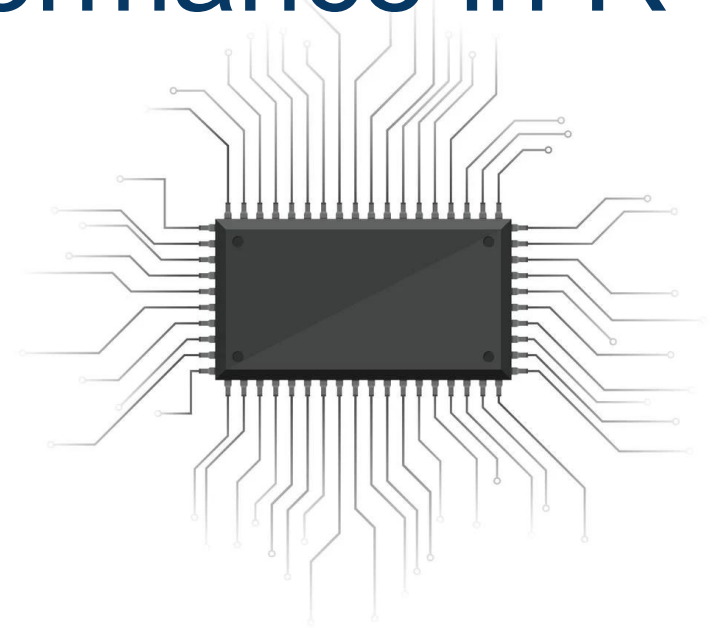


Big Data Ecology

Efficiency and performance in R

Christian König

Ecology and Macroecology Lab
Institute for Biochemistry and Biology
University of Potsdam



language design

“To understand computations in R, two slogans are helpful: Everything that exists is an object. Everything that happens is a function call.”

John M. Chambers



R is a multi-paradigm language:

- **Object-oriented**

- Everything is an object
- Objects are defined by a class and methods

- **Functional**

- Functions are “first-class citizens” that can be assigned to variables, passed as arguments, or even returned by a function
- Problem solving is centered around functions

- **Dynamic**

- Variable types can change
- No strict scoping rules
- Code is interpreted, not compiled

language design

- The design of the R language makes it extremely flexible and interactive
- This flexibility and ease of use may come at the cost of performance

2014

	Mac		
Language	Version/Compiler	Time	Rel. Time
C++	GCC-4.9.0	0.73	1.00
	Intel C++ 14.0.3	1.00	1.38
	Clang 5.1	1.00	1.38
Fortran	GCC-4.9.0	0.76	1.05
	Intel Fortran 14.0.3	0.95	1.30
Java	JDK8u5	1.95	2.69
Julia	0.2.1	1.92	2.64
Matlab	2014a	7.91	10.88
Python	Pypy 2.2.1	31.90	43.86
	CPython 2.7.6	195.87	269.31
R	3.1.1, compiled	204.34	280.90
	3.1.1, script	345.55	475.10
Mathematica	9.0, base	588.57	809.22



2018

	Mac		
Language	Version/Compiler	Time	Rel. Time
C++	GCC-7.3.0	1.60	1.00
	Intel C++ 18.0.2	1.67	1.04
	Clang 5.1	1.64	1.03
Fortran	GCC-7.3.0	1.61	1.01
	Intel Fortran 18.0.2	1.74	1.09
Java	9.0.4	3.20	2.00
Julia	0.7.0	2.35	1.47
	0.7.0, fast	2.14	1.34
Matlab	2018a	4.80	3.00
Python	CPython 2.7.14	145.27	90.79
	CPython 3.6.4	166.75	104.22
R	3.4.3	57.06	35.66
Mathematica	11.3.0, base	1634.94	1021.84

General tips - Vectorization

- Vectors are the basic data type in R
- Many operations and function in R work naturally and intuitively with vectors:

```
> x = 0:9 # create a vector from 0 to 9
> x + 1    # vector-scalar addition
[1] 1 2 3 4 5 6 7 8 9 10
> x + x    # vector-vector addition
[1] 0 2 4 6 8 10 12 14 16 18
```

! highly
optimized

- [Apply](#) functions (including your own!) to each row/columns of a matrix (`apply`) or each element of a list (`lapply`, `sapply`) or vector (`sapply`, `vapply`)

```
> sapply(x, function(x_i){x_i + 1})
[1] 1 2 3 4 5 6 7 8 9 10
```

! safer, but usually not
faster than loops

General tips - Memory-Management

- Per default, **R**
 - stores objects in memory
 - copies objects when you add/remove elements
 - copies objects when another object points to them

