V XORNADA DE USUARIOS DE R EN GALICIA

FERRAMENTAS PARA REDUCIR O TEMPO DE EXECUCIÓN EN R

Alejandra López

25 de outubro do 2018

Universidade de Santiago de Compostela

MOTIVACIÓN

¿Es R lento?

Velocidad en la ejecución
vs
Velocidad de programación

ÍNDICE

Movimiento Browniano o proceso de Wiener

Dado un espacio de probabilidad $(\Omega, \mathcal{A}, \mathbb{P})$ y una filtración $\mathcal{F}_t$ con $t \geq 0$, el proceso $W(t)$ es un proceso de Wiener respecto $\mathcal{F}_t$ si verifica:

i. $W(t)$ es $\mathcal{F}_t$-medible para todo t.

ii. $W(0) = 0$ casi seguro.

iii. Los incrementos, $W(t) - W(s)$ con $s < t$, tienen distribución normal con $\mathbb{E}\{W(t) - W(s)\} = 0$ y $\mathbb{V}\text{ar}\{W(t) - W(s)\} = t - s$.

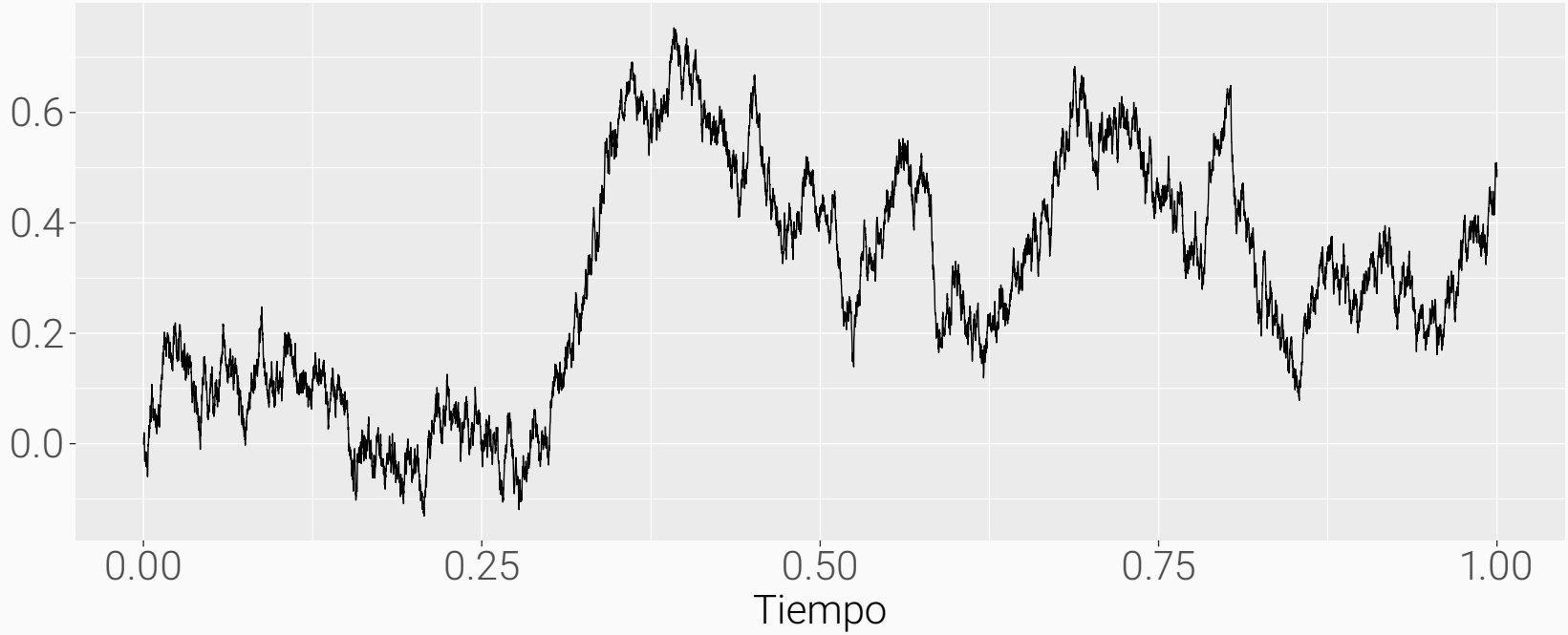
iv. $W(t) - W(s)$ es independiente de $\mathcal{F}_s$ para todo $s < t$.

v. El proceso $W(t)$ tiene trayectorias continuas.

