



Informe: Proyecto Final PFC

Santiago Avalo Monsalve - 2359442
Daniel Gómez Cano - 2359396
Edward Stivens Pinto - 2359431
Manuel Jesús Rosero Zuñiga - 2176007

Trabajo presentado a:
Carlos Andrés Delgado S.

Universidad del Valle
Fundamentos de Programación Funcional y Concurrente
Ingeniería de Sistemas
Tuluá - Valle del Cauca
2024

1. Informe de procesos

Esto es realizado tomando en cuenta lo explicado en el taller, por lo que se usarán los distintos Types otorgados por el documento los cuales engloban:

```
type Tablon = ( Int , Int , Int )
```

```
type Finca = Vector [ Tablon ]
```

```
type Distancia = Vector [ Vector [ Int ] ]
```


```
type ProgRiego = Vector [ Int ]
```

```
type TiempoInicioRiego = Vector [ Int ]
```

Además de en el caso de tablón también se usarán las funciones para buscar dentro del type, las cuales son tsup, treg y prio.

1.1 Calculando el tiempo de inicio de riego

Se implementó la Función “TiR” que recibe una “f” de “n” tamaño la cual representa una Finca y un “pi” el cual es una programación de riego, la cual es generada por una función que será vista más adelante, en este caso vamos a usar una finca que corresponde a un vector de la siguiente manera:



10	3	2
15	5	1
20	4	3

Al ser de tamaño 3 el “pi” que se le asignará a la función va a ser: [0, 1, 2]

✓ Local

```
> f = Vector1@1961 "Vector((10,3,2), (15,5,1), (20,4,3))"  
> pi = Vector1@1962 "Vector(0, 1, 2)"  
> this = App$@1963
```

Luego de que se tengan asignados los valores que necesita la función se pasa a la siguiente parte en la que se crea un Array denominado “tiempos”, este Array se le asigna de tamaño el “length” de f y se llena con 0s antes de usarlo como un placeholder. Esto antes de crear otro bucle for el cual esta vez es usado para llenar tiempos en sus respectivos placeholders, se empieza desde el elemento [1] del array ya que el inicio del primer riego siempre será 0.

También se crea un par de variables durante el bucle la cual posee de valor el que tiene pi en la posición [j - 1] o [j], dependiendo de si es “prevTablon” o “currTablon”. j es el valor del bucle que va subiendo y las variables representan el tablon previo y el tablon actual respectivamente.

```
✓ Local
> pi$1 = Vector1@1962 "Vector(0, 1, 2)"
> tiempos$1 = int[3]@1977
> f$1 = Vector1@1961 "Vector((10,3,2), (15,5,1), (20,4,3))"
  j = 1
  prevTablon = 0
  currTablon = 1
> MODULE$ = App$@1963
> random = Random@1972
```

Posteriormente en la posición en la que esté tiempos se realiza una ópera en la cual sumamos el valor del tiempos[prevTablon] con el valor del tiempo de riego de la f en el tablon previo, como el primer caso es 0 solo se asignó a Tiempos[1] un valor de 0 + 3.

```

✓ Local
> pi$1 = Vector1@1962 "Vector(0, 1, 2)"
✓ tiempos$1 = int[3]@1977
  0 = 0
  1 = 3
  2 = 0
> f$1 = Vector1@1961 "Vector((10,3,2), (15,5,1), (20,4,3))"
  j = 2
  prevTablon = 1
  currTablon = 2
> MODULE$ = App$@1963
> random = Random@1972

```

Ya en la posición tiempos[2] si se realiza la suma con un valor de más de 0 por lo que se puede ver como el 3 del tiempo[1] se ha sumado con el tiempo de riego del previo tablón el cual era 5, resultando en 8. Una vez que el bucle cierra lo que sigue es convertir el array en un vector mediante `.toVector` ya que `TiempoDeInicioRiego` es un vector.

```

✓ Local
> f = Vector1@1965 "Vector((10,3,2), (15,5,1), (20,4,3))"
> pi = Vector1@1966 "Vector(0, 1, 2)"
✓ tiempos = int[3]@1977
  0 = 0
  1 = 3
  2 = 8
> this = App$@1967

```

1.1.2 costoRiegoTablon

Se implementó la Función “costoRiegoTablon” que recibe un entero “i” el cual representa el número de un tablón al cual vamos a calcular el costo, una finca f de “n” tamaño, un pi que es la programación de riego, en este caso vamos a usar una finca que corresponde a un vector de la siguiente manera:

10	3	2
15	5	1

El pi sería entonces de [0, 1, 2] y el i sería de 1 para poder observar el costo de riego de este tablón luego del anterior, lo primero que se hace es crear la variable TiempoInicio para la cual usaremos el TiR que se vió anteriormente para poder calcular el tiempo total que se demora en regar hasta el tablon, por lo que luego de calcular el TiR se le suma el valor del tiempo de riego del tablón a calcular.

✓ Local

```
i = 1
> f = Vector1@1970 "Vector((10,3,2), (15,5,1))"
> pi = Vector1@1971 "Vector(0, 1)"
    tiempoInicio = 3
> this = App$@1972
```

Se calcula la suma resultando en un 8 como tiempoFinal para ser usado posteriormente, ya que primero se debe revisar un condicional el cual es $(T_s \text{ del tablon} - T_r \text{ del tablon}) \geq \text{TiempoInicio}$ el cual en este caso se cumple por lo que el resultado que retorna la función será $T_s - \text{TiempoFinal}$

```
✓ Local
  i = 1
> f = Vector1@1970 "Vector((10,3,2), (15,5,1))"
> pi = Vector1@1971 "Vector(0, 1)"
  tiempoInicio = 3
  tiempoFinal = 8
> this = App$@1972
```

1.1.3 costoRiegoFinca

Se implementó la Función “costoRiegoFinca” que recibe una finca de “n” tamaño y una programación de riego pi, y esta función sirve mediante una sumatoria o sea un bucle “for”. En este caso vamos a usar una finca que corresponde a un vector de la siguiente manera:

10	2	1
8	3	2
15	1	3

Cuya pi será tal como en varios anteriores [0, 1, 2] y luego de este punto lo único que se puede explicar de esta función es que realiza un bucle “for” que va probando con cada tablon llamando la función previa costoRiegoTablon, usando map para poder al final realizar una suma de cada uno de los resultados retornando un Int.

```

✓ Local
> f = Vector1@1961 "Vector((10,2,1), (8,3,2), (15,1,3))"
> pi = Vector1@1962 "Vector(0, 1, 2)"
> this = App$@1963

```

1.1.4 costoRiegoFincaPar

Se implementó la Función “costoRiegoFincaPar” que recibe una finca de “n” tamaño y una programación de riego pi, y esta función sirve mediante una sumatoria o sea un bucle “for”. Solo que vamos a usar la paralelización mediante par.map, en este caso vamos a usar una finca que corresponde a un vector de la siguiente manera:

10	3	2
15	5	1

Cuya pi será [0, 1] y tal como en el caso anterior utiliza un bucle for para ir probando en cada tablón, solo que ahora lo que se usa par.map lo que permite a cada uno de los elementos de la finca que se le aplique la función anterior, por lo que cada una se hace por separado durante el bucle for.

```

✓ Local
> f = Vector1@1961 "Vector((10,3,2), (15,5,1))"
> pi = Vector1@1962 "Vector(0, 1)"
> this = App$@1963

```

1.1.5 costoMovilidad

Se implementó la Función “costoMovilidad” que recibe una finca de “n” tamaño, una programación de riego pi y una “d” distancia, esta función sirve mediante una sumatoria o sea un bucle “for”. En este caso vamos a usar una finca y una distancia que corresponde a vectores de la siguiente manera:

10	2	1
8	3	2
15	1	3

0	4	7
4	0	5
7	5	0

Una vez que se ha llamado la función está al igual que en el caso anterior utiliza un bucle for en el cual suma cada uno de los valores de la distancia, usando el pi para revisar en cada una de las posiciones de la distancia. Para al final retorna un Int que corresponde a lo siguiente:

```
✓ Local
> d$1 = Vector1@1987 "Vector(Vector(0, 4, 7), Vector(4, 0, 5), Vector(7, 5, 0))"
> pi$4 = Vector1@1986 "Vector(0, 1, 2)"
  j = 0
> MODULE$ = App$@1988
> random = Random@2007
```

1.1.6 costoMovilidadPar

Se implementó la Función “costoMovilidadPar” los mismos valores que costoMovilidad los cuales son f, pi y d, esta función sirve mediante una sumatoria

o sea un bucle “for”. En este caso vamos a usar una finca y una distancia que corresponde a vectores de la siguiente manera:

10	3	2
15	5	1

0	5
5	0

Tal como en el caso anterior lo que se realizará ahora será un bucle for usando map para afectar a cada uno de los valores en la distancia con una suma, solo que ahora que estamos usando “.par” esto se realizará en paralelo, haciendo que cada función se resuelva independiente de las otras.

```
Local
> d$2 = Vector1@1991 "Vector(Vector(0, 5), Vector(5, 0))"
> pi$5 = Vector1@1990 "Vector(0, 1)"
  j = 0
> MODULE$ = App$@1992
> random = Random@2120
```

1.2 Generando programaciones de riego

Se implementó la Función “generarProgramacionesRiego” que recibe una finca de “n” tamaño la cual es lo único que necesita la función al ser usada, en este caso vamos a usar una finca que corresponde a un vector de la siguiente manera:

10	5	1
15	6	2
20	7	3

Una vez tenemos eso establecido para probar la función, lo siguiente que hace la función es recibir los valores y asignarlo a f, esto justo antes de usar la función “.length” para medir el límite de un bucle “for”.

```
✓ VARIABLES
  ✓ Local
    > f = Vector1@1961 "Vector((10,5,1), (15,6,2), (20,7,3))"
    > this = App$@1962
```

