Introduction to parallel computing with R

Thomas Hauser
Director of Research Computing
University of Colorado Boulder
https://github.com/ResearchComputing/Parallelization_Workshop

Creating parallel-ready R code

- Start with serial implementation
- Use function that are “parallel-friendly”
  - Use the apply family of functions
    - base::apply: Apply Functions Over Array Margins
    - base::by: Apply a Function to a Data Frame Split by Factors
    - base::eapply: Apply a Function Over Values in an Environment
    - base::lapply: Apply a Function over a List or Vector
    - base::mapply: Apply a Function to Multiple List or Vector Arguments
    - base::rapply: Recursively Apply a Function to a List
    - base::tapply: Apply a Function Over a Ragged Array
  - Easy to use parallel equivalents that will take care of a lot of things behind the scenes

Package parallel

- Based on multicore and snow
- Includes a parallel random number generator
- Supports single program multiple data paradigm (SPMD)
- Main interface is parallel version of lapply and similar
- Can use cores on a single machine (multicore) or several machines using MPI
- MPI support depends on the Rmpi package

Introduction to Parallel

- Scatter/gather paradigm
  - Scatter: breaking work into chunks and then distributes it to workers
  - Computation on a chunk
  - Gather: manager receives results and combines them to solution

Package: parallel

- Startup and stop
  - `makeCluster`: create a parallel cluster
  - # one or more parallel calls
  - `stopCluster`: stop a cluster
- Running a function on the cluster
  - clusterCall: calls a function fun with identical arguments ... on each node.
  - mclapply(x, function, …, mc.cores
    - All objects and packages are automatically available
  - parLapply(cl, x, function, …)
    - Have to explicitly export objects to the cluster
      - clusterExport
      - clusterEvalQ

Hello World – parallel version

- Run the following versions

```
$ sinteractive --partition=shas --qos=debug \
--time=30:00 --ntasks=24 --nodes=1 \
--reservation=parallelD4
$ module purge
$ module load R
$ cd $HOME/Parallelization_Workshop/Day4-
Parallel_R/examples/parallel
$ Rscript hello_parallel.R
```

Sum the rows of a matrix

```
M <- matrix(1:, nrow = 3)
for (i in 1:3) {
  print(sum(M[i, ]))
}
```

- Order of loop is irrelevant
- This is a parallelizable algorithm

Functional programming in R

- Apply

```r
M <- matrix(1:9, nrow = 3)
apply(M, 1, sum)
```

- `[1] 12 15 18`

- Functional programming in R
- Easier to parallelize
  - There are similar construct that work in parallel
- Independent parallel

Package parallel

- parApply: very similar to apply
- Simple approach to parallelization

Parallel version

```
library(parallel)
library(microbenchmark)
ncores = detectCores()
print(ncores)
cl <- makeCluster(ncores)
nrows <- 1000
M <- matrix(1:nrows^2, nrow = nrows, ncol =
nrows)
microbenchmark(parApply(M, 1, sum, cl=cl),
times=10)
stopCluster(cl)
```

