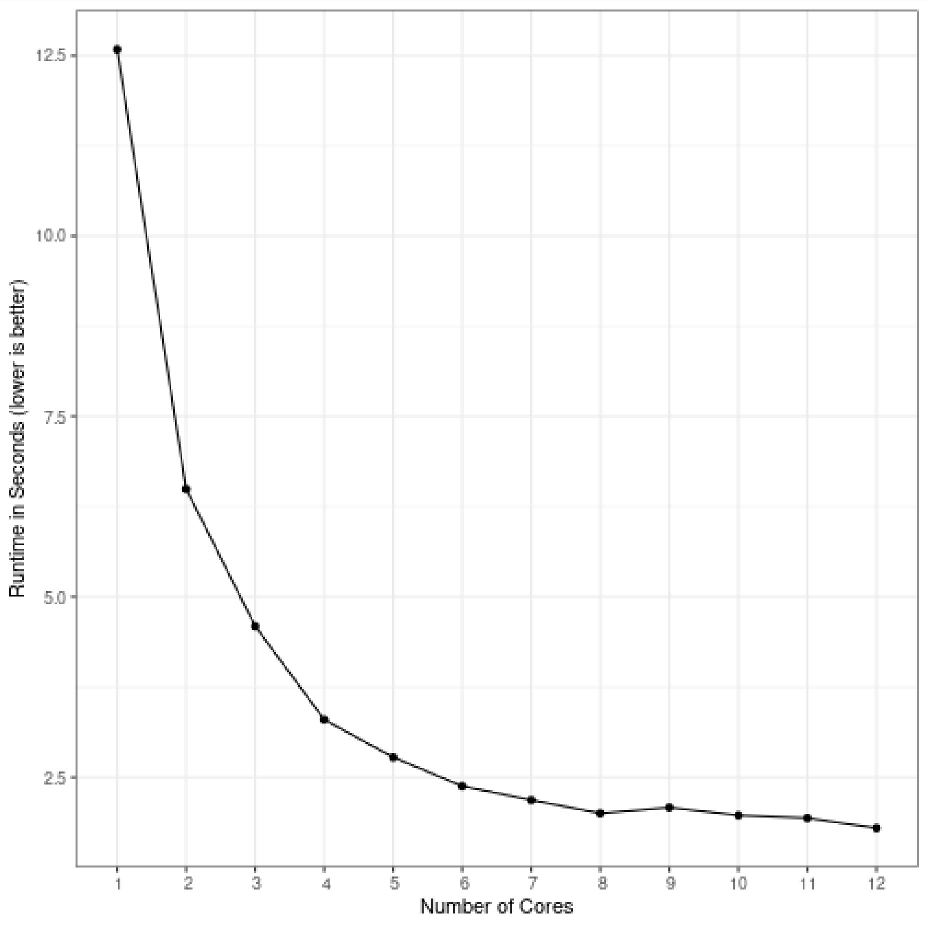
Simulating Many Games

```r
many_monty_hall = function(nruns)
{
  runtime = system.time({
    runs = lapply(1:nruns, monty_hall)
  })[3]

  c(mean(unlist(runs)), runtime)
}

set.seed(1234)
nruns = 1e6
many_monty_hall(nruns)
```