Posteriormente se crea la variable “indices” el cual recibe la cuenta del length como un vector el cual va a retornar una vez termine el ciclo, lo único que debe hacer luego de recibir los valores del ciclo es permutarlos para que retorne ese Vector[ProgRiego] la cual será un Vector de [0, 1, 2] al ser solo un tablero de tamaño 3.

```
✓ Local
  > f = Vector1@1961 "Vector((10,5,1), (15,6,2), (20,7,3))"
  > indices = Vector1@1971 "Vector(0, 1, 2)"
  > this = App$@1962
```

1.3 Generando programaciones de riego Paralelizada

Se implementó la Función “generarProgramacionesRiegoPar” que recibe una finca de “n” tamaño que será lo único que se usará en esta función, en este caso vamos a usar una finca que corresponde a un vector de la siguiente manera:

10	3	2
15	5	1
20	4	3

Lo primero que hace la función tal como antes es recoger los datos presentados para la realización de la operación necesaria, por ahora no se realiza paralelización.

```
✓ Local  
> f = Vector1@1961 "Vector((10,3,2), (15,5,1), (20,4,3))"  
> this = App$@1962
```

Lo que sigue es lo mismo que anteriormente que es usar un bucle para poder crear índices en forma de vectores, los cuales posteriormente gracias a la paralelización son permutados en un solo vector, algo que para nada parece muy efectivo con unas operaciones tan simples y tan pequeñas.

```
✓ Local  
> f = Vector1@1961 "Vector((10,3,2), (15,5,1), (20,4,3))"  
> indices = Vector1@1970 "Vector(0, 1, 2)"  
> this = App$@1962
```

1.3.1 programaciónRiegoOptimo

Se implementó la Función “programaciónRiegoOptimo” que recibe una finca de “n” y una “d” distancia, en este caso vamos a usar una finca y distancia que corresponde a un vector de la siguiente manera:

10	2	1
15	3	2
12	1	3

0	3	2
3	0	4
2	4	0

El programa usa entonces la “f” otorgada para poder crear la variante “programaciones” que reúne cada una de las posibles programaciones de riego, en este caso se crearon unas 6 para ser usadas posteriormente.

```
> f = Vector1@1985 "Vector((10,2,1), (15,3,2), (12,1,3))"
> d = Vector1@1986 "Vector(Vector(0, 3, 2), Vector(3, 0, 4), Vector(2, 4, 0))"
✓ programaciones = Vector1@2018 "Vector(Vector(0, 1, 2), Vector(0, 2, 1), Vecto...
  ✓ prefix1 = Object[6]@2022
    > 0 = Vector1@2023 "Vector(0, 1, 2)"
    > 1 = Vector1@2024 "Vector(0, 2, 1)"
    > 2 = Vector1@2025 "Vector(1, 0, 2)"
    > 3 = Vector1@2026 "Vector(1, 2, 0)"
    > 4 = Vector1@2027 "Vector(2, 0, 1)"
    > 5 = Vector1@2028 "Vector(2, 1, 0)"
  > this = App$@1987
```

Lo que sigue es la realización de un map en el que a cada uno de los elementos de programaciones se le suman los costos de riego de la finca con los de movilidad, usando pi que es tomado de programaciones para la operación.

```
✓ Local
  > f$4 = Vector1@1985 "Vector((10,2,1), (15,3,2), (12,1,3))"
  > d$3 = Vector1@1986 "Vector(Vector(0, 3, 2), Vector(3, 0, 4...
  > pi = Vector1@2023 "Vector(0, 1, 2)"
  > MODULE$ = App$@1987
  > random = Random@2036
```

Todo esto para crearnos una nueva variable llamada “costos” que nos retorna una serie de vectores compuestos del pi al que realizamos la operación anterior y su resultado, esto para realizar una última operación que nos retornará el valor final de este riego óptimo.

```
> f = Vector1@1985 "Vector((10,2,1), (15,3,2), (12,1,3))"
> d = Vector1@1986 "Vector(Vector(0, 3, 2), Vector(3, 0, 4), Vector(
> programaciones = Vector1@2018 "Vector(Vector(0, 1, 2), Vector(0, 2
✓ costos = Vector1@2072 "Vector((Vector(0, 1, 2),31), (Vector(0, 2,
  ✓ prefix1 = Object[6]@2077
    > 0 = Tuple2@2078 "(Vector(0, 1, 2),31)"
    > 1 = Tuple2@2079 "(Vector(0, 2, 1),32)"
    > 2 = Tuple2@2080 "(Vector(1, 0, 2),28)"
    > 3 = Tuple2@2081 "(Vector(1, 2, 0),30)"
    > 4 = Tuple2@2082 "(Vector(2, 0, 1),32)"
    > 5 = Tuple2@2083 "(Vector(2, 1, 0),33)"
  > this = App$@1987
```

Lo último que se realiza es esta operación de aplicar un mínimo a los vectores, solo que esta escrita de la siguiente manera `costos.minBy(_._2)` lo que permite que lo retornado sean los dos pares menores, o sea el más óptimo.

```
> x$1 = Tuple2@2078 "(Vector(0, 1, 2),31)"
> MODULE$ = App$@1987
> random = Random@2036
```

1.3.2 programaciónRiegoOptimoPar

Se implementó la Función “programaciónRiegoOptimoPar” que recibe una finca de “n” y una “d” distancia, en este caso vamos a usar una finca y distancia que corresponde a un vector de la siguiente manera:

10	3	2
15	5	1
20	4	3

0	5	10
5	0	15
10	15	0

Se procede a crear programaciones usando el `f` otorgado al llamar la función, creando 6 que serán usadas más adelante, hasta ahora no se ha usado la paralelización.

```
> f = Vector1@1985 "Vector((10,3,2), (15,5,1), (20,4,3))"
> d = Vector1@1986 "Vector(Vector(0, 5, 10), Vector(5, 0, 15), Vector(10, 15, 0))"
✓ programaciones = Vector1@2089 "Vector(Vector(0, 1, 2), Vector(0, 2, 1), Vector(1, 0, 2), Vector(1, 2, 0), Vector(2, 0, 1), Vector(2, 1, 0))"
✓ prefix1 = Object[6]@2093
  > 0 = Vector1@2094 "Vector(0, 1, 2)"
  > 1 = Vector1@2095 "Vector(0, 2, 1)"
  > 2 = Vector1@2096 "Vector(1, 0, 2)"
  > 3 = Vector1@2097 "Vector(1, 2, 0)"
  > 4 = Vector1@2098 "Vector(2, 0, 1)"
  > 5 = Vector1@2099 "Vector(2, 1, 0)"
> this = App$@1987
```

Tal como en el caso anterior se vuelve a crear costos el cual reunió las sumas del costo de movilidad y costo de riego luego de que dicha operación se aplicará a cada un dirección de programaciones, solo que en este caso se usó la paralelización al hacer el `map`. Lo que nos permitió calcular cada una de las operaciones de manera casi simultánea a pesar de que no se pudo registrar en el screenshot.

```

> f = Vector1@1985 "Vector((10,3,2), (15,5,1), (20,4,3))"
> d = Vector1@1986 "Vector(Vector(0, 5, 10), Vector(5, 0, 15), Vector(10, 15, 0))"
> programaciones = Vector1@2089 "Vector(Vector(0, 1, 2), Vector(0, 2, 1), Vector(1, 0, 2))"
✓ costos = ParVector@2218 "ParVector((Vector(0, 1, 2),42), (Vector(0, 2, 1),48), (Vector(1, 0, 2),35))"
> scala$collection$parallel$ParIterableLike$_tasksupport = ExecutionContextTaskSupport@2219
  ScanLeaf$module = null
  ScanNode$module = null
✓ vector = Vector1@2224 "Vector((Vector(0, 1, 2),42), (Vector(0, 2, 1),48), (Vector(1, 0, 2),35))"
  ✓ prefix1 = Object[6]@2226
    > 0 = Tuple2@2227 "(Vector(0, 1, 2),42)"
    > 1 = Tuple2@2228 "(Vector(0, 2, 1),48)"
    > 2 = Tuple2@2229 "(Vector(1, 0, 2),35)"
    > 3 = Tuple2@2230 "(Vector(1, 2, 0),50)"
    > 4 = Tuple2@2231 "(Vector(2, 0, 1),37)"
    > 5 = Tuple2@2232 "(Vector(2, 1, 0),46)"
  > this = App$@1987

```

Finalmente sucede tal como con la función anterior y solo se vuelven a llamar los costos para poder hallar el par menor, cosa para la que no sé usó la paralelización y al final nos retorna el valor adecuado.

```

> x$2 = Tuple2@2231 "(Vector(2, 0, 1),37)"
> MODULE$ = App$@1987
> random = Random@2166

```

2. Informe de paralización

2.1 generarProgramacionesRiegoPar

```

// Función Paralelizada
def generarProgramacionesRiegoPar( f : Finca ) : Vector [ProgRiego] = {
  // Genera las programaciones posibles de manera paralela
  val indices = (0 until f.length).toVector
  indices.permutations.toVector.par.toVector
}

```


Ley de Amdahl

$$S = 1/(1 - P) + P/N$$

$P = 0,95$ //Puesto que la paralelización está aplicada para que se aplique a todas las permutaciones

