Efficient R

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

Outline

- Measuring performance
- Improving performance
- Code organization
- Look for solutions that exist
- Do as little as possible
- Vectorize
- Avoid copies
- Case study

Timing your R code

```
> install.packages("microbenchmark")
```

```
> library("microbenchmark")
> mean1 <- function(x) mean(x)
> mean2 <- function(x) sum(x) / length(x)
> microbenchmark(mean1(1000), mean2(1000))
```

- See `Day4-Parallel-R/examples/serial/simple_timing.R`

Exercise: Run simple_timing.R

```
$ 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/serial
$ Rscript simple_timing.R
```

- What's your run time for this simple example?

Exercise: Simple Profiling

- File to profile:

  `Day4-Parallel_R/examples/serial/simple_profiling.R`

  - Simple example of calling several functions
- Create a new file to run the example

  `profile_simple_profiling.R`

  ```
  Rscript profile_simple_profiling.R
  ```

- Can you interpret the output?

Profiling your R code

```
> install.packages("profvis")

> p  <-  profvis({f()})
> htmlwidgets::saveWidget(p, 'test.html', selfcontained = FALSE)
```

- Best run out of Rstudio
- No information about C code

Making your code faster

- Reuse by looking for existing solutions
- Do less work
- Vectorize
- Avoid copies
- Byte-code compile
- Parallelize – after lunch

Pitfalls

- Writing fast but incorrect code
  - Do unit testing or other testing of your code
  - RUnit package
- Write code you think is faster, but is actually not better
  - Keep a record
    - Failures should be documented too
- Use the microbenchmark package to compare different solution
- Define target speed
  - Don’t over optimize
- Has someone solved this problem already

Do as little as possible

- `rowSums(), colSums(). rowMeans(), colMeans()`
  - Faster then equivalent invocations of apply()
- vapply() is faster than sapply()
- Any(x == 10) is faster than 10 %in% x
- Benchmark
- Example: sum_rows.R
  - Improve the performance and measure it

Vectorize

- Take a “whole object” approach
  - Operations on the entire vector instead of the individual elements
- Vectorized function
  - Makes it simpler - operation on an object
  - Loops of the vectorized functions are generally written in C
    - Faster because of lower overhead
  - Example: sum_rows_vector.R

Exercise: sum_rows

- Improve the performance of the function below

```
sumrows1 <- function(M, nrows) {
    s <- c(nrows)
   for (i in 1:nrows)
       s[i] = sum(M[i, ])
     return(s)
     }
```

Solution – sum_rows

- sum_rows_vector.R

Be careful with memory

- Avoid copies
  - > c(), append(), cbind(), rbind(), or paste() to grow object
  - R needs to allocate space for new object
  - Then copy old object to new space
- Example combining strings

