P3

Ismael Crespo

2 de marzo de 2022

1. Introducción.

Conocer el orden como se realizan los códigos computacionales es de suma importancia para realizar códigos eficientes, este trabajo presenta diferentes rutas para llegar a un mismo objetivo, encontrar los números primos p en una secuencia numérica, y se comparan los tiempos de ejecución para concluir cuál es la mejor opción, la comparación entre los distintos métodos se realizó en *R 4.1.2*.

2. Objetivos.

1.-Examinar las diferencias en los tiempos de ejecución variando, el orden de los números, la cantidad de núcleos asignados al cluster y la variante de la rutina para determinar si el número es primo. 2.-Encontrar todos los divisores de un número (todos los enteros mayores a uno y menores al número mismo que lo dividen exactamente) y examinar si las conclusiones cambian. 3.-Encontrar los factores y sus multiplicidades, es decir, que encuentre para aquellos números primos $1 < p \leq n$ y sus potencias para que el producto de los factores con esas potencias de n y examinar nuevamente si este cambio afectó las conclusiones.

3. Programación en R.

Tomando tres rutinas desarrolladas por E.Schaeffer [2] para encontrar números primos, pasando de rutinas robustas a rutinas mas simplificadas que retiren obviedades de los ciclos, p. ej., si un número no es divisible por 2, tampoco lo será para los demás números pares. (Código 1). El equipo utilizado para la medición de rendimientos es una computadora portátil *Acer E5-573* con un Procesador *Intel Core i5 2.20 GHz*,una memoria RAM instalada de 8.0 GB y un sistema operativo de 64 bits con 4 núcleos.Para consulta más amplia consulte la referencia [4]

Código 1: Función eliminando los elementos que no son divisibles entre 2 para encontrar los número primos.

```
1   primo_2 <- function(n) {
2     if (n < 4) {
3         return(TRUE)
4     }
5     if (n %% 2 == 0) { # par
6         return(FALSE)
7     }
8     for (i in seq(3, max(3, n - 1), 2)) {
9         if (n %% i == 0) {
10            return(FALSE)
11        }}
12    return(TRUE)}
```

Cada una de las tres rutinas fueron evaluadas con los mismos números, sin embargo, para analizarlos estos se ordenaron de manera ascendente (`ot`), descendente (`it`) y aleatorio (`at`), dichas combinaciones de rutinas y ordenamiento numérico fueron evaluadas tanto para 3 núcleos y 1 núcleo por lo que para la primera parte del trabajo se realizaron 18 pruebas diferentes (Código 2).

Código 2: La rutina 3 evaluada para cada ordenamiento de los números y la selección de los núcleos a utilizar.

```
registerDoParallel(makeCluster(detectCores() - 1)) #Nucleos a utilizar
ot_3<- c(ot_3, system.time(foreach(n = original, .combine=c)
%dopar% primo_3(n))[3]) # de menor a mayor
it_3 <- c(it_3, system.time(foreach(n = invertido, .combine=c)
%dopar% primo_3(n))[3]) # de mayor a menor
at_3<- c(at_3, system.time(foreach(n = aleatorio, .combine=c)
%dopar% primo_3(n))[3]) # orden aleatorio
```

La segunda parte del trabajo encuentra el total de números que dividen exactamente a n, posteriormente se encuentran nuevamente todos los valores (p) y la potencia x a la que p tiene que ser elevado para ser igual a n, La Ecuación 1 describe el enunciado y la Ecuación 2 muestra la forma algebraica para encontrar el valor de x (véase el Código 3).La segunda parte del trabajo en su totalidad fue evaluada utilizando tres núcleos.

$$p^x = n \tag{1}$$

$$x = log_p n \tag{2}$$

Código 3: Función para encontrar el valor de la potencia x.