$N = 8$ //Cantidad estimada de procesadores

$$S = 1/(1 - 0,95) + 0,95/8$$

$$S = 6,71$$

```
iteracion = 1
Benchmark GenerarProgramacionesRiego long = 8
Secuencial: 14.5692 ms ms
Paralelo: 14.3292 ms ms
Speedup: 1.0167490159953103
```

```
iteracion = 2
Benchmark GenerarProgramacionesRiego long = 8
Secuencial: 16.1801 ms ms
Paralelo: 14.306 ms ms
Speedup: 1.1310009786103734
```

```
iteracion = 3
Benchmark GenerarProgramacionesRiego long = 8
Secuencial: 13.9486 ms ms
Paralelo: 14.4917 ms ms
Speedup: 0.9625233754493953
```

```
iteracion = 4
Benchmark GenerarProgramacionesRiego long = 8
Secuencial: 15.7291 ms ms
Paralelo: 14.644 ms ms
Speedup: 1.0740986069379952
```

```
iteracion = 5
Benchmark GenerarProgramacionesRiego long = 8
Secuencial: 14.498 ms ms
Paralelo: 14.1794 ms ms
Speedup: 1.0224692159047633
```

```
iteracion = 6
Benchmark GenerarProgramacionesRiego long = 8
Secuencial: 14.3151 ms ms
Paralelo: 14.734 ms ms
Speedup: 0.9715691597665264
```

```
iteracion = 7
Benchmark GenerarProgramacionesRiego long = 8
Secuencial: 14.1646 ms ms
Paralelo: 14.0606 ms ms
Speedup: 1.007396554912308
```

```
iteracion = 8
Benchmark GenerarProgramacionesRiego long = 8
Secuencial: 14.3377 ms ms
Paralelo: 15.3702 ms ms
Speedup: 0.9328245566095431
```

```
iteracion = 9
Benchmark GenerarProgramacionesRiego long = 8
Secuencial: 15.3728 ms ms
Paralelo: 13.9109 ms ms
Speedup: 1.1050902529670978
```

```
iteracion = 10
Benchmark GenerarProgramacionesRiego long = 8
Secuencial: 13.9002 ms ms
Paralelo: 14.0503 ms ms
Speedup: 0.9893169540863896
```

```
iteracion = 1
Benchmark GenerarProgramacionesRiego long = 7
Secuencial: 1.9272 ms ms
Paralelo: 1.7065 ms ms
Speedup: 1.1293290360386756
```

```
iteracion = 2
Benchmark GenerarProgramacionesRiego long = 7
Secuencial: 1.6838 ms ms
Paralelo: 1.8685 ms ms
Speedup: 0.9011506556061011
```

```
iteracion = 3
Benchmark GenerarProgramacionesRiego long = 7
Secuencial: 1.7114 ms ms
Paralelo: 1.6878 ms ms
Speedup: 1.0139826993719636
```

```
iteracion = 4
Benchmark GenerarProgramacionesRiego long = 7
Secuencial: 1.7241 ms ms
Paralelo: 1.8477 ms ms
Speedup: 0.933106023705147
```

```
iteracion = 5
Benchmark GenerarProgramacionesRiego long = 7
Secuencial: 1.7612 ms ms
Paralelo: 1.8181 ms ms
Speedup: 0.9687035916616248
```

```
iteracion = 6
Benchmark GenerarProgramacionesRiego long = 7
Secuencial: 1.712 ms ms
Paralelo: 1.7353 ms ms
Speedup: 0.9865729268714343
```

```
iteracion = 7
Benchmark GenerarProgramacionesRiego long = 7
Secuencial: 1.8651 ms ms
Paralelo: 1.9319 ms ms
Speedup: 0.9654226409234432
```

```
iteracion = 8
Benchmark GenerarProgramacionesRiego long = 7
Secuencial: 1.6812 ms ms
Paralelo: 1.9578 ms ms
Speedup: 0.8587189702727551
```

```
iteracion = 9
Benchmark GenerarProgramacionesRiego long = 7
Secuencial: 1.7376 ms ms
Paralelo: 2.1001 ms ms
Speedup: 0.8273891719441933
```

```
iteracion = 10
Benchmark GenerarProgramacionesRiego long = 7
Secuencial: 1.7756 ms ms
Paralelo: 1.8234 ms ms
Speedup: 0.9737852363716135
```

```
iteracion = 1
Benchmark GenerarProgramacionesRiego long = 6
Secuencial: 0.4226 ms ms
Paralelo: 0.3828 ms ms
Speedup: 1.1039707419017764
```

```
iteracion = 2
Benchmark GenerarProgramacionesRiego long = 6
Secuencial: 0.446 ms ms
Paralelo: 0.2831 ms ms
Speedup: 1.5754150476863298
```

```
iteracion = 3
Benchmark GenerarProgramacionesRiego long = 6
Secuencial: 0.2754 ms ms
Paralelo: 0.2615 ms ms
Speedup: 1.053154875717017
```

```
iteracion = 4
Benchmark GenerarProgramacionesRiego long = 6
Secuencial: 0.2649 ms ms
Paralelo: 0.5019 ms ms
Speedup: 0.5277943813508668
```

```
iteracion = 5
Benchmark GenerarProgramacionesRiego long = 6
Secuencial: 0.2911 ms ms
Paralelo: 0.2579 ms ms
Speedup: 1.1287320666925165
```

```
iteracion = 6
Benchmark GenerarProgramacionesRiego long = 6
Secuencial: 0.3817 ms ms
Paralelo: 0.3733 ms ms
Speedup: 1.022502009107956
```

```
iteracion = 7
Benchmark GenerarProgramacionesRiego long = 6
Secuencial: 0.3642 ms ms
Paralelo: 0.3716 ms ms
Speedup: 0.9800861141011842
```

```
iteracion = 8
Benchmark GenerarProgramacionesRiego long = 6
Secuencial: 0.3604 ms ms
Paralelo: 0.2847 ms ms
Speedup: 1.2658939234281699
```

```
iteracion = 9
Benchmark GenerarProgramacionesRiego long = 6
Secuencial: 0.2802 ms ms
Paralelo: 0.2647 ms ms
Speedup: 1.0585568568190404
```

```
iteracion = 10
Benchmark GenerarProgramacionesRiego long = 6
Secuencial: 0.339 ms ms
Paralelo: 0.3936 ms ms
Speedup: 0.8612804878048781
```


iteracion = 1
Benchmark GenerarProgramacionesRiego long = 5
Secuencial: 0.062 ms ms
Paralelo: 0.087 ms ms
Speedup: 0.7126436781609196

iteracion = 2
Benchmark GenerarProgramacionesRiego long = 5
Secuencial: 0.0622 ms ms
Paralelo: 0.0678 ms ms
Speedup: 0.9174041297935103

iteracion = 3
Benchmark GenerarProgramacionesRiego long = 5
Secuencial: 0.065 ms ms
Paralelo: 0.0589 ms ms
Speedup: 1.1035653650254669

iteracion = 4
Benchmark GenerarProgramacionesRiego long = 5
Secuencial: 0.0622 ms ms
Paralelo: 0.0713 ms ms
Speedup: 0.8723702664796633

iteracion = 5
Benchmark GenerarProgramacionesRiego long = 5
Secuencial: 0.0772 ms ms
Paralelo: 0.0644 ms ms
Speedup: 1.1987577639751554

iteracion = 6
Benchmark GenerarProgramacionesRiego long = 5
Secuencial: 0.0604 ms ms
Paralelo: 0.0743 ms ms
Speedup: 0.8129205921938089

iteracion = 7
Benchmark GenerarProgramacionesRiego long = 5
Secuencial: 0.101 ms ms
Paralelo: 0.0962 ms ms
Speedup: 1.0498960498960501

iteracion = 8
Benchmark GenerarProgramacionesRiego long = 5
Secuencial: 0.0704 ms ms
Paralelo: 0.0621 ms ms
Speedup: 1.1336553945249597

iteracion = 9
Benchmark GenerarProgramacionesRiego long = 5
Secuencial: 0.0645 ms ms
Paralelo: 0.0645 ms ms
Speedup: 1.0

iteracion = 10
Benchmark GenerarProgramacionesRiego long = 5
Secuencial: 0.0606 ms ms
Paralelo: 0.1016 ms ms
Speedup: 0.5964566929133859

iteracion = 1
Benchmark GenerarProgramacionesRiego long = 4
Secuencial: 0.0442 ms ms
Paralelo: 0.0209 ms ms
Speedup: 2.1148325358851676

iteracion = 2
Benchmark GenerarProgramacionesRiego long = 4
Secuencial: 0.0272 ms ms
Paralelo: 0.0185 ms ms
Speedup: 1.4702702702702704

iteracion = 3
Benchmark GenerarProgramacionesRiego long = 4
Secuencial: 0.0311 ms ms
Paralelo: 0.0258 ms ms
Speedup: 1.2054263565891472

iteracion = 4
Benchmark GenerarProgramacionesRiego long = 4
Secuencial: 0.0181 ms ms
Paralelo: 0.0214 ms ms
Speedup: 0.8457943925233646

iteracion = 5
Benchmark GenerarProgramacionesRiego long = 4
Secuencial: 0.0221 ms ms
Paralelo: 0.0229 ms ms
Speedup: 0.9650655021834061

iteracion = 6
Benchmark GenerarProgramacionesRiego long = 4
Secuencial: 0.0228 ms ms
Paralelo: 0.0311 ms ms
Speedup: 0.7331189710610932

iteracion = 7
Benchmark GenerarProgramacionesRiego long = 4
Secuencial: 0.0271 ms ms
Paralelo: 0.018 ms ms
Speedup: 1.5055555555555555

iteracion = 8
Benchmark GenerarProgramacionesRiego long = 4
Secuencial: 0.0285 ms ms
Paralelo: 0.0302 ms ms
Speedup: 0.9437086092715232

iteracion = 9
Benchmark GenerarProgramacionesRiego long = 4
Secuencial: 0.0279 ms ms
Paralelo: 0.0267 ms ms
Speedup: 1.044943820224719

iteracion = 10
Benchmark GenerarProgramacionesRiego long = 4
Secuencial: 0.0585 ms ms
Paralelo: 0.0231 ms ms
Speedup: 2.5324675324675328

2.2 costoRiegoFincaPar

```
// Función Paralelizada
def costoRiegoFincaPar( f : Finca , pi : ProgRiego) : Int = {
  // Devuelve el costo total de regar una finca f dada una
  // programación de riego pi, calculando en paralelo
  (0 until f . length).par.map( i => costoRiegoTablon(i , f , pi)) .sum
}
```

Ley de Amdahl

$$S = 1/(1 - P) + P/N$$

$P = 0,85$ //Puesto que la paralelización está aplicada para que se aplique a todas los valore del map, menos a la suma final