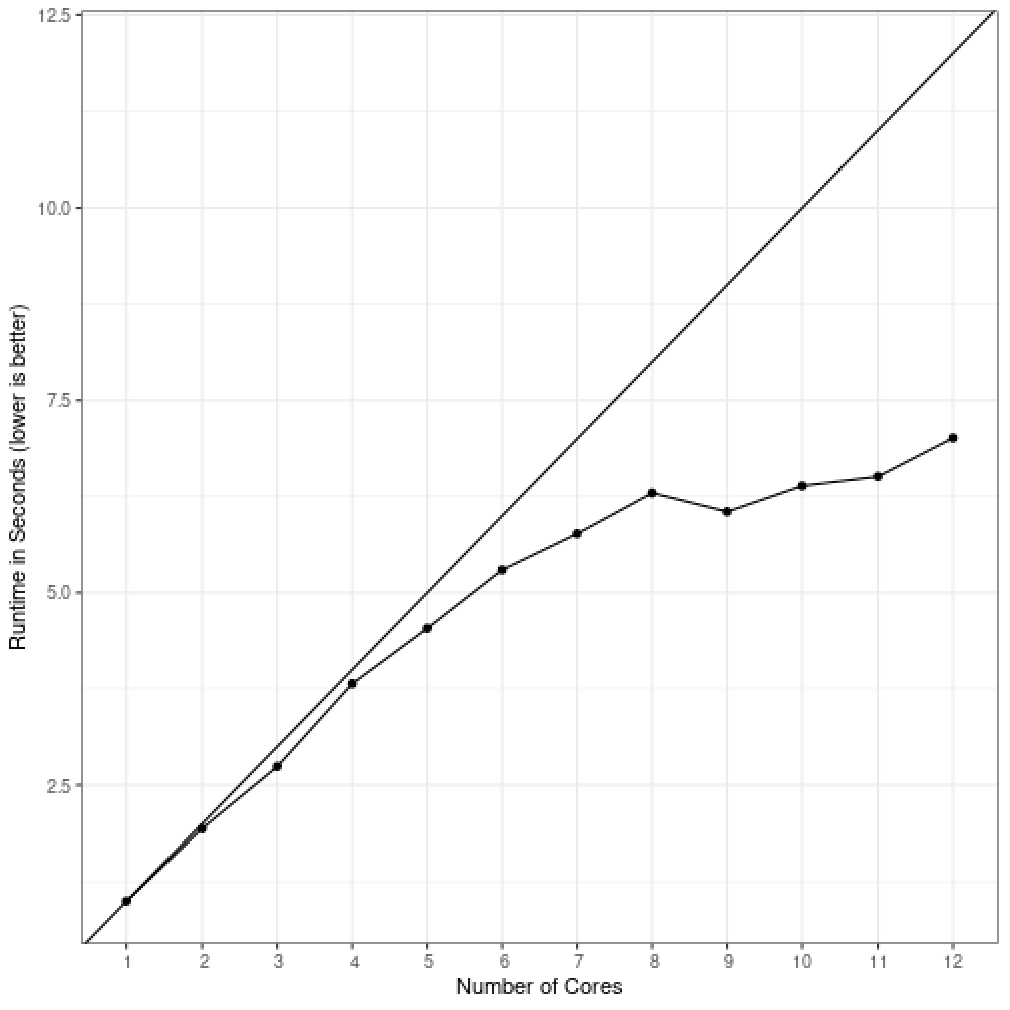
```
##              elapsed
##  0.667541 12.668000
```

Going Parallel

```r
library(parallel)
many_monty_hall = function(nruns, ncores)
{
  runtime = system.time({
    runs = mclapply(1:nruns, monty_hall, r
  })[3]

  c(mean(unlist(runs)), runtime)
}

set.seed(1234)
ncores = 1:12
nruns = 1e6
mmh = lapply(ncores, many_monty_hall, nru
df = cbind(ncores, do.call(rbind, mmh)) |
  as.data.frame()
```