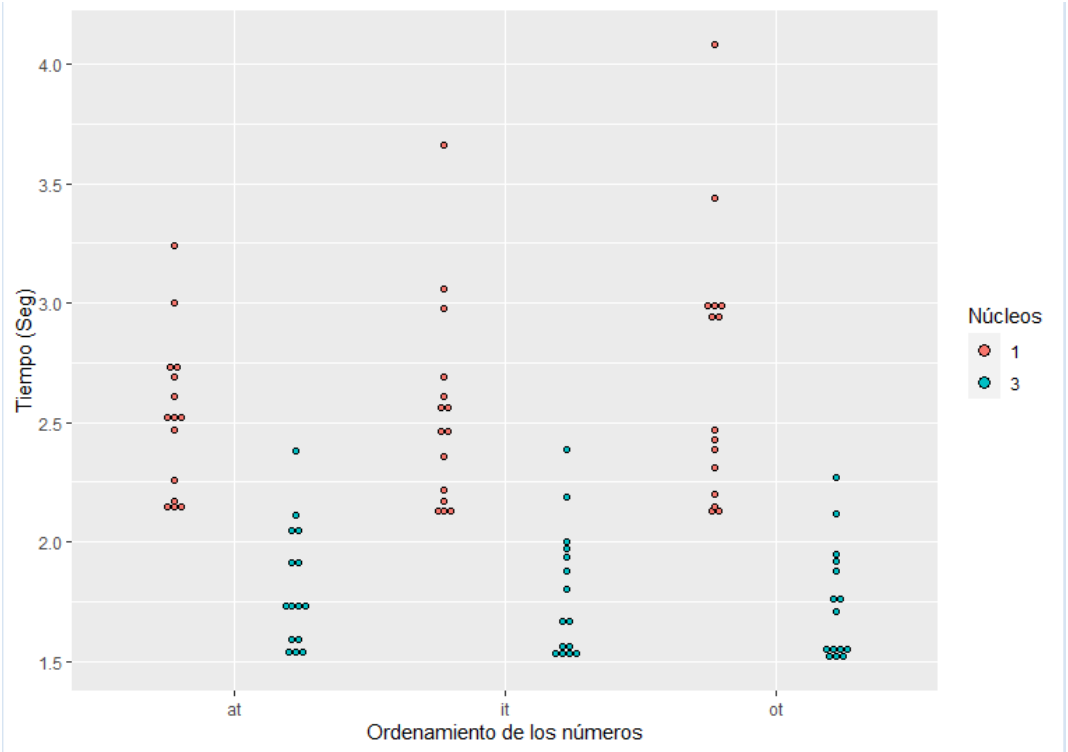
```
potencia<- function(n) {
    if (n == 1 || n == 2) {
        return(TRUE)
    }
    if (n %% 2 == 0) {
        return(FALSE)
    }
    for (i in seq(3, max(3, ceiling(sqrt(n))), 2)) {
        if ((n %% i) == 0) {
            return(FALSE)
        } else { #Encuentra numero primo
           return(x=log(i,base=hasta)) #Encuentra la potencia
           }}}
```

4. Resultados.

La figura 1 muestra los tiempos en segundos que tomo encontrar los números primos desde 1 a 4000, la Figura 1a muestra los tiempos requeridos utilizando solamente un núcleo, de aquí, por conveniencia se decidió comparar la rutina No.3 con cada ordenamiento utilizando 1 y 3 núcleos Figura 1b. De la Figura 1a se deduce que la rutina No.3 es la mas eficiente, posteriormente en la 1b se observa un mejor rendimiento cuando se dedican N0.3 núcleos a esta operación.

En el Cuadro 1 se observan los valores para la media, máximos y mínimos de cada caso así como su desviación estándar, existe una clara reducción en los tiempos de ejecución utilizando tres núcleos, la tabulación se realizó con la función `stargazer` [3]. En el Cuadro 2 se observan valores p muy por debajo del 0.05 % cuando variamos el número de núcleos, es decir que el cambio en esta variable tiene un impacto importante en los rendimientos, así mismo observamos que variar el ordenamiento no tiene significancia, sin embargo, el uso de diferentes rutinas también tiene valores p menores a 0.05 %.Los valores de p fueron obtenidos con la función `wilcox.test`, véase Referencia [1].