$N = 8$ //Cantidad estimada de procesadores

$$S = 1/(1 - 0,85) + 0,85/8$$

$$S = 3,9$$

iteracion = 1
Benchmark CostoRiegoFinca long = 8
Secuencial: 55.1668 ms ms
Paralelo: 14.812 ms ms
Speedup: 3.7244666486632463

iteracion = 2
Benchmark CostoRiegoFinca long = 8
Secuencial: 14.2325 ms ms
Paralelo: 18.1475 ms ms
Speedup: 0.7842678054828488

iteracion = 3
Benchmark CostoRiegoFinca long = 8
Secuencial: 13.9985 ms ms
Paralelo: 15.5486 ms ms
Speedup: 0.9003061368869223

iteracion = 4
Benchmark CostoRiegoFinca long = 8
Secuencial: 14.0466 ms ms
Paralelo: 15.0882 ms ms
Speedup: 0.9309659203881179

iteracion = 5
Benchmark CostoRiegoFinca long = 8
Secuencial: 15.0975 ms ms
Paralelo: 16.052 ms ms
Speedup: 0.9405370047346125

iteracion = 6
Benchmark CostoRiegoFinca long = 8
Secuencial: 14.0312 ms ms
Paralelo: 15.1206 ms ms
Speedup: 0.9279525944737643

iteracion = 7
Benchmark CostoRiegoFinca long = 8
Secuencial: 14.3676 ms ms
Paralelo: 15.4741 ms ms
Speedup: 0.9284934180340052

iteracion = 8
Benchmark CostoRiegoFinca long = 8
Secuencial: 14.6671 ms ms
Paralelo: 14.4732 ms ms
Speedup: 1.0133971754691429

iteracion = 9
Benchmark CostoRiegoFinca long = 8
Secuencial: 14.1305 ms ms
Paralelo: 14.4584 ms ms
Speedup: 0.97732114203508

iteracion = 10
Benchmark CostoRiegoFinca long = 8
Secuencial: 14.0356 ms ms
Paralelo: 14.417 ms ms
Speedup: 0.9735451203440383

iteracion = 1
Benchmark CostoRiegoFinca long = 7
Secuencial: 11.1794 ms ms
Paralelo: 10.0928 ms ms
Speedup: 1.1076609067850347

iteracion = 2
Benchmark CostoRiegoFinca long = 7
Secuencial: 2.8049 ms ms
Paralelo: 2.9974 ms ms
Speedup: 0.9357776739841196

iteracion = 3
Benchmark CostoRiegoFinca long = 7
Secuencial: 1.9256 ms ms
Paralelo: 3.0395 ms ms
Speedup: 0.6335252508636289

iteracion = 4
Benchmark CostoRiegoFinca long = 7
Secuencial: 2.4472 ms ms
Paralelo: 2.6456 ms ms
Speedup: 0.9250075597218023

iteracion = 5
Benchmark CostoRiegoFinca long = 7
Secuencial: 1.8259 ms ms
Paralelo: 2.6946 ms ms
Speedup: 0.6776144882357308

iteracion = 6
Benchmark CostoRiegoFinca long = 7
Secuencial: 2.3077 ms ms
Paralelo: 2.3529 ms ms
Speedup: 0.9807896638191169

iteracion = 7
Benchmark CostoRiegoFinca long = 7
Secuencial: 1.9244 ms ms
Paralelo: 2.6911 ms ms
Speedup: 0.7150979153506002

iteracion = 8
Benchmark CostoRiegoFinca long = 7
Secuencial: 2.1004 ms ms
Paralelo: 2.3283 ms ms
Speedup: 0.9021174247304901

iteracion = 9
Benchmark CostoRiegoFinca long = 7
Secuencial: 1.9433 ms ms
Paralelo: 2.075 ms ms
Speedup: 0.9365301204819276

iteracion = 10
Benchmark CostoRiegoFinca long = 7
Secuencial: 1.7573 ms ms
Paralelo: 2.0887 ms ms
Speedup: 0.8413367166179921

iteracion = 1
Benchmark CostoRiegoFinca long = 6
Secuencial: 1.0423 ms ms
Paralelo: 1.4207 ms ms
Speedup: 0.733652424860984

iteracion = 2
Benchmark CostoRiegoFinca long = 6
Secuencial: 0.5821 ms ms
Paralelo: 1.5972 ms ms
Speedup: 0.364450288004007

iteracion = 3
Benchmark CostoRiegoFinca long = 6
Secuencial: 0.8994 ms ms
Paralelo: 1.1368 ms ms
Speedup: 0.7911681914144968

iteracion = 4
Benchmark CostoRiegoFinca long = 6
Secuencial: 0.667 ms ms
Paralelo: 1.4573 ms ms
Speedup: 0.45769573869484664

iteracion = 5
Benchmark CostoRiegoFinca long = 6
Secuencial: 0.6457 ms ms
Paralelo: 1.77 ms ms
Speedup: 0.36480225988700565

iteracion = 6
Benchmark CostoRiegoFinca long = 6
Secuencial: 0.9095 ms ms
Paralelo: 1.1049 ms ms
Speedup: 0.8231514164177753

iteracion = 7
Benchmark CostoRiegoFinca long = 6
Secuencial: 0.6753 ms ms
Paralelo: 1.0812 ms ms
Speedup: 0.6245837957824639

iteracion = 8
Benchmark CostoRiegoFinca long = 6
Secuencial: 0.6558 ms ms
Paralelo: 0.9312 ms ms
Speedup: 0.7042525773195877

iteracion = 9
Benchmark CostoRiegoFinca long = 6
Secuencial: 0.4244 ms ms
Paralelo: 0.7888 ms ms
Speedup: 0.5380324543610548

iteracion = 10
Benchmark CostoRiegoFinca long = 6
Secuencial: 0.3981 ms ms
Paralelo: 0.587 ms ms
Speedup: 0.6781942078364566

iteracion = 1
Benchmark CostoRiegoFinca long = 5
Secuencial: 0.5092 ms ms
Paralelo: 1.1468 ms ms
Speedup: 0.4440181374258807

iteracion = 2
Benchmark CostoRiegoFinca long = 5
Secuencial: 0.2725 ms ms
Paralelo: 0.8623 ms ms
Speedup: 0.3160153078974835

iteracion = 3
Benchmark CostoRiegoFinca long = 5
Secuencial: 0.1466 ms ms
Paralelo: 1.0472 ms ms
Speedup: 0.1399923605805959

iteracion = 4
Benchmark CostoRiegoFinca long = 5
Secuencial: 0.1454 ms ms
Paralelo: 0.7072 ms ms
Speedup: 0.2055995475113122

iteracion = 5
Benchmark CostoRiegoFinca long = 5
Secuencial: 0.1108 ms ms
Paralelo: 0.6747 ms ms
Speedup: 0.16422113531940122

iteracion = 6
Benchmark CostoRiegoFinca long = 5
Secuencial: 0.0919 ms ms
Paralelo: 0.7338 ms ms
Speedup: 0.12523848460070863

iteracion = 7
Benchmark CostoRiegoFinca long = 5
Secuencial: 0.1838 ms ms
Paralelo: 0.7528 ms ms
Speedup: 0.24415515409139213

iteracion = 8
Benchmark CostoRiegoFinca long = 5
Secuencial: 0.0975 ms ms
Paralelo: 0.6877 ms ms
Speedup: 0.14177693761814747

iteracion = 9
Benchmark CostoRiegoFinca long = 5
Secuencial: 0.1505 ms ms
Paralelo: 0.4548 ms ms
Speedup: 0.33091468777484606

iteracion = 10
Benchmark CostoRiegoFinca long = 5
Secuencial: 0.0685 ms ms
Paralelo: 0.3584 ms ms
Speedup: 0.19112723214285715

iteracion = 1
Benchmark CostoRiegoFinca long = 4
Secuencial: 0.3269 ms ms
Paralelo: 0.9704 ms ms
Speedup: 0.33687139323990106

iteracion = 2
Benchmark CostoRiegoFinca long = 4
Secuencial: 0.2392 ms ms
Paralelo: 0.8939 ms ms
Speedup: 0.2675914531826826

iteracion = 3
Benchmark CostoRiegoFinca long = 4
Secuencial: 0.1772 ms ms
Paralelo: 0.6976 ms ms
Speedup: 0.2540137614678899

iteracion = 4
Benchmark CostoRiegoFinca long = 4
Secuencial: 0.186 ms ms
Paralelo: 0.5507 ms ms
Speedup: 0.33775195206101327

iteracion = 5
Benchmark CostoRiegoFinca long = 4
Secuencial: 0.1809 ms ms
Paralelo: 0.9974 ms ms
Speedup: 0.18137156607178667

iteracion = 6
Benchmark CostoRiegoFinca long = 4
Secuencial: 0.0954 ms ms
Paralelo: 0.604 ms ms
Speedup: 0.15794701986754967

iteracion = 7
Benchmark CostoRiegoFinca long = 4
Secuencial: 0.0557 ms ms
Paralelo: 0.5525 ms ms
Speedup: 0.10081447963800905

iteracion = 8
Benchmark CostoRiegoFinca long = 4
Secuencial: 0.055 ms ms
Paralelo: 0.4089 ms ms
Speedup: 0.13450721447786745

iteracion = 9
Benchmark CostoRiegoFinca long = 4
Secuencial: 0.0744 ms ms
Paralelo: 0.2546 ms ms
Speedup: 0.2922230950510605

iteracion = 10
Benchmark CostoRiegoFinca long = 4
Secuencial: 0.0559 ms ms
Paralelo: 0.2777 ms ms
Speedup: 0.20129636298163486

2.3 costoMovilidadPar

```
// Función Palelizada
def costoMovilidadPar( f : Finca , pi : ProgRiego , d: Distancia) : Int = {
  // Calcula el costo de movilidad de manera paralela
  (0 until pi.length - 1).par.map(j => d(pi(j))(pi(j + 1))).sum
}
```

Ley de Amdahl

$$S = 1/(1 - P) + P/N$$

$P = 0,85$ //Puesto que la paralelización está aplicada para que se aplique a todas los valore del map, menos a la suma final