EJEMPLO

Para simular trayectorias de un proceso de Wiener estándar dividimos el intervalo $[0, N]$ en $0 = t_0 < \cdots < t_n = N$ puntos con $t_i - t_{i-1} = \Delta$. Tomamos $W(0) = W(t_0) = 0$, fijamos $i = 1$ e iteramos el algoritmo

1. Generar $\varepsilon_{t_i} \sim N(0, 1)$.
2. $W(t_i) = W(t_{i-1}) + \varepsilon_{t_i} \cdot \sqrt{\Delta}$.
3. i = i + 1.
4. Volver a 1 mientras $i \leq n$.



EJEMPLO

Podemos implementar el algoritmo en R utilizando un bucle `for`.

```
rWiener <- function(n, N = 1) {
  Delta <- N/n
  W <- numeric(n + 1)
  for (i in 2:(n + 1))
    W[i] <- W[i - 1] + rnorm(1) * sqrt(Delta)
  return(W)
}

set.seed(987)
system.time(
  W <- rWiener(n = 1e8)
)
R>    user  system elapsed
R> 352.84   10.83  377.24
```

COMPILACIÓN A BYTECODE

El paquete **compiler** permite compilar funciones de R, de modo que obtenemos una versión en *bytecode* que puede ejecutarse más rápido.

```
library(compiler)
rWiener_comp <- cmpfun(rWiener)
# enableJIT(3)

set.seed(987)
system.time(
  W1 <- rWiener_comp(n = 1e8)
)
R>    user  system elapsed
R> 154.07    1.70  179.86

all.equal(W, W1)
R> [1] TRUE
```

VECTORIZACIÓN

Las funciones anteriores son ineficientes, ya que no es necesario utilizar un bucle `for`, podemos simular la trayectoria del proceso con una suma acumulada:

```
rWiener_vec <- function(n, N = 1) {
  Delta <- N/n
  W <- c(0, cumsum(sqrt(Delta) * rnorm(n)))
  return(W)
}

set.seed(987)
system.time(
  W2 <- rWiener_vec(n = 1e8)
)
R> user  system elapsed
R> 6.39    0.91   10.44

all.equal(W, W2)
R> [1] TRUE
```

Lenguajes compilados

LENGUAJES COMPILADOS

Existen funciones que requieren bucles explícitos y no pueden ser vectorizadas, lo que implica que su ejecución puede ser lenta. Una solución es recurrir a lenguajes de implementación compilada, como C, Fortran o C++. Las funciones `.C()`, `.Call()`, `.Fortran()` o `.External()` permiten llamar a código compilado.

- Las funciones `.C()` y `.Fortran()` son comúnmente utilizadas para rutinas numéricas. Como desventaja destaca que solo podemos pasar ciertos tipos de datos y debemos convertir los datos en el código interpretado.
- Las funciones `.Call()` y `.External()` permiten manipular objetos de R en C, pero su uso es más complicado que la interfaz `.C()`.

Interfaz .C

Para llamar a una función en C desde R se requieren dos propiedades:

1. La función en C no devuelve un valor.
2. Todos los argumentos son punteros.

```
#include <R.h>
#include <Rmath.h>

void rWiener_C(int *n_, double *Delta_, double *W)
{
  int n = n_[0];
  double Delta = Delta_[0];

  W[0] = 0;
  GetRNGstate();

  for (int i = 1; i < (n + 1); i++)
    W[i] = W[i - 1] + rnorm(0, 1) * sqrt(Delta);

  PutRNGstate();
}
```