(a) Tiempos requeridos para correr cada rutina(1,2,3),para cada orden numérico(ot,it,at).



(b) Tiempos requeridos para correr la rutina 3, con cada orden numérico, para 1 y 3 núcleos.

Figura 1: Comparación de los tiempos requeridos para cada configuración de las rutina.

Cuadro 1: Datos estadísticos de las rutinas, (ot_3,it_3 y at_3, representan los valores para la rutina No.3 utilizando 3 núcleos(

Statistic	Promedio	Median	Max	Median	St. Dev.
ot_1_1	2.33	2.98	3.28	2.94	0.23
it_1_1	2.84	3.02	3.95	2.87	0.36
at_1_1	2.81	2.97	3.69	2.89	0.23
ot_2_1	2.30	2.70	2.95	2.71	0.14
it_2_1	2.60	2.71	2.83	2.70	0.08
at_2_1	2.53	2.72	3.11	2.66	0.18
ot_3_1	2.09	2.27	2.70	2.17	0.20
it_3_1	2.07	2.25	2.92	2.20	0.21
at_3_1	2.08	2.28	2.72	2.20	0.20
ot_3	1.51	1.61	1.96	1.55	0.14
it_3	1.50	1.69	2.34	1.59	0.25
at_3	1.53	1.69	1.88	1.70	0.13

Cuadro 2: Valores p entre diferentes experimentos, variando el número de núcleos, la rutina y el ordenamiento.

Variable 1	Variable 2	$p - value$
ot_3	ot_3_1	$3{,}58 \times 10^{-9}$
ot_3	it_3	$3{,}19 \times 10^{-1}$
ot_1	ot_3	$5{,}19 \times 10^{-4}$
ot_2	it_2	1,00
ot_1	it_1	0,60
at_1	it_1	0,90
at_3	it_3	0,38

Los resultados de la segunda etapa del trabajo, se presentan en la Figura 2, nuevamente utilizaremos la rutina No.3 para comprar los rendimientos ejecutando las rutinas con tres núcleos, ahora para encontrar todos los números divisores exactos dentro de n y x a partir de la Ecuación 2. No se observa un cambio importante, como se ve en la Figura 1, en los tiempos requeridos para correr cada rutina, el Cuadro 3 presenta los datos estadísticos para estas rutinas, se observa una medai.

Cuadro 3: Datos estadísticos de la rutina No.3, la rutina que encuentra los divisores exactos y la rutina para encontrar la potencia x.

Statistic	Min	Promedio	Max	Median	St. Dev.
ot_3	1.51	1.61	1.96	1.55	0.14
it_3	1.50	1.69	2.34	1.59	0.25
at_3	1.53	1.69	1.88	1.70	0.13
ot_residuo	1.55	1.61	1.83	1.60	0.07
it_residuo	1.56	1.61	1.84	1.59	0.07
at_residuo	1.55	1.60	1.74	1.58	0.05
ot_potencia	1.52	1.91	3.37	1.84	0.46
it_potencia	1.53	1.84	2.75	1.85	0.31
at_potencia	1.53	2.01	4.22	1.86	0.66

Referencias

[1] Data Analytics.org.uk. Basic statistical tests using r. 2022. URL `https://www.dataanalytics.org.uk/basic-statistical-tests-using-r/`.

[2] E.Schaeffer. Simulación. 2022. URL `https://satuelisa.github.io/simulation`.

[3] H.Marek. stargazer.well-formatted regression and summary statistics tables. r package version 5.2.2. 2018. URL `https://CRAN.R-project.org/package=stargazer`.

[4] I.Crespo. P2:autómata celular. 2022. URL `https://github.com/IsmaelHC/Simulacion-NANO-2022/tree/main/P3`.