$N = 8$ //Cantidad estimada de procesadores

$$S = 1/(1 - 0,85) + 0,85/8$$

$$S = 3,9$$

<p>iteracion = 1 Benchmark CostoMovilidad long = 8 Secuencial: 13.9167 ms ms Paralelo: 14.536 ms ms Speedup: 0.9573954320388201</p>	<p>iteracion = 1 Benchmark CostoMovilidad long = 7 Secuencial: 1.8403 ms ms Paralelo: 2.0022 ms ms Speedup: 0.9191389471581259</p>	<p>iteracion = 1 Benchmark CostoMovilidad long = 6 Secuencial: 0.3779 ms ms Paralelo: 0.6863 ms ms Speedup: 0.5506338336004662</p>	<p>iteracion = 1 Benchmark CostoMovilidad long = 5 Secuencial: 0.0792 ms ms Paralelo: 0.3378 ms ms Speedup: 0.2344582593250444</p>	<p>iteracion = 1 Benchmark CostoMovilidad long = 4 Secuencial: 0.0374 ms ms Paralelo: 0.5795 ms ms Speedup: 0.0645383951682485</p>
<p>iteracion = 2 Benchmark CostoMovilidad long = 8 Secuencial: 14.1418 ms ms Paralelo: 14.6165 ms ms Speedup: 0.9675230048233161</p>	<p>iteracion = 2 Benchmark CostoMovilidad long = 7 Secuencial: 1.9367 ms ms Paralelo: 2.6923 ms ms Speedup: 0.719347769565056</p>	<p>iteracion = 2 Benchmark CostoMovilidad long = 6 Secuencial: 0.2984 ms ms Paralelo: 0.881 ms ms Speedup: 0.33870601589103294</p>	<p>iteracion = 2 Benchmark CostoMovilidad long = 5 Secuencial: 0.0757 ms ms Paralelo: 0.3377 ms ms Speedup: 0.22416345869114598</p>	<p>iteracion = 2 Benchmark CostoMovilidad long = 4 Secuencial: 0.0374 ms ms Paralelo: 0.2169 ms ms Speedup: 0.22037805440295066</p>
<p>iteracion = 3 Benchmark CostoMovilidad long = 8 Secuencial: 16.0189 ms ms Paralelo: 14.8292 ms ms Speedup: 1.0802268497289131</p>	<p>iteracion = 3 Benchmark CostoMovilidad long = 7 Secuencial: 1.908 ms ms Paralelo: 2.199 ms ms Speedup: 0.8676671214188267</p>	<p>iteracion = 3 Benchmark CostoMovilidad long = 6 Secuencial: 0.3011 ms ms Paralelo: 0.6799 ms ms Speedup: 0.44285924400647153</p>	<p>iteracion = 3 Benchmark CostoMovilidad long = 5 Secuencial: 0.0706 ms ms Paralelo: 0.4377 ms ms Speedup: 0.16129769248343614</p>	<p>iteracion = 3 Benchmark CostoMovilidad long = 4 Secuencial: 0.0757 ms ms Paralelo: 0.2277 ms ms Speedup: 0.33245498462889767</p>
<p>iteracion = 4 Benchmark CostoMovilidad long = 8 Secuencial: 13.9767 ms ms Paralelo: 14.2135 ms ms Speedup: 0.9833397826010483</p>	<p>iteracion = 4 Benchmark CostoMovilidad long = 7 Secuencial: 1.8141 ms ms Paralelo: 2.0063 ms ms Speedup: 0.9042017644420077</p>	<p>iteracion = 4 Benchmark CostoMovilidad long = 6 Secuencial: 0.4223 ms ms Paralelo: 0.4269 ms ms Speedup: 0.9892246427734833</p>	<p>iteracion = 4 Benchmark CostoMovilidad long = 5 Secuencial: 0.0906 ms ms Paralelo: 0.3352 ms ms Speedup: 0.2702863961813842</p>	<p>iteracion = 4 Benchmark CostoMovilidad long = 4 Secuencial: 0.0278 ms ms Paralelo: 0.217 ms ms Speedup: 0.12811059907834102</p>
<p>iteracion = 5 Benchmark CostoMovilidad long = 8 Secuencial: 14.23 ms ms Paralelo: 14.8553 ms ms Speedup: 0.9579072788836308</p>	<p>iteracion = 5 Benchmark CostoMovilidad long = 7 Secuencial: 1.7919 ms ms Paralelo: 2.346 ms ms Speedup: 0.7638107416879796</p>	<p>iteracion = 5 Benchmark CostoMovilidad long = 6 Secuencial: 0.2835 ms ms Paralelo: 0.7993 ms ms Speedup: 0.3546853496809708</p>	<p>iteracion = 5 Benchmark CostoMovilidad long = 5 Secuencial: 0.0838 ms ms Paralelo: 0.3218 ms ms Speedup: 0.26041019266625237</p>	<p>iteracion = 5 Benchmark CostoMovilidad long = 4 Secuencial: 0.035 ms ms Paralelo: 0.1783 ms ms Speedup: 0.19629837352776222</p>
<p>iteracion = 6 Benchmark CostoMovilidad long = 8 Secuencial: 14.7071 ms ms Paralelo: 14.2413 ms ms Speedup: 1.032707688202622</p>	<p>iteracion = 6 Benchmark CostoMovilidad long = 7 Secuencial: 1.7257 ms ms Paralelo: 1.9556 ms ms Speedup: 0.8824401718142769</p>	<p>iteracion = 6 Benchmark CostoMovilidad long = 6 Secuencial: 0.2643 ms ms Paralelo: 0.8042 ms ms Speedup: 0.3286495896543148</p>	<p>iteracion = 6 Benchmark CostoMovilidad long = 5 Secuencial: 0.0785 ms ms Paralelo: 0.4659 ms ms Speedup: 0.16849109250912214</p>	<p>iteracion = 6 Benchmark CostoMovilidad long = 4 Secuencial: 0.034 ms ms Paralelo: 0.4533 ms ms Speedup: 0.07500551511140525</p>
<p>iteracion = 7 Benchmark CostoMovilidad long = 8 Secuencial: 14.7228 ms ms Paralelo: 15.0676 ms ms Speedup: 0.9771164618121001</p>	<p>iteracion = 7 Benchmark CostoMovilidad long = 7 Secuencial: 1.7538 ms ms Paralelo: 2.1021 ms ms Speedup: 0.8343085485942628</p>	<p>iteracion = 7 Benchmark CostoMovilidad long = 6 Secuencial: 0.3945 ms ms Paralelo: 1.0062 ms ms Speedup: 0.3920691711389386</p>	<p>iteracion = 7 Benchmark CostoMovilidad long = 5 Secuencial: 0.0678 ms ms Paralelo: 0.2788 ms ms Speedup: 0.24318507890961263</p>	<p>iteracion = 7 Benchmark CostoMovilidad long = 4 Secuencial: 0.0356 ms ms Paralelo: 0.1264 ms ms Speedup: 0.2816455696202531</p>
<p>iteracion = 8 Benchmark CostoMovilidad long = 8 Secuencial: 15.474 ms ms Paralelo: 14.8171 ms ms Speedup: 1.0443339114941521</p>	<p>iteracion = 8 Benchmark CostoMovilidad long = 7 Secuencial: 1.8411 ms ms Paralelo: 2.4859 ms ms Speedup: 0.7406170803330785</p>	<p>iteracion = 8 Benchmark CostoMovilidad long = 6 Secuencial: 0.308 ms ms Paralelo: 0.8417 ms ms Speedup: 0.36592610193655695</p>	<p>iteracion = 8 Benchmark CostoMovilidad long = 5 Secuencial: 0.0995 ms ms Paralelo: 0.3695 ms ms Speedup: 0.2692828146143437</p>	<p>iteracion = 8 Benchmark CostoMovilidad long = 4 Secuencial: 0.0383 ms ms Paralelo: 0.1328 ms ms Speedup: 0.2884036144578313</p>
<p>iteracion = 9 Benchmark CostoMovilidad long = 8 Secuencial: 14.4095 ms ms Paralelo: 14.655 ms ms Speedup: 0.9832480382122143</p>	<p>iteracion = 9 Benchmark CostoMovilidad long = 7 Secuencial: 1.8739 ms ms Paralelo: 2.0717 ms ms Speedup: 0.9045228556258146</p>	<p>iteracion = 9 Benchmark CostoMovilidad long = 6 Secuencial: 0.554 ms ms Paralelo: 0.8317 ms ms Speedup: 0.6661055669111459</p>	<p>iteracion = 9 Benchmark CostoMovilidad long = 5 Secuencial: 0.0727 ms ms Paralelo: 0.4264 ms ms Speedup: 0.1704971857410882</p>	<p>iteracion = 9 Benchmark CostoMovilidad long = 4 Secuencial: 0.032 ms ms Paralelo: 0.1434 ms ms Speedup: 0.22315202231520223</p>
<p>iteracion = 10 Benchmark CostoMovilidad long = 8 Secuencial: 15.065 ms ms Paralelo: 16.3542 ms ms Speedup: 0.9211700969781463</p>	<p>iteracion = 10 Benchmark CostoMovilidad long = 7 Secuencial: 1.831 ms ms Paralelo: 2.0839 ms ms Speedup: 0.8786410096453765</p>	<p>iteracion = 10 Benchmark CostoMovilidad long = 6 Secuencial: 0.3584 ms ms Paralelo: 0.9205 ms ms Speedup: 0.38935361216730036</p>	<p>iteracion = 10 Benchmark CostoMovilidad long = 5 Secuencial: 0.1041 ms ms Paralelo: 0.3278 ms ms Speedup: 0.31757169005491154</p>	<p>iteracion = 10 Benchmark CostoMovilidad long = 4 Secuencial: 0.0415 ms ms Paralelo: 0.1487 ms ms Speedup: 0.27908540685944855</p>

2.4 ProgramacionRiegoOptimoPar

```
// Función Paralelizada
def ProgramacionRiegoOptimoPar( f : Finca , d: Distancia) : (ProgRiego , Int) = {
  // Dada una finca, calcula la programación óptima de riego
  val programaciones = generarProgramacionesRiegoPar(f)
  val costos = programaciones.par.map(pi =>
    (pi , costoRiegoFincaPar(f , pi) + costoMovilidadPar(f, pi ,d))
  )
  costos.minBy(_._2)
}
```