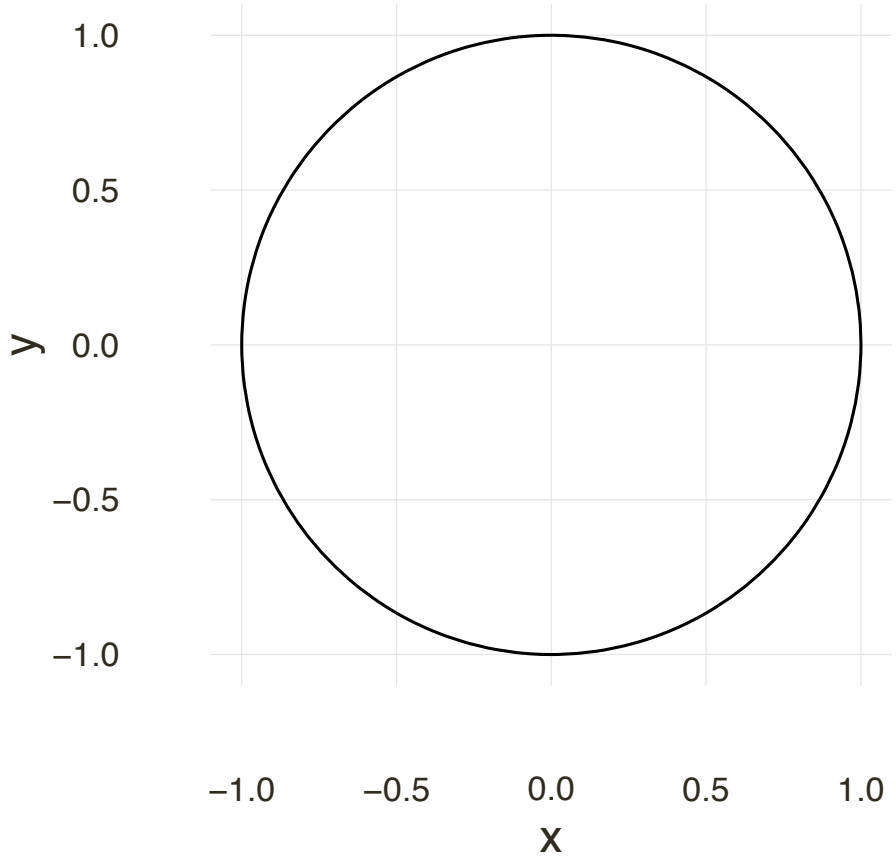
Byte code

- Using the R byte compiler
- May not work for your function
- Easy to try
- Myfunc
- Myfunc_c <- compiler::cmpfun(myfunc)

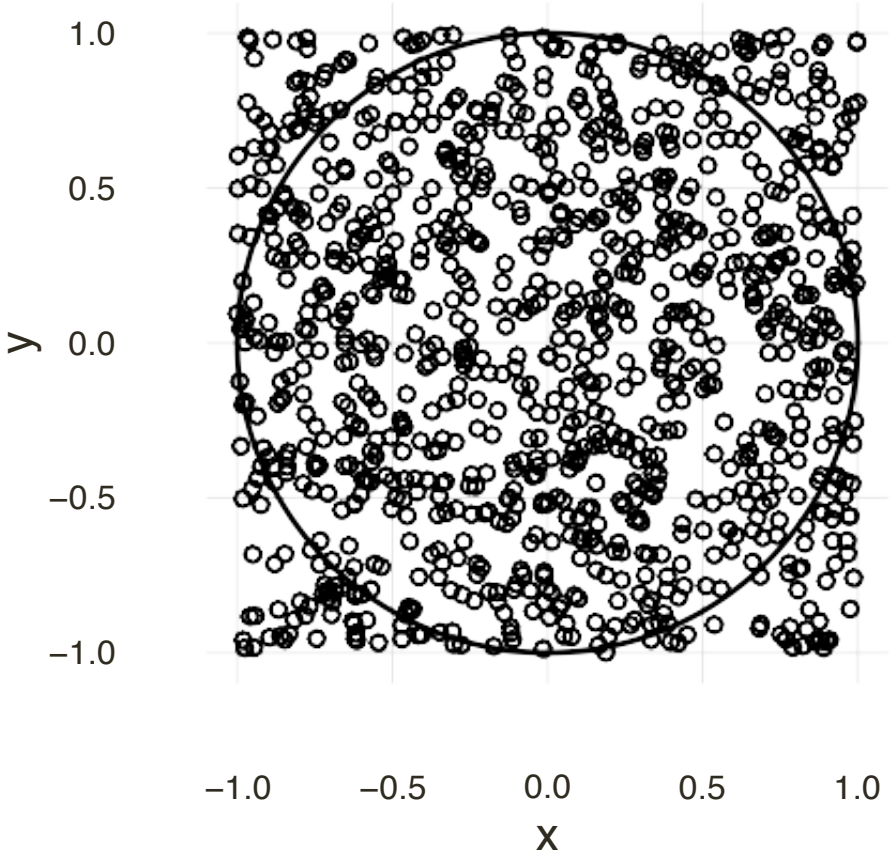
- Microbenchmark()