Interfaz .C

En R:

```
dyn.load('rWiener_C.dll')
rWiener_C <- function(n, N = 1) {
  .C("rWiener_C", n     = as.integer(n),
                  Delta = as.double(N/n),
                  W     = double(n + 1))$W
}

set.seed(987)
system.time(
  W3 <- rWiener_C(n = 1e8)
)
R> user  system elapsed
R> 6.29    0.18    7.50

all.equal(W, W3)
R> [1] TRUE
```

Interfaz `.Call`

La interfaz `.Call` permite manipular objetos de R en C.

```
#include <R.h>
#include <Rdefines.h>
#include <Rinternals.h>
#include <Rmath.h>

SEXP rWiener_Call(SEXP n_, SEXP Delta_)
{
  int n;
  double Delta, *pW;
  SEXP W;

  n     = *INTEGER(n_);
  Delta = *REAL(Delta_);

  PROTECT(W = NEW_NUMERIC(n + 1));
  pW = NUMERIC_POINTER(W);

  pW[0] = 0;
  GetRNGstate();
  for (int i = 1; i < (n + 1); i++)
    pW[i] = pW[i - 1] + rnorm(0, 1) * sqrt(Delta);
  PutRNGstate();

  UNPROTECT(1);

  return W;
}
```

Interfaz .Call

En R:

```
dyn.load('rWiener_Call.dll')
rWiener_Call <- function(n, N = 1) {
  .Call("rWiener_Call", n = as.integer(n),
                        Delta = as.double(N/n))
}

set.seed(987)
system.time(
  W4 <- rWiener_Call(n = 1e8)
)
R> user  system elapsed
R> 6.11    0.11    7.06

all.equal(W, W4)
R> [1] TRUE
```

Paquete Rcpp

El paquete **Rcpp** proporciona clases en C++ que facilitan la interacción con código en C o C++ utilizando la interfaz `.Call()` de R. Entre otras ventajas, permite trabajar en C++ con objetos de R a través de funciones wrapper.

```
library(Rcpp)
sourceCpp("rWiener_Cpp.cpp")

set.seed(987)
system.time(
  W5 <- rWiener_Cpp(n = 1e8)
)
R> user  system elapsed
R> 5.95    0.33    6.77

all.equal(W, W5)
R> [1] TRUE
```

Eddelbuettel, D. y Balamuta, J. J. (2017). Extending R with C++: A Brief Introduction to Rcpp. PeerJ Preprints, 5:e3188v1.

Paquete Rcpp

Utilizando Rcpp sugar:

```
#include <Rcpp.h>

using namespace Rcpp;

// [[Rcpp::export]]
NumericVector rWiener_Cpp(const int n, const double Delta)
{
  Rcpp::NumericVector W(n + 1);
  Rcpp::NumericVector x(n);
  double suma = 0;

  W[0] = 0;
  x = rnorm(n) * sqrt(Delta);

  for (int i = 1; i < (n + 1); i++)
  {
    suma += x[i];
    W[i] = suma;
  }

  return W;
}
```