Ley de Amdahl

$$S = 1/(1 - P) + P/N$$

$P = 0,85$ //Puesto que la paralelización está aplicada para que se aplique a todas los valore del map, menos al minBy final

$N = 8$ //Cantidad estimada de procesadores

$S = 1/(1 - 0,85) + 0,85/8$

$S = 3,9$

```
iteracion = 1
Benchmark RiegoOptimo long = 4
Secuencial: 0.2263 ms ms
Paralelo: 1.2606 ms ms
Speedup: 0.1795176899888942
```

```
iteracion = 2
Benchmark RiegoOptimo long = 4
Secuencial: 0.2293 ms ms
Paralelo: 1.4882 ms ms
Speedup: 0.15407875285579894
```

```
iteracion = 3
Benchmark RiegoOptimo long = 4
Secuencial: 0.1738 ms ms
Paralelo: 1.4766 ms ms
Speedup: 0.11770283082757688
```

```
iteracion = 4
Benchmark RiegoOptimo long = 4
Secuencial: 0.325 ms ms
Paralelo: 1.0811 ms ms
Speedup: 0.3006197391545648
```

```
iteracion = 5
Benchmark RiegoOptimo long = 4
Secuencial: 0.2676 ms ms
Paralelo: 1.3102 ms ms
Speedup: 0.2042436269271867
```

```
iteracion = 6
Benchmark RiegoOptimo long = 4
Secuencial: 0.1487 ms ms
Paralelo: 0.9079 ms ms
Speedup: 0.16378455777067957
```

```
iteracion = 7
Benchmark RiegoOptimo long = 4
Secuencial: 0.1793 ms ms
Paralelo: 0.7701 ms ms
Speedup: 0.23282690559667574
```

```
iteracion = 8
Benchmark RiegoOptimo long = 4
Secuencial: 0.097 ms ms
Paralelo: 1.0645 ms ms
Speedup: 0.09112259276655707
```

```
iteracion = 9
Benchmark RiegoOptimo long = 4
Secuencial: 0.1054 ms ms
Paralelo: 0.5958 ms ms
Speedup: 0.17690500167841558
```

```
iteracion = 10
Benchmark RiegoOptimo long = 4
Secuencial: 0.1747 ms ms
Paralelo: 0.6807 ms ms
Speedup: 0.2566475686793007
```


iteracion = 1
Benchmark RiegoOptimo long = 8
Secuencial: 59.1319 ms ms
Paralelo: 219.8043 ms ms
Speedup: 0.2690206697503188

iteracion = 2
Benchmark RiegoOptimo long = 8
Secuencial: 57.5784 ms ms
Paralelo: 103.6528 ms ms
Speedup: 0.5554929533982681

iteracion = 3
Benchmark RiegoOptimo long = 8
Secuencial: 52.109 ms ms
Paralelo: 208.9087 ms ms
Speedup: 0.24943432226613826

iteracion = 4
Benchmark RiegoOptimo long = 8
Secuencial: 52.4058 ms ms
Paralelo: 174.4949 ms ms
Speedup: 0.3003285482842192

iteracion = 5
Benchmark RiegoOptimo long = 8
Secuencial: 57.5649 ms ms
Paralelo: 105.7911 ms ms
Speedup: 0.5441374557973213

iteracion = 6
Benchmark RiegoOptimo long = 8
Secuencial: 65.269 ms ms
Paralelo: 143.7425 ms ms
Speedup: 0.4540689079430231

iteracion = 7
Benchmark RiegoOptimo long = 8
Secuencial: 55.3874 ms ms
Paralelo: 182.1266 ms ms
Speedup: 0.30411483001384754

iteracion = 8
Benchmark RiegoOptimo long = 8
Secuencial: 58.3153 ms ms
Paralelo: 143.6895 ms ms
Speedup: 0.4058424589131426

iteracion = 9
Benchmark RiegoOptimo long = 8
Secuencial: 51.7435 ms ms
Paralelo: 201.8831 ms ms
Speedup: 0.25630426717243787

iteracion = 10
Benchmark RiegoOptimo long = 8
Secuencial: 58.8667 ms ms
Paralelo: 182.461 ms ms
Speedup: 0.3226262050520385

iteracion = 1
Benchmark RiegoOptimo long = 7
Secuencial: 6.7962 ms ms
Paralelo: 18.7602 ms ms
Speedup: 0.3622669267918252

iteracion = 2
Benchmark RiegoOptimo long = 7
Secuencial: 6.9555 ms ms
Paralelo: 13.5748 ms ms
Speedup: 0.5123832395320741

iteracion = 3
Benchmark RiegoOptimo long = 7
Secuencial: 6.364 ms ms
Paralelo: 27.4345 ms ms
Speedup: 0.23197069383440558

iteracion = 4
Benchmark RiegoOptimo long = 7
Secuencial: 6.3036 ms ms
Paralelo: 17.9845 ms ms
Speedup: 0.3505018210125386

iteracion = 5
Benchmark RiegoOptimo long = 7
Secuencial: 6.5769 ms ms
Paralelo: 20.6434 ms ms
Speedup: 0.3185957739519653

iteracion = 6
Benchmark RiegoOptimo long = 7
Secuencial: 6.3379 ms ms
Paralelo: 15.9026 ms ms
Speedup: 0.3985448920302341

iteracion = 7
Benchmark RiegoOptimo long = 7
Secuencial: 6.4265 ms ms
Paralelo: 15.0907 ms ms
Speedup: 0.42585831008501923

iteracion = 8
Benchmark RiegoOptimo long = 7
Secuencial: 7.6693 ms ms
Paralelo: 12.9858 ms ms
Speedup: 0.5905912612237983

iteracion = 9
Benchmark RiegoOptimo long = 7
Secuencial: 6.3301 ms ms
Paralelo: 12.6128 ms ms
Speedup: 0.5018790435113535

iteracion = 10
Benchmark RiegoOptimo long = 7
Secuencial: 7.6726 ms ms
Paralelo: 18.9527 ms ms
Speedup: 0.404828863433706

iteracion = 1
Benchmark RiegoOptimo long = 6
Secuencial: 1.1009 ms ms
Paralelo: 8.4714 ms ms
Speedup: 0.12995490709918078

iteracion = 2
Benchmark RiegoOptimo long = 6
Secuencial: 1.3455 ms ms
Paralelo: 3.7902 ms ms
Speedup: 0.35499445939528257

iteracion = 3
Benchmark RiegoOptimo long = 6
Secuencial: 1.0161 ms ms
Paralelo: 1.6569 ms ms
Speedup: 0.6132536664856056

iteracion = 4
Benchmark RiegoOptimo long = 6
Secuencial: 0.9904 ms ms
Paralelo: 1.8566 ms ms
Speedup: 0.5334482387159323

iteracion = 5
Benchmark RiegoOptimo long = 6
Secuencial: 0.9553 ms ms
Paralelo: 5.195 ms ms
Speedup: 0.183888354186718

iteracion = 6
Benchmark RiegoOptimo long = 6
Secuencial: 0.8831 ms ms
Paralelo: 1.9845 ms ms
Speedup: 0.4449987402368355

iteracion = 7
Benchmark RiegoOptimo long = 6
Secuencial: 0.7927 ms ms
Paralelo: 1.9277 ms ms
Speedup: 0.41121543808683925

iteracion = 8
Benchmark RiegoOptimo long = 6
Secuencial: 1.023 ms ms
Paralelo: 2.0948 ms ms
Speedup: 0.4883521099866335

iteracion = 9
Benchmark RiegoOptimo long = 6
Secuencial: 0.829 ms ms
Paralelo: 2.997 ms ms
Speedup: 0.2766099432766099

iteracion = 10
Benchmark RiegoOptimo long = 6
Secuencial: 0.8994 ms ms
Paralelo: 2.2826 ms ms
Speedup: 0.39402435818803117

iteracion = 1
Benchmark RiegoOptimo long = 5
Secuencial: 1.6572 ms ms
Paralelo: 2.5931 ms ms
Speedup: 0.6390806370753153

iteracion = 2
Benchmark RiegoOptimo long = 5
Secuencial: 0.5727 ms ms
Paralelo: 2.9467 ms ms
Speedup: 0.19435300505650388

iteracion = 3
Benchmark RiegoOptimo long = 5
Secuencial: 0.5665 ms ms
Paralelo: 1.7519 ms ms
Speedup: 0.32336320566242366

iteracion = 4
Benchmark RiegoOptimo long = 5
Secuencial: 0.5997 ms ms
Paralelo: 1.3614 ms ms
Speedup: 0.4405024239753195

iteracion = 5
Benchmark RiegoOptimo long = 5
Secuencial: 0.2873 ms ms
Paralelo: 1.643 ms ms
Speedup: 0.17486305538648814

iteracion = 6
Benchmark RiegoOptimo long = 5
Secuencial: 0.6656 ms ms
Paralelo: 2.5057 ms ms
Speedup: 0.26563435367362415

iteracion = 7
Benchmark RiegoOptimo long = 5
Secuencial: 0.3127 ms ms
Paralelo: 1.0346 ms ms
Speedup: 0.30224242125265803

iteracion = 8
Benchmark RiegoOptimo long = 5
Secuencial: 0.1959 ms ms
Paralelo: 1.5463 ms ms
Speedup: 0.12668951691133673

iteracion = 9
Benchmark RiegoOptimo long = 5
Secuencial: 0.1983 ms ms
Paralelo: 1.4247 ms ms
Speedup: 0.13918719730469573

iteracion = 10
Benchmark RiegoOptimo long = 5
Secuencial: 0.225 ms ms
Paralelo: 0.9065 ms ms
Speedup: 0.24820739106453393

3. Informe de corrección

Antes de revisar las funciones principales, miramos los generadores de entradas aleatorias, para la finca y para la matriz de distancia