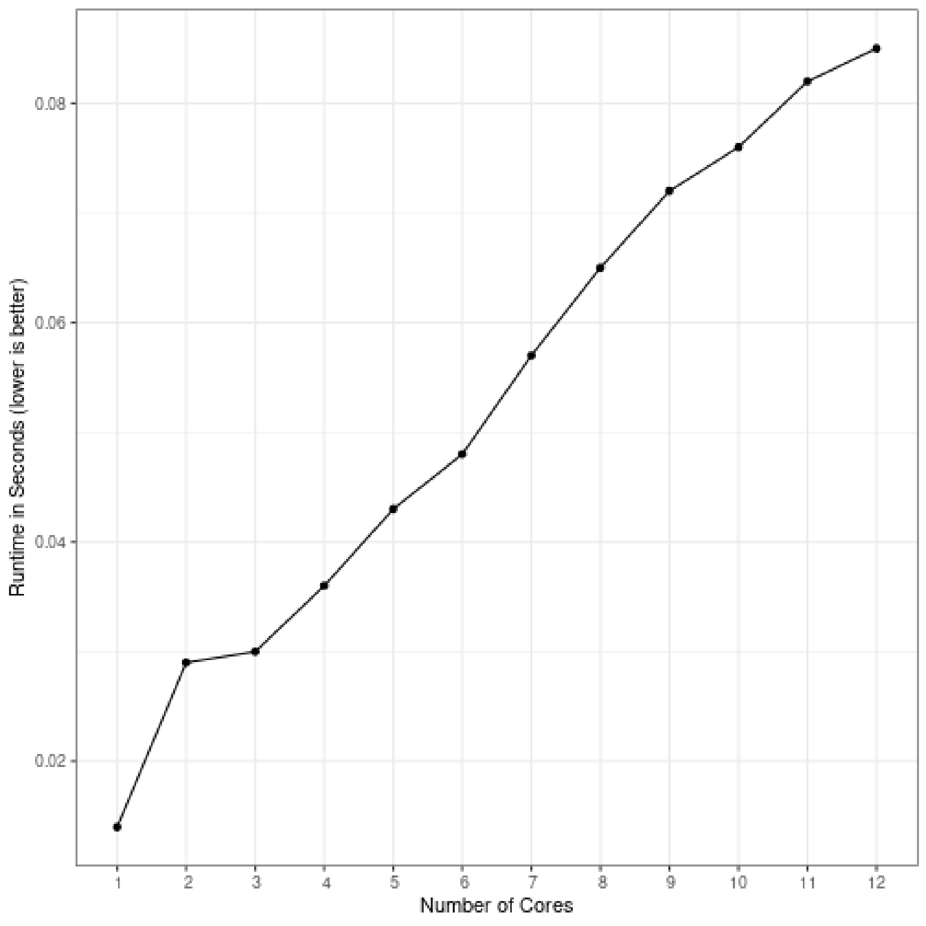


Parallel Speedup

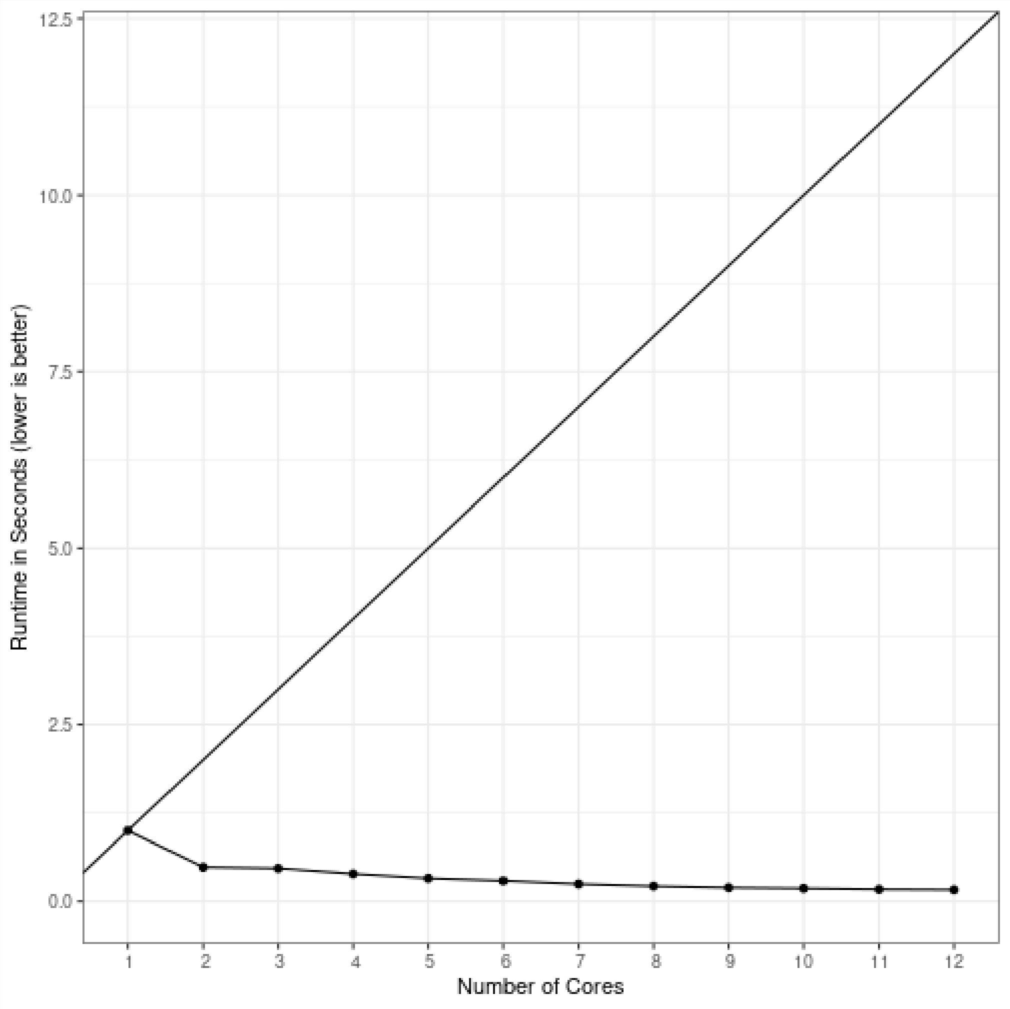


Going Parallel Again

```
set.seed(1234)
ncores = 1:12
nruns = 1e3
mmh = lapply(ncores, many_monty_hall, nru
df = cbind(ncores, do.call(rbind, mmh)) |
  as.data.frame()
```