→ Avoid growing objects iteratively (e.g. in for-loops)

```
> x = c()  
> for(i in 1:10){x = c(x, i)} # creates a new copy of x every iteration  
> x  
[1] 1 2 3 4 5 6 7 8 9 10
```

→ If necessary, pre-allocate memory and modify object in-place

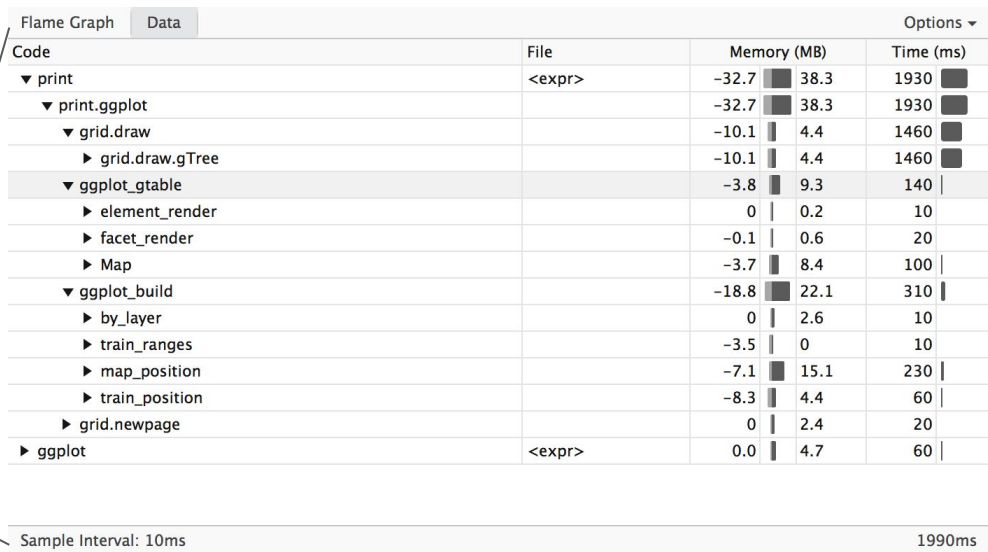
```
> x = vector(mode = "numeric", length = 10)  
> for(i in 1:10){x[i] = i} # modifies elements of x in place  
> x  
[1] 1 2 3 4 5 6 7 8 9 10
```

General tips - Use existing digital infrastructure

- There is an R-package for almost everything!
 - Currently there are almost 18000 R-packages in CRAN, many additional packages are hosted on Github
 - The existence of a package means someone has spent **thought, effort and time** on a problem that you don't have to spend (acknowledge that by citing!)
 - Packages are often implemented in C and thus offer superior performance over self-written solutions
- Outsource computations to more efficient tools (e.g. when connected to a database or GIS)
- Check for existing solutions on Stackoverflow, Github, Blogs, etc.

Profiling & benchmarking

- To optimize your code, you need to understand which parts are performing poorly
- Timing & Benchmarking:
 - `bench::mark()`
 - `System.time()`
- Profiling:
 - `utils::Rprof()`
 - `profvis::profvis()`



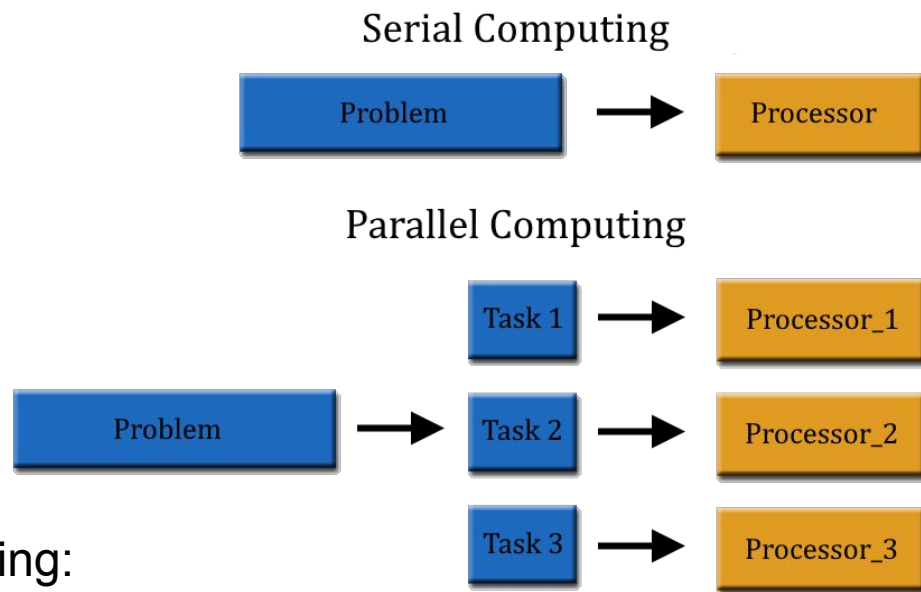
Code	File	Memory (MB)	Time (ms)	Options
▼ print	<expr>	-32.7 38.3	1930	
▼ print.ggplot		-32.7 38.3	1930	
▼ grid.draw		-10.1 4.4	1460	
▶ grid.draw.gTree		-10.1 4.4	1460	
▼ ggplot_gtable		-3.8 9.3	140	
▶ element_render		0 0.2	10	
▶ facet_render		-0.1 0.6	20	
▶ Map		-3.7 8.4	100	
▼ ggplot_build		-18.8 22.1	310	
▶ by_layer		0 2.6	10	
▶ train_ranges		-3.5 0	10	
▶ map_position		-7.1 15.1	230	
▶ train_position		-8.3 4.4	60	
▶ grid.newpage		0 2.4	20	
▶ ggplot	<expr>	0.0 4.7	60	