Exercise: Parallel sum_rows

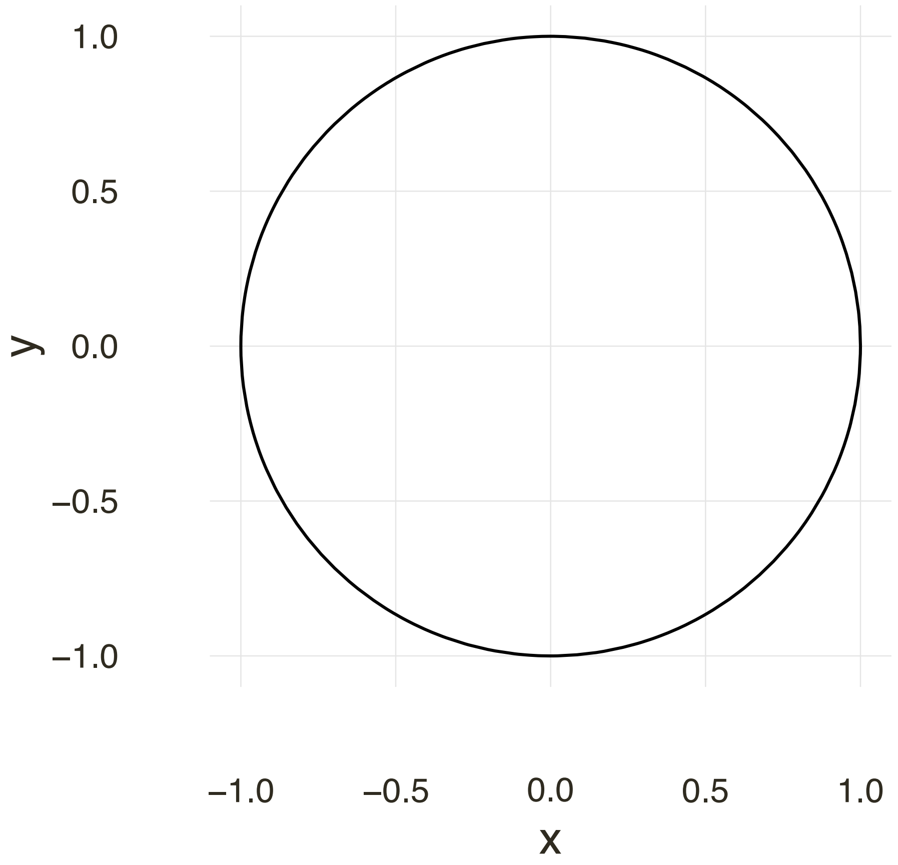
- Get a new node

```
$ Rscript sum_rows_apply_parallel.R
```

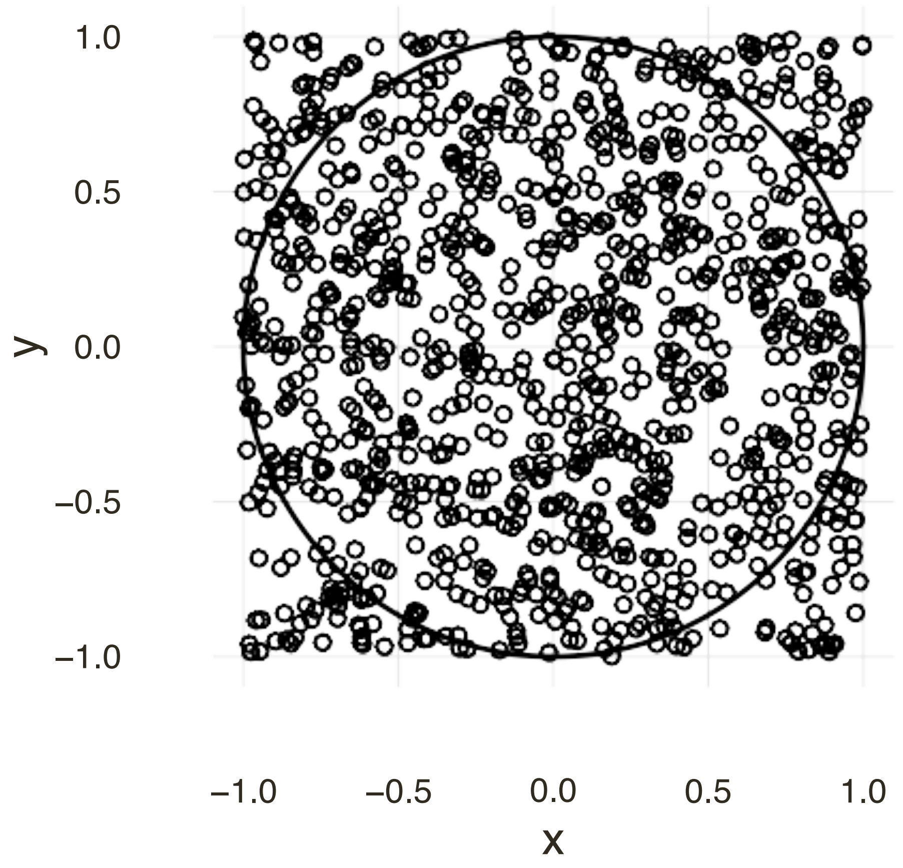
- Use a different number of cores
  - 4, 8, 12