Parallel Speedup Again



Improving k-Means Clustering Outcomes in Parallel

Parallelism in k-Means Clustering

- Parallelize the algorithm itself
  - Good for very large datasets
  - Tricky…
- Parallelize on the number of random starts
  - Easy to do!
  - May not be that valuable
- Parallelize on the number of clusters
  - Also easy to do!
  - Turn your brain off! Machine your learnings!!!

Measuring Cluster Performance

- Would like an analogue of classification accuracy
- How do we compare two sets of cluster labels?
- Rand Measure / Rand Index https://en.wikipedia.org/wiki/Rand_index
  - Given a set of n elements S and two partitions of S into r-length subsets $X = \{X_1, \ldots, X_r\}$ and $Y = \{Y_1, \ldots, Y_r\}$

$$R = \frac{a + b}{\binom{n}{2}}$$

    Where

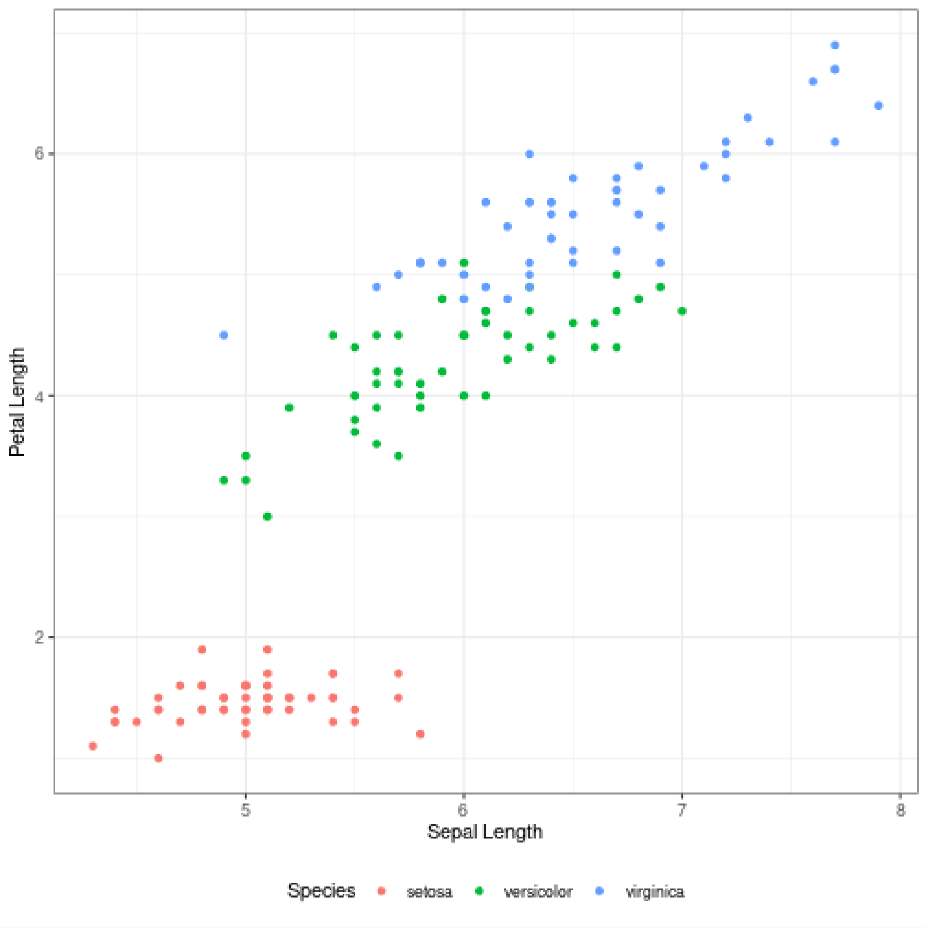
    - a is the # of pairs of elements of S that agree across X and Y
    - b is the # of pairs of elements of S that disagree across X and Y