3.1 fincaAlAzar

$$F = \langle T_0, \dots, T_{n-1} \rangle$$

$$T = \langle ts_i^F, tr_i^F, p_i^F \rangle$$

```
// Genera una finca al azar
def fincaAlAzar(long: Int) : Finca = {
  // Crea una finca de long tableros ,
  // con valores aleatorios entre 1 y long * 2 para el tiempo
  // de supervivencia , entre 1 y long para el tiempo
  // de regado y entre 1 y 4 para la prioridad
  val v = Vector.fill(long)(
    (random.nextInt(long*2) + 1 ,
    random.nextInt(long)+1 ,
    random.nextInt(4) + 1 )
  )
  v
}
```

```
v = Vector.fill(long)
(
  (random.nextInt(long * 2) + 1,    //  $ts_i^F$ 
  random.nextInt(long) + 1,        //  $tr_i^F$ 
  random.nextInt(4) + 1)           //  $p_i^F$ 
)
```

long = cantidad de Tableros en la finca

3.2 distanciaAlAzar

Matriz (*long* x *long*), diagonal de 0 e igual a su transpuesta

```
//Generar matriz de distancia al azar
def distanciaAlAzar(long: Int) : Distancia = {
  // Crea una matriz de distancias para una finca
  // de long tableros , con valores aleatorios entre
  // 1 y long * 3
  val v = Vector.fill(long ,long)( random.nextInt(Long*3) + 1 )
  Vector.tabulate( long , long) ( ( i , j ) =>
    if ( i < j ) v ( i ) ( j )
    else if ( i == j ) 0
    else v ( j ) ( i ) )
}
```

```
val v = Vector.fill(long ,long)(random.nextInt(long * 3) + 1)
Vector.tabulate( long , long) ((i, j) =>
  if (i < j) v(i)(j)
  else if (i == j) 0
  else v(j)(i))
```

```
else if (i == j) 0 // Diagonal de 0
Vector.tabulate( long , long) // long x long
```

3.3 tIR (Generador de tiempos de riego)

Dada una programación de riego Π , se puede calcular, para cada tablón T_i , el tiempo en que se iniciará su riego t_i^Π según:

$$t_{\pi_0}^\Pi = 0,$$

$$t_{\pi_j}^\Pi = t_{\pi_{j-1}}^\Pi + tr_{\pi_{j-1}}^F, \quad j = 1, \dots, n-1.$$

```
def tIR ( f : Finsa , pi : ProgRiego ) : TiempoInicioRiego = {
  // Dada una finca f y una programación de riego pi,
  // y f.length == n, tIR(f, pi) devuelve t: TiempoInicioRiego
  // tal que t(i) es el tiempo en que inicia el riego del
  // tablón i de la finca f según pi
  val tiempos = Array.fill( f.length)(0)
  for (j <- 1 until pi.length) {
    val prevTablon = pi ( j - 1 )
    val currTablon = pi ( j )
    tiempos(currTablon) = tiempos( prevTablon ) + treg( f, prevTablon)
  }
  tiempos.toVector
}
```

```

val tiempos = Array.fill(f.length)(0)
for (j <- 1 until pi.length) {
  val prevTablon = pi(j - 1)
  val currTablon = pi(j)
  tiempos(currTablon) = tiempos(prevTablon) + treg(f, prevTablon)
}

```

val tiempos = Array.fill(f.length)(0) //Se crea un vector de 0 de longitud f (finca.length)
// Para cada tablón "j", se calcula el tiempo de inicio de riego con el valor de riego del tablón
y el tiempo de riego acumulado

3.4 generarProgramacionesRiego

f: FINCA -> Vector(Vector(Int))

```

def generarProgramacionesRiego(f:Finca):Vector[ProgRiego] = {
  // Dada una finca de n tablon , devuelve todas las
  // posibles programaciones de riego de la finca
  val indices = (0 until f.length).toVector
  indices.permutations.toVector
}

```

P_f :

f = Finca

val indices = (0 until f.length).toVector // Se crea el vector con los índices de los tablon
indices.permutations.toVector // Se crean todas las permutaciones de los índices

Caso Base:

Finca = (T₁, T₂)

Finca.length = 2

indices = Vector(0, 1)

indices.permutations.toVector = Vector(Vector(0, 1), Vector(1, 0))

Caso Inductivo:

Finca = (T₁, ..., T_n)

Finca.length = n

indices = Vector(0, 1, 2, ..., n - 1)

indices.permutations.toVector = Vector(Vector₁(0, 1, ..., n - 1), Vector₂(1, 2, ..., n - 1, 0),
....., Vector_{n!}(n - 1, ..., 0, 1, 2))

3.5 costoRiegoTablon

f :

El costo de riego de un tablón T_i de la finca F , dada una programación de riego Π , se define como:

$$CR_F^\Pi[i] = \begin{cases} ts_i^F - (t_i^\Pi + tr_i^F), & \text{si } ts_i^F - tr_i^F \geq t_i^\Pi, \\ p_i^F \cdot ((t_i^\Pi + tr_i^F) - ts_i^F), & \text{de lo contrario.} \end{cases}$$

```
//Costo de riego del tablon
def costoRiegoTablon( i : Int, f : Finca , pi : ProgRiego ) : Int = {
  val tiempoInicio = tIR(f , pi)(i)
  val tiempoFinal = tiempoInicio + treg ( f , i )

  // tiempo de supervivencia - tiempo de regado >= tiempo inicio riego
  if (tsup(f , i) - treg(f , i) >= tiempoInicio) {

    //ts - (ti + tr)
    tsup(f , i) - tiempoFinal
  } else {
    //pi * ((ti + tr) - ts)
    tprio(f , i) * (tiempoFinal - tsup(f , i))
  }
}
```

P_f :

```
val tiempoInicio = tIR(f , pi)(i)
val tiempoFinal = tiempoInicio + treg ( f , i )
```

```
if (tsup(f , i) - treg(f , i) >= tiempoInicio) {
  tsup(f , i) - tiempoFinal
} else {
  tprio(f , i) * (tiempoFinal - tsup(f , i)) }
```

$tiempoInicio$ = generar Tiempos de Inicio de Riego para cada tablón
 $tiempoFinal$ = $tiempoInicio$ + tiempo de regado del tablón actual

Caso Base:

```
Finca (  $T_1 = (3, 2, 4)$ ,  $T_2 = (1, 1, 4)$  )
 $tsup_1 = 3$ ,  $treg_1 = 2$ ,  $tprio_1 = 4$ 
 $tsup_2 = 1$ ,  $treg_2 = 1$ ,  $tprio_2 = 4$ 
ProgRiego = (0, 1)
tiempoInicio = (0 , 2)
```

$Costo T_1 =$
 $tsup(f, i) - treg(f, i) \geq tiempoInicio$
 $3 - 2 \geq 0 \quad //true$
 $Costo T_1 = tsup(f, i) - tiempoFinal$
 $Costo T_1 = 3 - (0 + 2) = 1$

Caso inductivo:

$Costo T_2 =$
 $tsup(f, i) - treg(f, i) \geq tiempoInicio$
 $1 - 1 \geq 2 \quad //false$
 $Costo T_2 = tprio(f, i) * (tiempoFinal - tsup(f, i))$
 $Costo T_2 = 4 * ((2 + 1) - 1) = 4 * 2 = 8$

3.6 costoRiegoFinca

$f: costoRiegoFinca = costoT_1 + costoT_2 + \dots + costoT_n$

```
//Costo de riego de una finca
def costoRiegoFinca(f : Finca , pi : ProgRiego ) :Int = {
    //Sumatoria de todos los costos de riego de cada tablon
    ( 0 until f.length) . map( i => costoRiegoTablon(i , f , pi)).sum
}
```

$P_f:$
 $(0 \text{ until } f.length). \text{map}(i \Rightarrow costoRiegoTablon(i, f, pi)).sum$
// Aplica a cada tablón un costo de tablon y luego los suma

Caso base:

$Finca (T_1 = (3, 2, 4), T_2 = (1, 1, 4))$
 $tsup_1 = 3, treg_1 = 2, tprio_1 = 4$
 $tsup_2 = 1, treg_2 = 1, tprio_2 = 4$
 $ProgRiego = (0, 1)$
 $tiempoInicio = (0, 2)$
 $Costo T_1 = 1$
 $Costo T_2 = 8$
 $CostoRiegoFinca = CT_1 + CT_2 = 1 + 8 = 9$

Caso inductivo:

$Finca (T_1, T_2, \dots, T_n)$

$ProgRiego = (0, 1, 2, \dots, n - 1)$

$tiempoInicio = (tIT_1, tIT_2, \dots, tIT_n)$

$CostoRiegoFinca = CT_1 + CT_2 + \dots + CT_n$

3.7 costoMovilidad

f :

El costo de movilidad del sistema móvil de riego está dado por una matriz de distancias D_F , donde $D_F[i, j]$ representa el costo de mover el sistema móvil de T_i a T_j . Este costo es proporcional a la distancia entre los tabloncillos. Para este ejercicio, se asume que $D_F[i, j] = D_F[j, i]$ y $D_F[i, i] = 0$.

El costo de movilidad para una programación Π se define como:

$$CM_F^\Pi = \sum_{j=0}^{n-2} D_F[\pi_j, \pi_{j+1}].$$

```
//Costo de movilidad
def costoMovilidad(f : Finca , pi : ProgRiego , d : Distancia) : Int = {
  ( 0 until pi.length - 1 ).map( j => d (pi( j ))(pi( j + 1 ))).sum
}
```

P_f :

$(0 \text{ until } pi.length - 1).map(j \Rightarrow d(pi(j))(pi(j + 1))).sum$

//Aplica para cada par (T_1, T_2) un valor en la matriz de distancia, y luego sumar todos los valores

Caso Base:

$distancia = \begin{pmatrix} 0 & 3 \\ 3 & 0 \end{pmatrix}$

$progRiego = (0, 1)$

$distancia (0, 1) = 3$

Caso inductivo:

$distancia = \begin{pmatrix} 0 & V(1, 0) & \dots & V(n, 0) \\ V(0, 1) & 0 & \dots & \dots \\ \dots & \dots & 0 & V(n, n - 1) \\ V(0, n) & \dots & V(n - 1, n) & 0 \end{pmatrix}$

$progRiego = (0, 1, 2, \dots, n)$

$\text{distancia}(0,1) = V(0,1)$

$\text{distancia}(1,2) = V(1,2)$

...