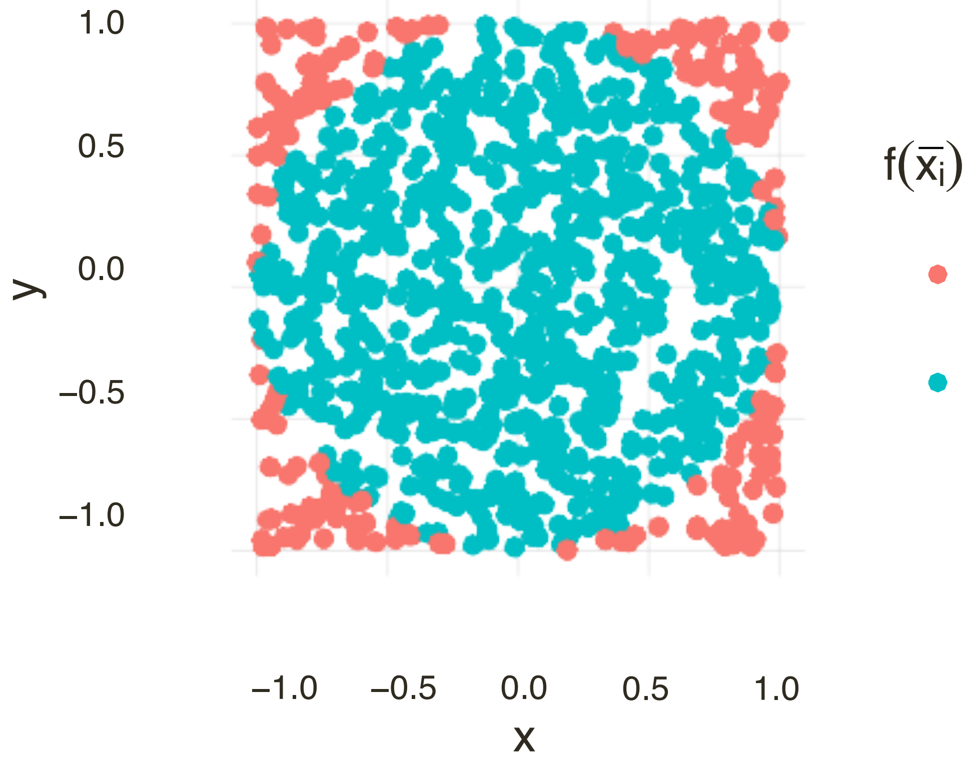
Calculate PI with Monte Carlo

- Goal: estimate the area of a circle with radius = 1 and area = π using Monte Carlo integration.



Monte Carlo Integration





Monte Carlo with R

```r
approx_pi <- function(n) {
  # estimate pi w/ MC integration
  x <- runif(n, min = -1, max = 1)
  y <- runif(n, min = -1, max = 1)
  V <- 4
  f_hat <- ifelse(x^2 + y^2 <= 1, 1, 0)
  V * sum(f_hat) / n
}
```

`apply()` for vectors

`sapply()` returns vectors, matrices, and arrays

```
pi_hat <- sapply(n, approx_pi)
str(pi_hat)
```

```
 num [1:1000] 4 2.8 3.2 2.7 3.04 ...
```

Local parallelization

- Each MC integration is embarrassingly parallel!
- To parallelize:
  - start a cluster
  - compute simultaneously across the cluster
  - gather results
  - close cluster

the `parallel` package

```
cl <- makeCluster(2)

pi_hat <- parSapply(cl, n, approx_pi)

stopCluster(cl)
```

Parallel Random Numbers

```
> RNGkind("L'Ecuyer-CMRG")
> set.seed(1.0)
```

- Creates independent streams for each process

Exercise - pi_parallel.R

- Is the program using the weak scaling or strong scaling approach?
- Modify the program so that it uses the other approach.
- Run the program on your own node using
  - 4, 16 and 24 cores

Questions?

- Email rc-help@colorado.edu
- Twitter: CUBoulderRC

- Link to survey on this topic:

http://tinyurl.com/curc-survey16

- Slides: https://github.com/ResearchComputing/Parallelization_Workshop

License