Rand Measure

```r
rand_measure = function(l1, l2){
  n = length(l1)
  a = b = 0L

  for (i in 1:n){
    for (j in (i+1L):n){
      if (j > n) # R indexing is stupid
        break

      same1 = (l1[i] == l1[j])
      same2 = (l2[i] == l2[j])

      if (same1 && same2)
        a = a + 1L
      else if (!same1 && !same2)
        b = b + 1L
    }
  }

  (a + b) / choose(n, 2)
}
```

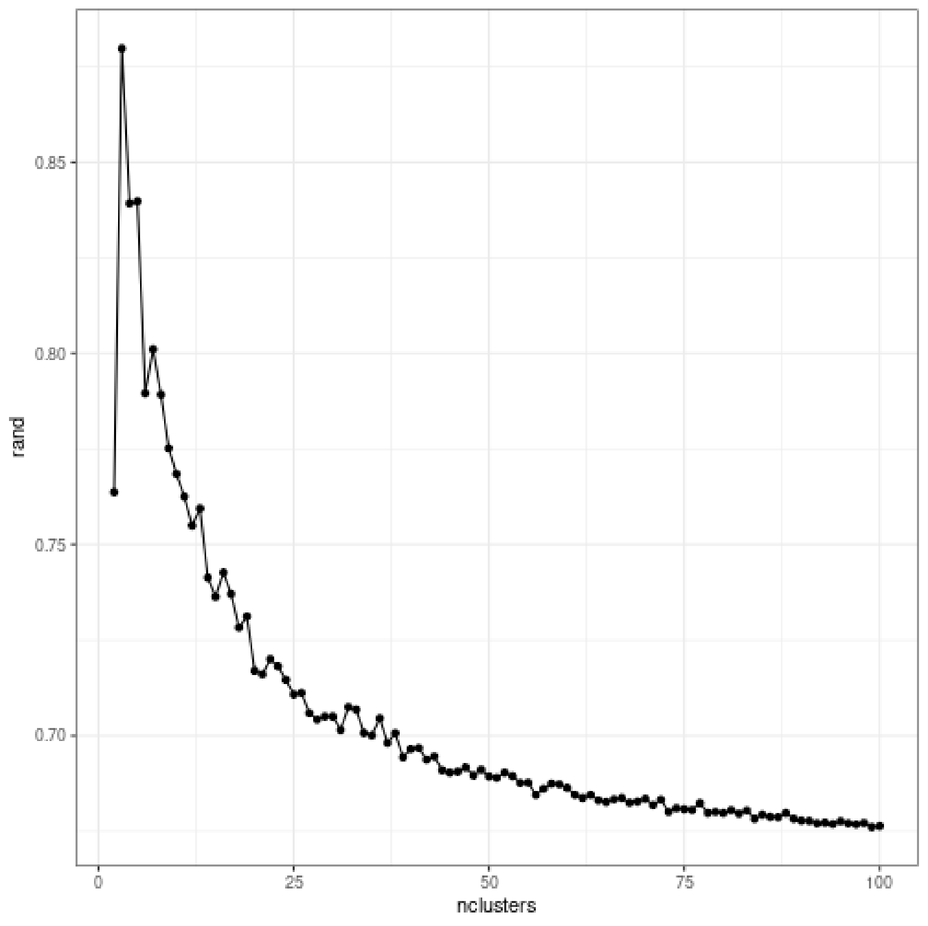
Using the Iris Dataset

- Fisher's famous iris dataset
- Required by law in clustering lectures



Finding the k in k-Means

```r
library(parallel)
labels = as.numeric(iris[, 5])
x = iris[, -5]

options(mc.cores=8)
nclusters = 2:100
clusters = mclapply(
  nclusters,
  function(i) kmeans(x, i, nstart=10)$clu
)
rand = mclapply(
  clusters,
  rand_measure,
  l2 = labels
) |> unlist()
```



Wrapup

Scalability?

- "scale up"
- "scale out"
- What do these mysterious phrases mean?

Vertical (up) vs Horizontal (out) Scaling

- Industry jargon
- (Opinion) Not really that well-defined
- More about *hardware* than *applications*
- Example
  - *Scale up* - Get a better processor
  - *Scale out* - Get multiple nodes
- Have we seen either so far?

Wrapup

- Be careful with RNG's in parallel!
- Where to look for (task) parallelism
    - Independent simulations
    - Multiple model fitting
    - *Many* ETL operations
- "Workflow parallelism" is *ubiquitous* in data/analysis pipelines
- Next time: data parallelism

Questions?