BENCHMARK

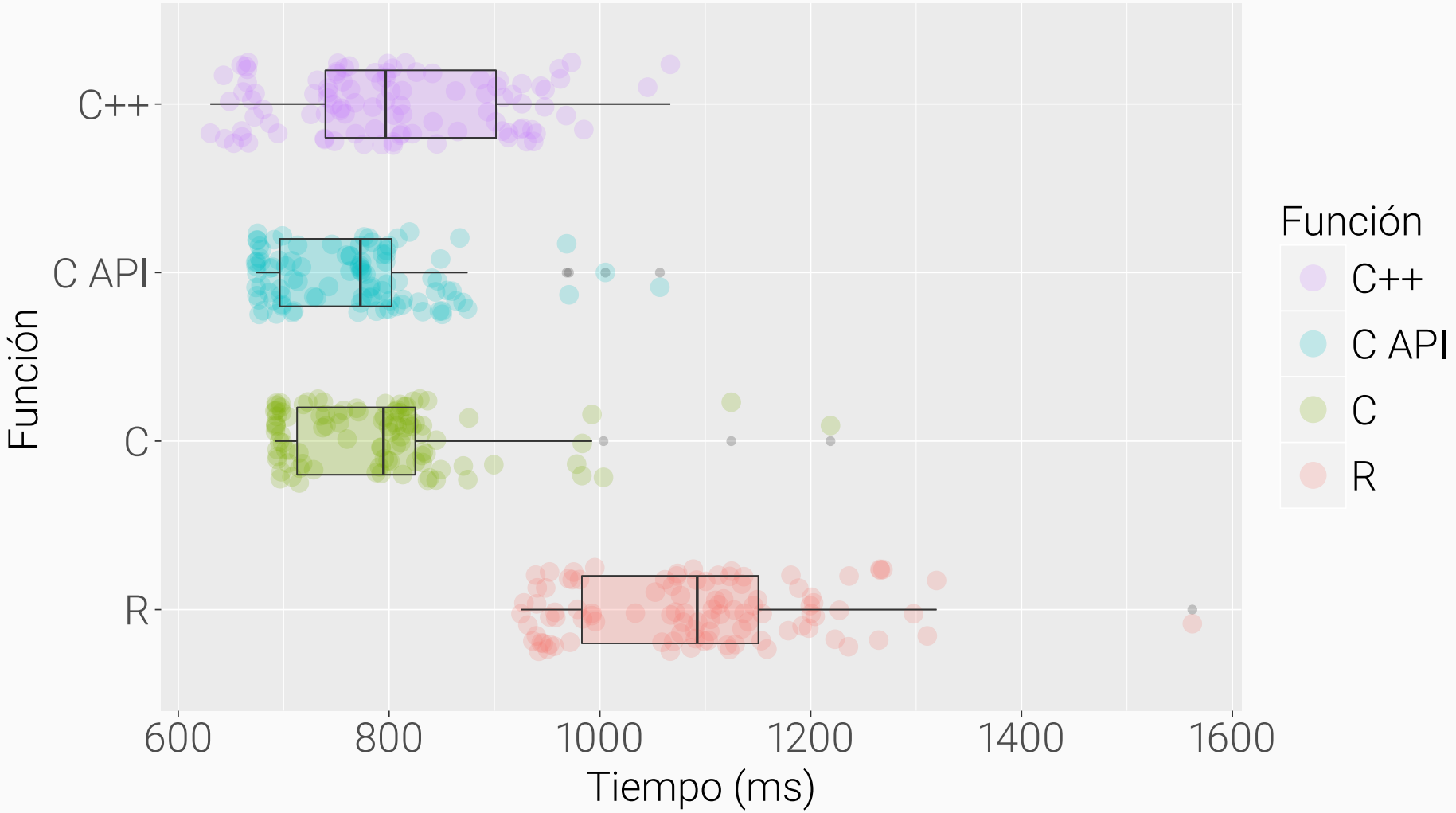


Figura 2: Comparativa del tiempo de ejecución ($n = 10^7$ y 100 simulaciones).