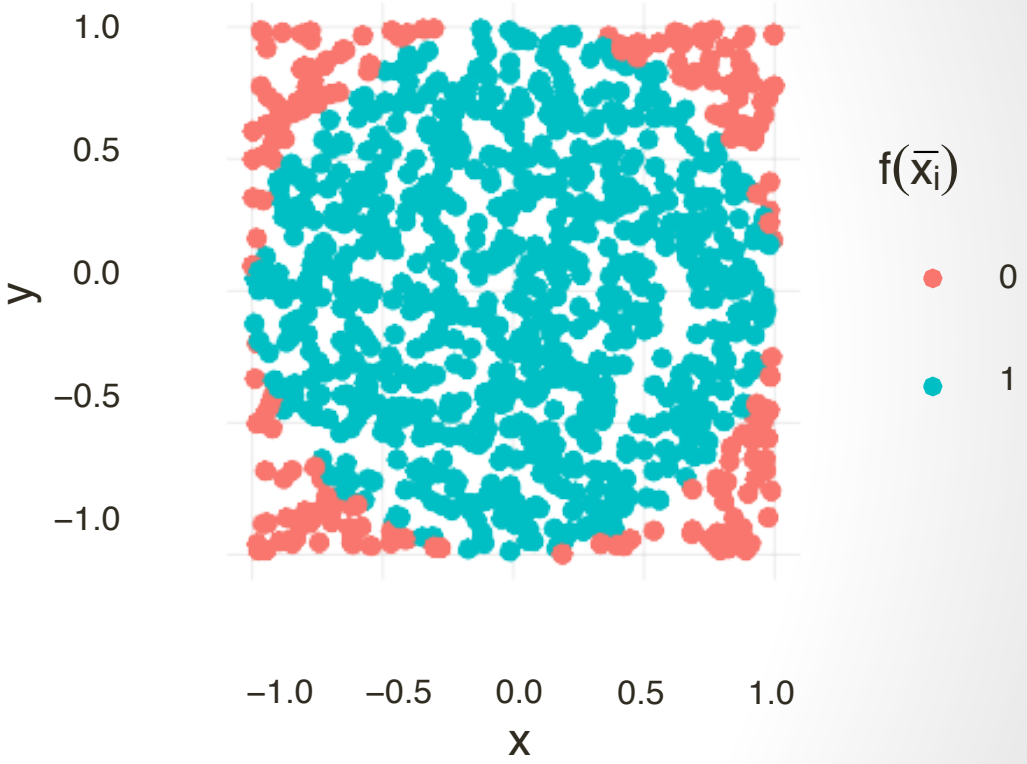
Calculate PI with Monte Carlo

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



Monte Carlo Integration

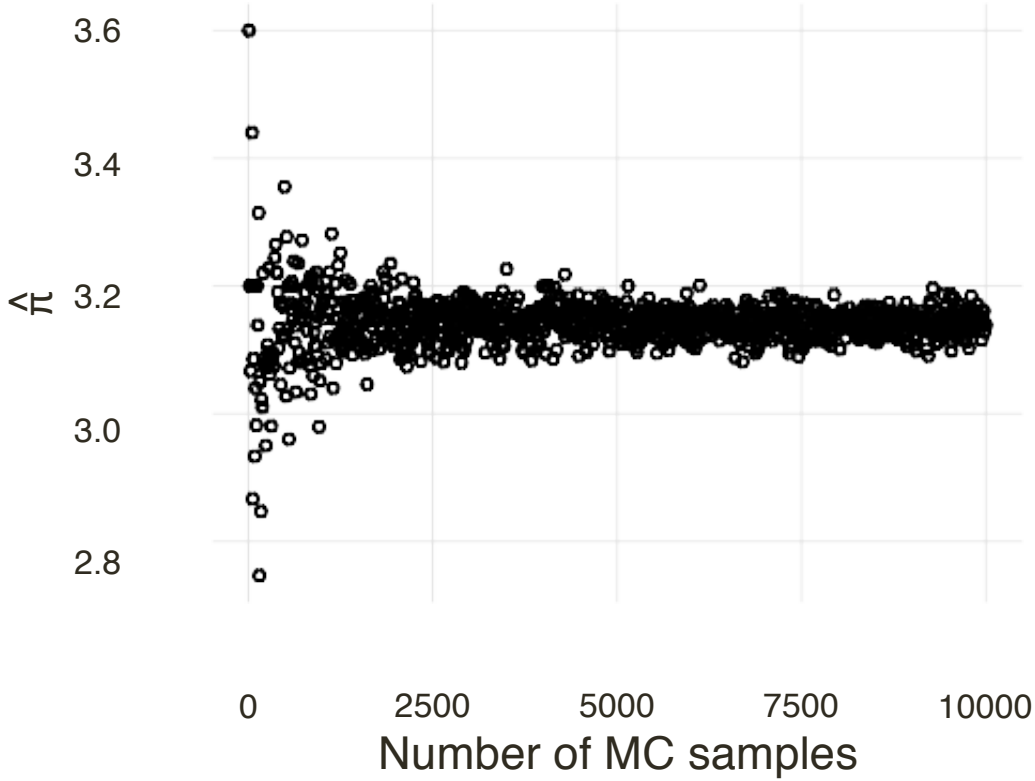




Monte Carlo with R

```
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
}
```

How does N influence $\hat{\pi}$?



Avoiding a for-loop

`lapply()` returns a list

```
pi_hat <- lapply(n, approx_pi)
str(pi_hat[1:5])
```

```
List of 5
 $ : num 2.4
 $ : num 2.4
 $ : num 3.33
 $ : num 3
 $ : num 3.28
```

`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 ...
```

Exercise: Run pi_serial.R

- Run the program
- Create a version that uses the byte compile

```
myfunc_c <- compiler::cmpfun(myfunc)
```

- Does the program run faster?
- Why or why not?
- Solution: pi_compiled.R

Questions?

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

- Link to survey on this topic:

http://tinyurl.com/curc-survey16

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

License