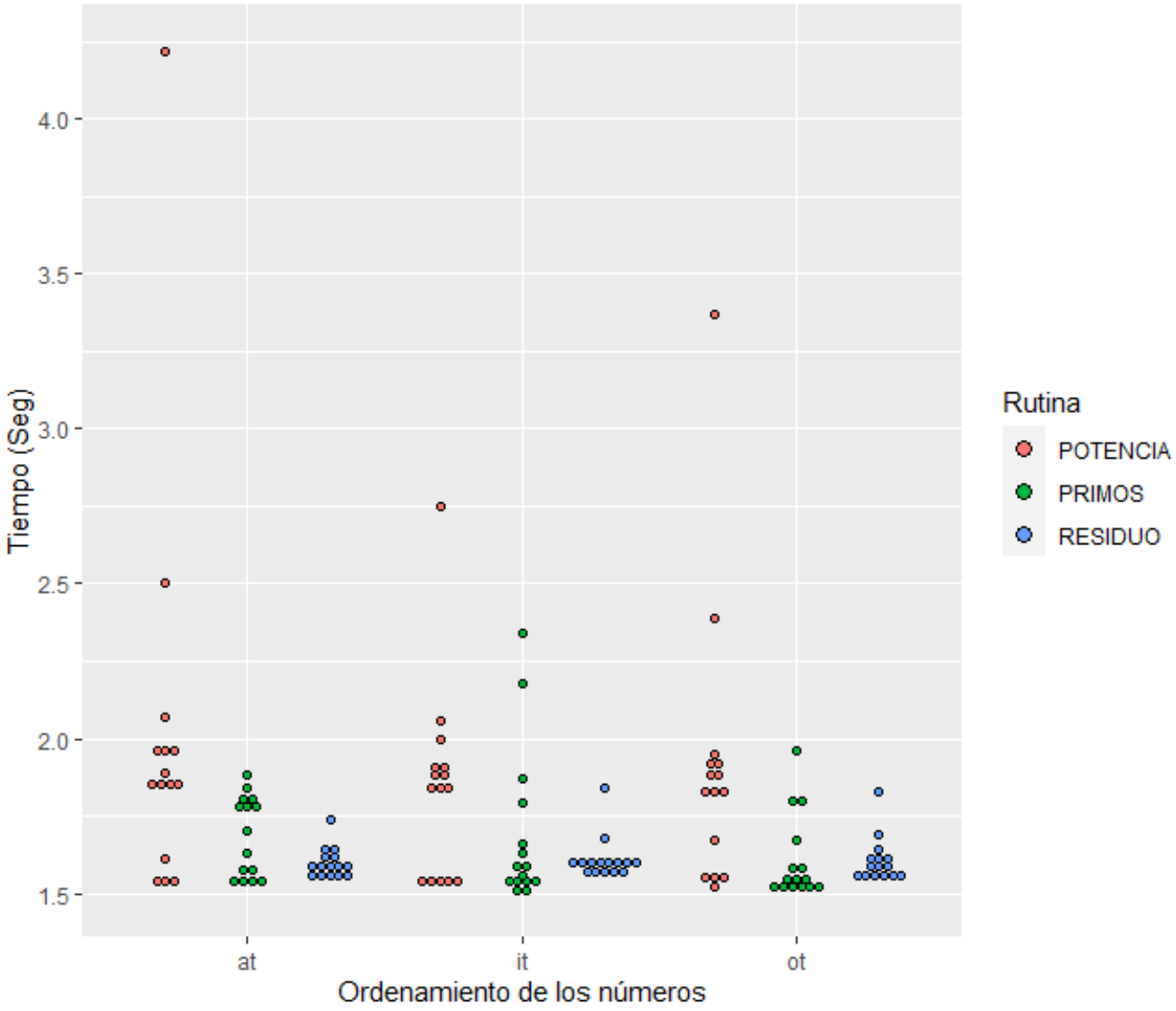


Figura 2: Tiempos requeridos,utilizando tres núcleos, para correr la rutina 3 (PRIMOS), la rutina para encontrar los números divisorios exactos(RESIDUO) y para encontrar la potencia x (POTENCIA).