BENCHMARK

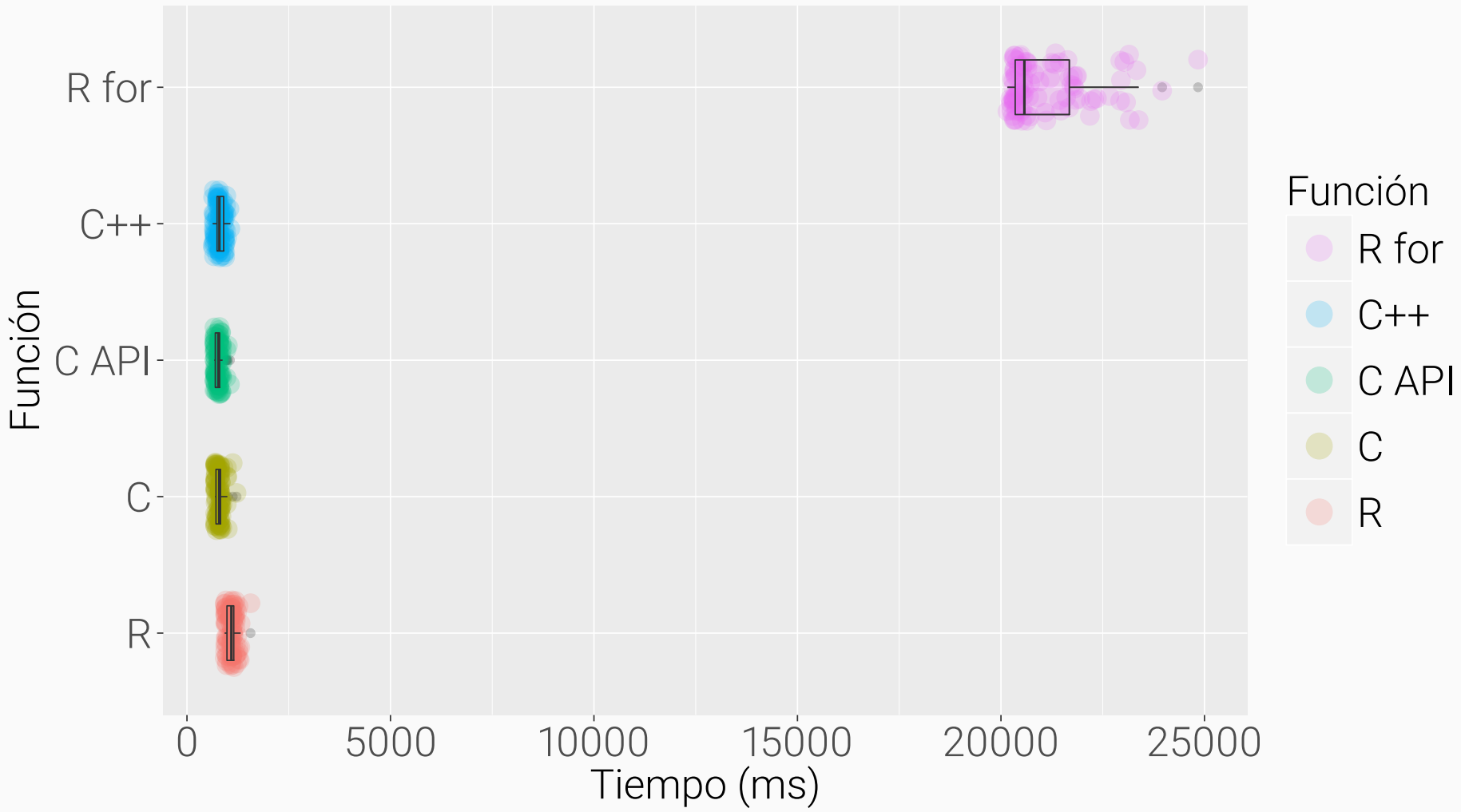


Figura 3: Comparativa del tiempo de ejecución ($n = 10^7$ y 100 simulaciones).

Programación en la GPU

Paquete gpuR

Las *unidades de procesamiento gráfico* o GPUs (*Graphic Processing Units*) son coprocesadores dedicados al procesamiento de gráficos u operaciones de coma flotante. En la actualidad son utilizadas también para cálculos computacionalmente intensivos.

El paquete `gpuR` permite realizar cálculos en la GPU desde R. Aunque existen otros paquetes que permiten hacer uso de la capacidad de la GPU (`gputools`, `cudaBayesreg`, `HiPLARM`, `HiPLARb` o `gmatrix`), están limitados a las GPU de NVIDIA.

Rupp, K., Tillet, P., Rudolf, F., Weinbub, J., Grasser, T., y Jüngel, A. (2016). ViennaCL - Linear algebra library for multi- and many-core architectures. SIAM Journal on Scientific Computing.