Sample Interval: 10ms

1990ms

Parallel processing

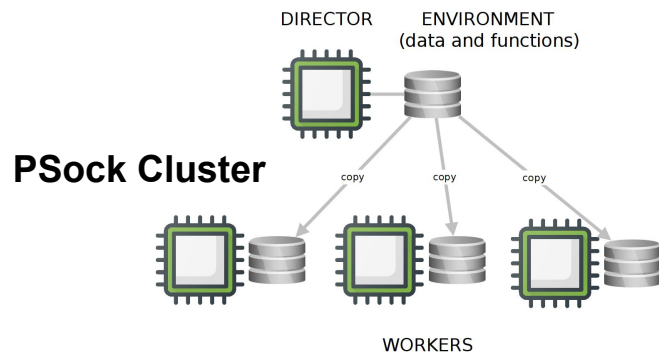
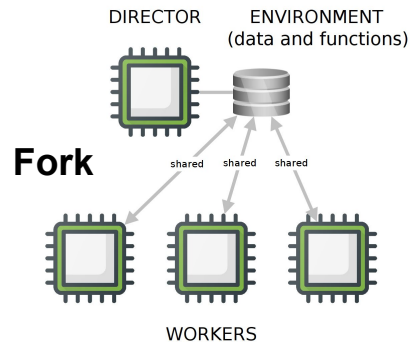
- Split up a complex problem into smaller tasks
- Solve these tasks separately (and simultaneously!) on different processors
- Core R packages for parallel processing:
 - `parallel`: parallel apply family functions, register compute Clusters
 - `foreach` + `doParallel`: parallel for-loops



Source:
<https://www.teldat.com/blog/en/parallel-computing-bit-instruction-task-level-parallelism-multicore-computers/>

Parallel processing

- Parallel workers (CPUs) need to be organized in a parallel backend
- **Fork:**
 - workers have a shared environment (functions, objects, packages, etc.) → Faster
 - works only on local machines
 - works only on machines with UNIX-style OS
- **Parallel Socket Cluster:**
 - workers get their own copy of the environment
 - scales easily, works with distributed CPUs
 - works on UNIX and Windows machines



It's not all about computation!

- Keep an eye on the bigger picture
 - Which parts of your code are crucial for performance?
 - What's your ratio of coding time vs. runtime?
- Know your IDE
 - Version control
 - Debugger
 - Advanced editing features
 - Manage multiple processes
 - Access to OS terminal



"The real problem is that programmers have spent far too much time worrying about efficiency in the wrong places and at the wrong times; premature optimization is the root of all evil (or at least most of it) in programming."

Donald Knuth

It's not all about computation!

- Organize your work
 - Plan your project
 - Split code into separate files, e.g. `data_preparation.R`, `analysis.R`, ...
 - Organize code into logical units within files
- Write readable, well-documented code
 - Use consistent naming conventions
 - Comment your code liberally but precisely
 - Don't repeat yourself

Practical

Work through the R practical `performance.Rmd`
(30 min)

Further readings

E-books and tutorials:

<https://csgillespie.github.io/efficientR/index.html>

<https://adv-r.hadley.nz/techniques.html>

<https://data-flair.training/blogs/r-performance-tuning-techniques/>

Parallel computing:

https://psu-psychology.github.io/r-bootcamp-2018/talks/parallel_r.html

https://www.blasbenito.com/post/02_parallelizing_loops_with_r/

Publications:

Morandat, F. et al. 2012. Evaluating the Design of the R Language (J Noble, Ed.). - ECOOP 2012 – Object-Oriented Programming: 104–131.

Aruoba, S. B. and Fernández-Villaverde, J. 2014. A Comparison of Programming Languages in Economics. National Bureau of Economic Research