$\text{distancia}(n-1, n) = V(n-1, n)$

$\text{CostoMovilidad} = V(0,1) + V(1,2) + \dots + V(n-1, n)$

3.8 ProgramacionRiegoOptimo

f: Finca, distancia \rightarrow ProgramacionRiego mas baja

```
def ProgramacionRiegoOptimo(f : Finca , d :Distancia) : (ProgRiego, Int) =  
  // Dada una finca devuelve la programacion de riego optima  
  val programaciones = generarProgramacionesRiego(f)  
  val costos = programaciones.map( pi =>  
    ( pi, costoRiegoFinca(f , pi) + costoMovilidad(f , pi , d) )  
  )  
  costos.minBy(_._2)  
}
```

P_f:

val programaciones = generarProgramacionesRiego(f)

val costos = programaciones.map(pi =>

(pi , costoRiegoFincaPar(f , pi) + costoMovilidadPar(f, pi , d)))

costos.minBy(_._2)

Caso Inductivo:

val programaciones = generarProgramacionesRiegoPar(f)

//Genera todas las programaciones

val costos = programaciones.map(pi =>

(pi , costoRiegoFincaPar(f , pi) + costoMovilidadPar(f, pi , d)))

*//Aplica un map a cada programación, generando el costo de cada programación
en un nuevo vector*

costos.minBy(_._2) //Obtiene la programación con el costo mínimo o más bajo

4. Conclusiones

4.1 Presentación de Resultados de Benchmarks

4.1.1 Benchmark costoRiegoFinca

Tamaño de la finca (tablones)	Tiempo Secuencial (ms)	Tiempo Paralelizado (ms)	Aceleración (%)
7	8,906	8,178	0,175
8	55,338	9,347	83.109
9	68,259	71,634	-4,929

4.1.2 Benchmark costoMovilidad

Tamaño de la finca (tablones)	Tiempo Secuencial (ms)	Tiempo Paralelizado (ms)	Aceleración (%)
7	6,154	1,454	76,372
8	10,818	8,495	21,473
9	68,002	71,147	-4,615

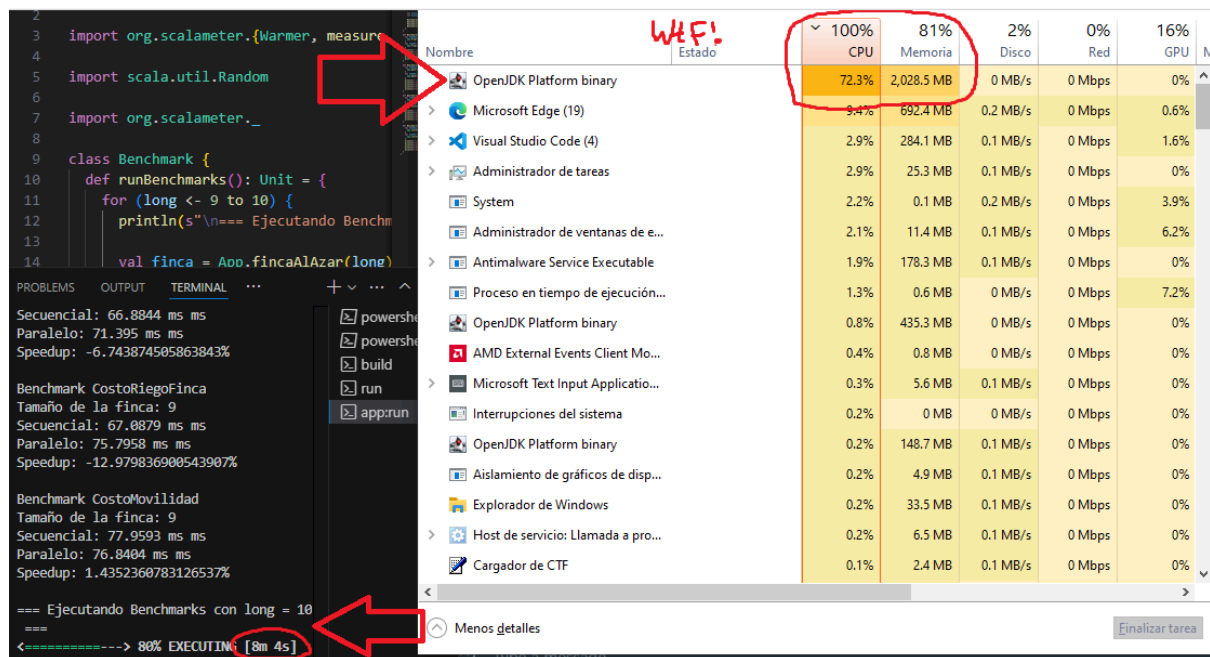
4.1.3 Benchmark generarProgramacionRiego

Tamaño de la finca (tablones)	Tiempo Secuencial (ms)	Tiempo Paralelizado (ms)	Aceleración (%)
7	1,329	1,318	0,797
8	11,900	7,468	37,253
9	82,960	69,251	16,524

4.1.4 Benchmark riegoOptimo

Tamaño de la finca (tablones)	Tiempo Secuencial (ms)	Tiempo Paralelizado (ms)	Aceleración (%)
7	19,346	21,993	-13,683
8	54,252	152,699	-181,461
9	547,748	1342,500	-145,094

4.1.5 Intento Benchmark Tablones Mayores a 10



4.2 Análisis y Conclusiones

4.2.1 Tamaños de fincas donde el paralelismo genera ganancias significativas

- **costoRiegoFinca:**

En el tamaño de 8 tablonos, la paralelización muestra una aceleración significativa del 83.109%. Este es el único caso con una ganancia notable, mientras que para 7 y 9 tablonos el paralelismo no es eficiente (aceleración cercana a cero o negativa).

- **costoMovilidad:**

En los tamaños de 7 y 8 tablonos, el paralelismo genera una aceleración del 76.372% y 21.473% respectivamente. Sin embargo, para 9 tablonos el rendimiento cae, con un impacto negativo del -4.615%.

- **generarProgramacionesRiego:**

La paralelización es más efectiva en tamaños intermedios como 8 tablonos, con una aceleración del 37.253%, mientras que para 9 tablonos la aceleración es menor (16.524%) y casi inexistente en 7 tablonos (0.797%).

- **riegoOptimo:**

El paralelismo no es beneficioso en ningún tamaño. En todos los casos (7, 8 y 9 tablonos), la versión paralelizada introduce una sobrecarga significativa, con desaceleraciones que van desde -13.683% hasta -181.461%.

4.2.2 ¿Las versiones paralelas introducen sobrecarga en casos pequeños?

Sí, en muchos casos las versiones paralelas tienen un tiempo mayor que las secuenciales para tamaños pequeños. Esto puede deberse a la sobrecarga de creación de tareas paralelas: En tamaños pequeños, la gestión del paralelismo (creación de threads, comunicación entre núcleos, etc.) toma más tiempo que la ejecución secuencial directa. También funciones con computación poco costosa: Si la operación no es intensiva en cálculos, el costo de coordinar las tareas paralelas supera cualquier beneficio.

Casos claros donde se introduce sobrecarga:

- **costoRiegoFinca (7 tablonos):** La aceleración es apenas del 0.175%, lo que sugiere que el paralelismo no aporta valor en este caso.
- **costoMovilidad (9 tablonos):** El paralelismo es ineficiente y genera una pérdida del -4.615%.
- **riegoOptimo (todos los tamaños):** El paralelismo introduce una sobrecarga significativa en todos los casos analizados.

4.2.3. Beneficios del paralelismo para diferentes escenarios

Los escenarios donde el paralelismo presenta beneficios surgen en funciones con gran cantidad de cálculos independientes, como costoRiegoFinca en fincas medianas (8 tablonos). Aquí, el paralelismo aprovecha la división de trabajo entre núcleos para acelerar los cálculos. En funciones con altas cargas computacionales, como generarProgramacionRiego, muestran beneficios en tamaños medianos y grandes, donde la división de tareas compensa la sobrecarga inicial.

Por el contrario, los escenarios donde el paralelismo no es beneficioso se presentan en funciones con tamaños pequeños o con baja complejidad computacional, como costoMovilidad para fincas pequeñas (7 tablonos). También en operaciones que involucran gran coordinación o comunicación entre tareas,

como `riegoOptimo`, donde la sobrecarga de paralelización supera los beneficios en cualquier tamaño analizado.

4.2.4 Conclusión general del proyecto

Este proyecto muestra claramente las ventajas y limitaciones del paralelismo en programación funcional. Algunos puntos clave son:

- **Ganancias significativas en ciertos casos:** El paralelismo es más efectivo cuando se trabaja con datos grandes y operaciones intensivas en cálculos independientes. Esto se refleja en funciones como `costoRiegoFinca` y `generarProgramacionRiego` para tamaños intermedios.
- **Sobrecarga en operaciones pequeñas o coordinadas:** En operaciones con tamaños de entrada pequeños o donde las tareas están fuertemente interdependientes, la sobrecarga del paralelismo puede resultar en tiempos peores que los secuenciales, como en `riegoOptimo`.
- **Eficiencia del paralelismo:** Este proyecto demuestra que la eficiencia del paralelismo depende del balance entre el costo de la tarea y el costo de coordinar las tareas paralelas. Aplicar paralelismo indiscriminadamente puede ser contraproducente.
- **Lecciones de relevancia:** Es crucial analizar la naturaleza del problema antes de decidir paralelizar una función. No todas las operaciones se benefician del paralelismo. También, el realizar benchmarks exhaustivos como los realizados en este proyecto ayuda a identificar los escenarios donde el paralelismo aporta un valor real.

En resumen, el paralelismo es una herramienta poderosa cuando se aplica correctamente, pero puede generar sobrecarga significativa si no se usa con cuidado. Este proyecto resalta la importancia del análisis de rendimiento para optimizar el diseño y la implementación de soluciones funcionales y concurrentes.