Paquete gpuR

```
library(gpuR)
# verificamos que tenemos una GPU válida
detectGPUs()
R> [1] 1

A <- matrix(rnorm(2^24), nrow = sqrt(2^24))
B <- matrix(rnorm(2^24), nrow = sqrt(2^24))
A_gpu <- gpuMatrix(A, type = "double")
B_gpu <- gpuMatrix(B, type = "double")
A_vcl <- vclMatrix(A, type = "double")
B_vcl <- vclMatrix(B, type = "double")

system.time(C <- A %*% B)[3]
R>  elapsed
R>    55.23
system.time(C_gpu <- A_gpu %*% B_gpu)[3]
R>  elapsed
R>    27.08
system.time(C_vcl <- A_vcl %*% B_vcl)[3]
R>  elapsed
R>     0.04
```

Asignación por referencia

Paquete `data.table`

Muchas de las funciones que utilizamos para leer archivos externos (por ejemplo `.csv`) o conectar con bases de datos, devuelven por defecto un objeto tipo `data.frame`.

El paquete `data.table` ofrece una versión mejorada de `data.frame` que permite manipular datos rápidamente. La manipulación de objetos `data.table` posee una sintaxis diferente:

```
DT[where, select|update|do, by]
```

Paquete data.table

```
library(data.table)
m <- matrix(1, nrow = 1e6L, ncol = 100L)
m_df <- as.data.frame(m)
m_dt <- as.data.table(m)

system.time(for (i in 1:1000) m[i, 1] <- i)[3]
R>  elapsed
R>    0.340
system.time(for (i in 1:1000) m_df[i, 1] <- i)[3]
R>  elapsed
R>   15.856
system.time(for (i in 1:1000) m_dt[i, V1 := i])[3]
R>  elapsed
R>    0.279
system.time(for (i in 1:1000) set(m_dt, i, 1, i))[3]
R>  elapsed
R>    0.001
```

Paralelización

PARALELIZACIÓN

Trabajando con R es habitual encontrarnos en situaciones donde repetimos una serie de cálculos varias veces. La mayoría de ordenadores actuales poseen procesadores multi core, por lo que si los cálculos no necesitan comunicarse entre ellos (son *embarrassingly parallel*) podemos ejecutarlos en paralelo y reducir el tiempo de ejecución.

Ejemplos de este tipo de problemas son:

- Simulaciones de modelos con diferentes parámetros
- MCMC
- Bootstrap, cross-validation

PARALELIZACIÓN: EJEMPLO

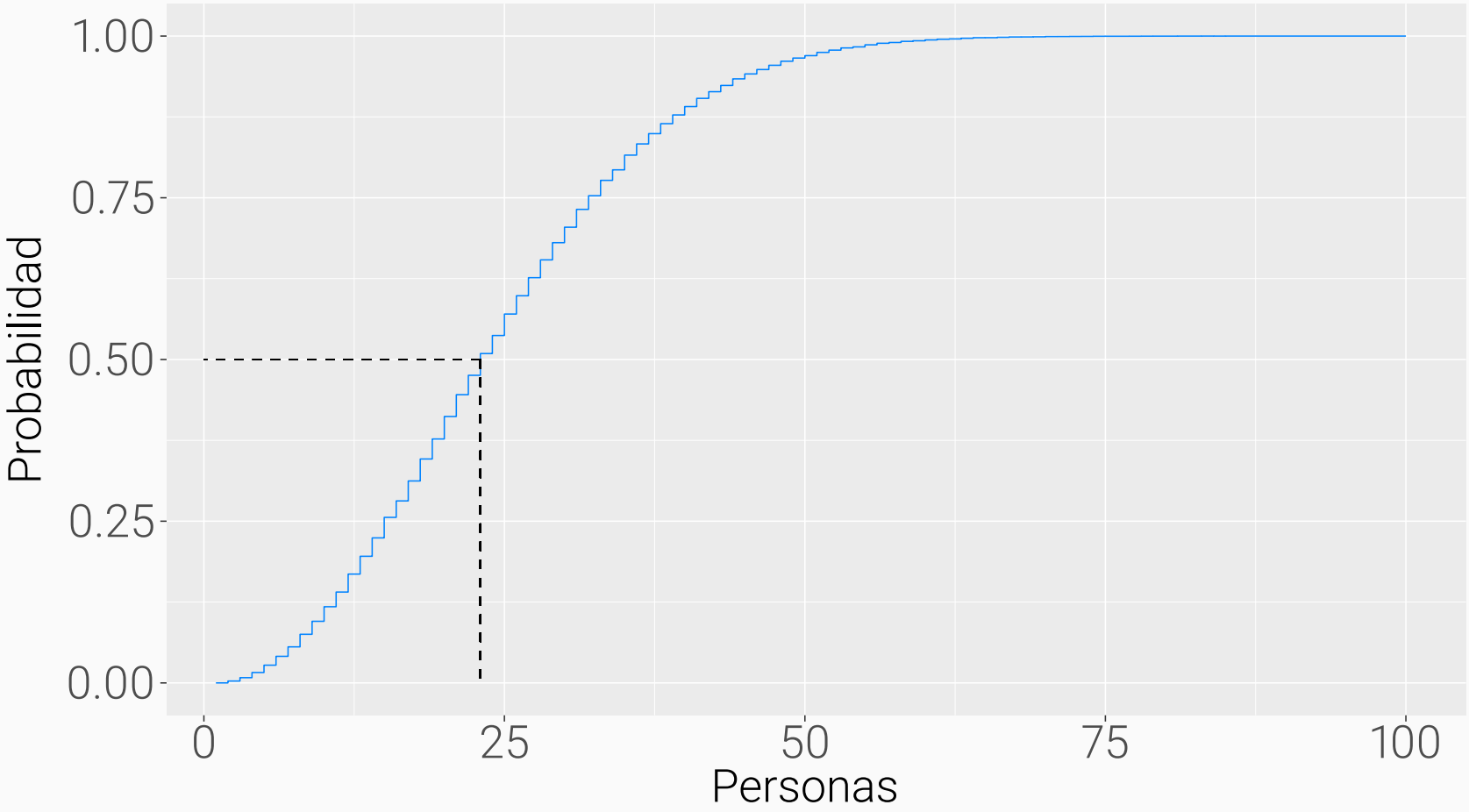


Figura 4: Ejemplo de simulación en paralelo del problema del cumpleaños.

PARALELIZACIÓN: EJEMPLO

```
psim <- function(n, nsims = 1e5L) {...}

prob <- numeric(100)
system.time( for (n in 1:100) prob[n] <- psim(n) )
R>    user  system elapsed
R> 170.17    0.08  170.84

system.time( prob <- lapply(1:100, psim) )
R>    user  system elapsed
R> 173.81    1.72  179.17

library(parallel)
cl <- makeCluster(3)
system.time( prob <- parLapply(cl, 1:100, psim) )
R> user  system elapsed
R> 0.06    0.02   82.99
stopCluster(cl)

cl <- makeCluster(3)
system.time( prob <- clusterApply(cl, 1:100, psim) )
R> user  system elapsed
R> 0.14    0.10   84.34
stopCluster(cl)

library(doParallel)
cl <- makeCluster(3)
registerDoParallel(cl)
system.time( prob <- foreach(n = 1:100, .combine = c) %dopar% psim(n) )
R> user  system elapsed
R> 0.06    0.00   76.25
stopCluster(cl)
```

Microsoft Corporation y Weston, S. (2017). doParallel: Foreach Parallel Adaptor for the 'parallel' Package.

REFERENCIAS

- Microsoft Corporation y Steve Weston. *doParallel: Foreach Parallel Adaptor for the 'parallel' Package*. R package version 1.0.11. 2017.
- Matt Dowle y Arun Srinivasan. *data.table: Extension of 'data.frame'*. 2018.
- Dirk Eddelbuettel y James Joseph Balamuta. "Extending R with C++: A Brief Introduction to Rcpp". En: *PeerJ Preprints* 5 (2017).
- R Core Team. "R: A Language and Environment for Statistical Computing". En: *R Foundation for Statistical Computing* (2018).
- Karl Rupp y col. "ViennaCL-Linear Algebra Library for Multi- and Many-Core Architectures". En: *SIAM Journal on Scientific Computing* (2016).