



Proyecto Programación Funcional y Concurrente (Documentación)

Edwar Yamir Forero Blanco (2259664)

Faber Alexis Solis Gamboa (2259714)

Santiago Aníbal Carrillo Torres (2259465)

Programación Funcional y Concurrente

Carlos Delgado

Universidad del Valle

Tuluá, Valle del Cauca

Junio 2024

Descripción de la solución.

Link Git Hub: <https://github.com/Edwar-Forero/pfc-ayuda-proyecto-2024-I.git>

Función Itinerarios:

Organizar los vuelos e itinerarios desde un lugar de salida hacia un destino, posee cierto nivel de complejidad. Para ello se usa un algoritmo de búsqueda, llamado 'Depth First Search' el cual requiere un nodo inicial que se identifica como 'ORG', luego de ello, el algoritmo inicia la visita de todos los nodos que tienen como destino el vuelo actual.

Por esta razón el **algoritmo de búsqueda, por sí solo**, no se decidió paralelizar, ya que requiere de una secuencia, la cual se va dando con la listas de vuelos de salida y destino. Lo cual entorpece la paralelización y no permite un correcto funcionamiento.

Esta primera función posee **paralelización de datos en 'dfs'**, la cual se encarga de organizar los vuelos que tienen como salida el aeropuerto especificado, para luego usar la función de búsqueda y organizar itinerarios. Es decir, que se realizará de forma paralela la generación de los itinerarios, más no, se paraleliza el algoritmo de búsqueda.

Además, se decidió generar otra **paralelización de datos en la operación reverse** sobre la lista especificada, con el fin de optimizar lo máximo posible. Se realiza solo la paralelización en esta operación, para así evitar que la sincronización de datos con la paralelización pasada no genere conflictos y empeore el resultado.

Función itinerarios tiempo:

En esta función se utilizó tanto **paralelización de tareas como de datos**. Primero, se divide la lista de itinerarios en dos mitades. Cada mitad se procesa por separado en **"parallelTiempoTotal"**, lo que permite un mejor rendimiento al ejecutar estas dos tareas de manera simultánea. Esto mejora la eficiencia y reduce el tiempo de procesamiento, especialmente cuando se trabaja con listas de itinerarios grandes. También se utiliza **paralelización de datos** para convertir la lista de itinerarios en una colección paralela. Esto permite que las operaciones sobre la lista de itinerarios, como el mapeo y la clasificación, se realicen en paralelo.

Función itinerariosEscalasPar:

En esta función se utilizó **paralelización tanto de tareas como de datos** para mejorar el rendimiento y la eficiencia. Primero, se divide la lista de itinerarios en dos mitades. Cada mitad se procesa por separado en la función “**calcularEscalasPar**”, que calcula las escalas de cada itinerario. Al ejecutar estas dos tareas de manera simultánea, se optimiza el uso de recursos y se reduce el tiempo de procesamiento, especialmente para listas grandes de itinerarios. **La paralelización de datos** se aplica para convertir la lista resultante en una colección paralela, lo que permite que operaciones como el mapeo y la clasificación se realicen en paralelo, acelerando así el procesamiento general.

Función itinerariosAirePar:

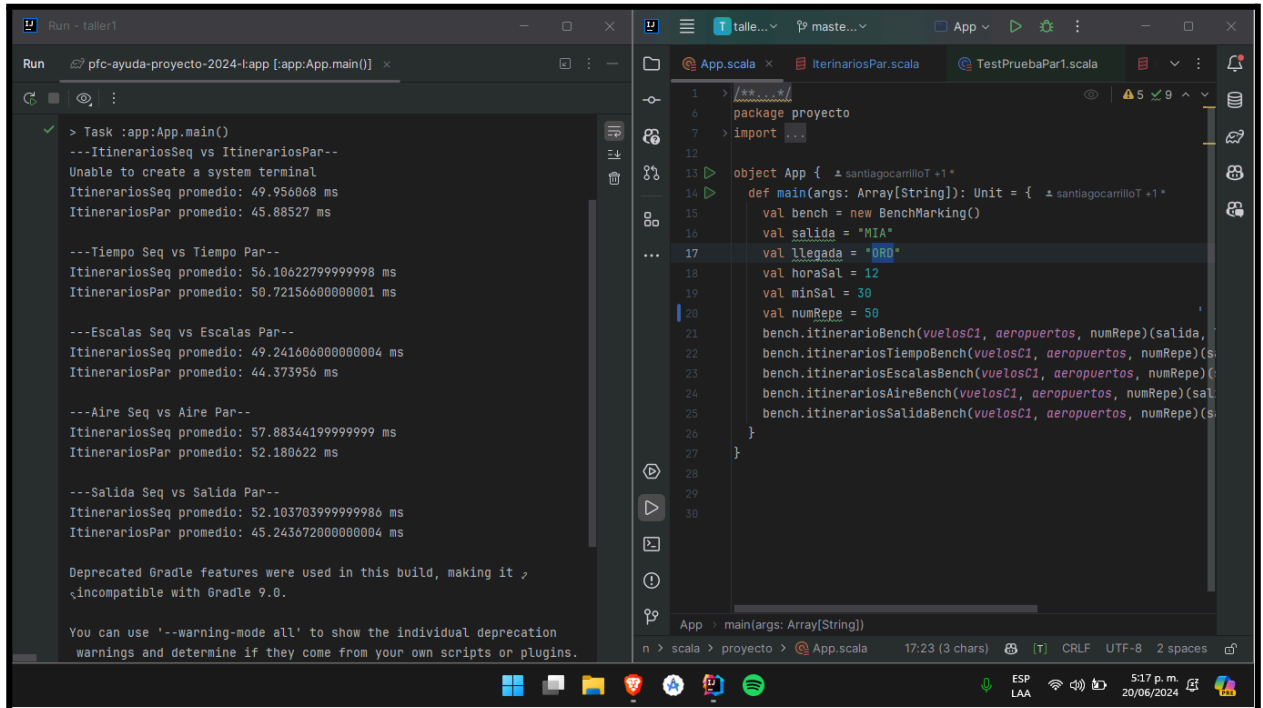
Esta función también se beneficia de la paralelización de tareas y datos. Primero, se divide la lista de itinerarios en dos mitades y cada mitad se procesa por separado en la función “**calculoAirePar**”, que calcula el tiempo total de vuelo de cada itinerario. Al dividir las tareas y ejecutarlas simultáneamente, se mejora significativamente la eficiencia y se reduce el tiempo de procesamiento, especialmente para listas de itinerarios grandes. **La paralelización de datos** se utiliza para transformar la lista resultante en una colección paralela, permitiendo que las operaciones de mapeo y clasificación se realicen en paralelo y mejorando así el rendimiento global de la función.

Función itinerariosSalidaPar: En esta función se aplica la paralelización de tareas y datos para calcular el itinerario que permite salir lo más tarde posible y llegar a tiempo a la cita. La lista de itinerarios se divide en dos mitades, y cada mitad se procesa por separado en la función “**calcularLlegadaPar**”, que calcula la diferencia entre la hora preferida de llegada y la hora real de llegada del último vuelo del itinerario. Al ejecutar estas tareas de manera simultánea, se mejora la eficiencia y se reduce el tiempo de procesamiento. Además, la paralelización de datos permite convertir la lista resultante en una colección paralela, acelerando las operaciones de mapeo y clasificación para obtener el mejor itinerario.

A continuación, se muestran 2 pruebas donde se compara el rendimiento de los dos métodos de solución (Secuencial y Paralelizado):

Benchmarking.

Prueba 1: “MIA”, “ORD” y 50 repeticiones para tomar el promedio de ejecución.



The screenshot displays an IDE with two main panels. The left panel, titled 'Run - taller1', shows the output of a benchmarking task. The right panel shows the source code of the application in Scala.

Run Console Output:

```
> Task :app:App.main()
--ItinerariosSeq vs ItinerariosPar--
Unable to create a system terminal
ItinerariosSeq promedio: 49.956068 ms
ItinerariosPar promedio: 45.88527 ms

---Tiempo Seq vs Tiempo Par--
ItinerariosSeq promedio: 56.186227999999998 ms
ItinerariosPar promedio: 50.721566000000001 ms

---Escalas Seq vs Escalas Par--
ItinerariosSeq promedio: 49.241606000000004 ms
ItinerariosPar promedio: 44.373956 ms

---Aire Seq vs Aire Par--
ItinerariosSeq promedio: 57.883441999999999 ms
ItinerariosPar promedio: 52.180622 ms

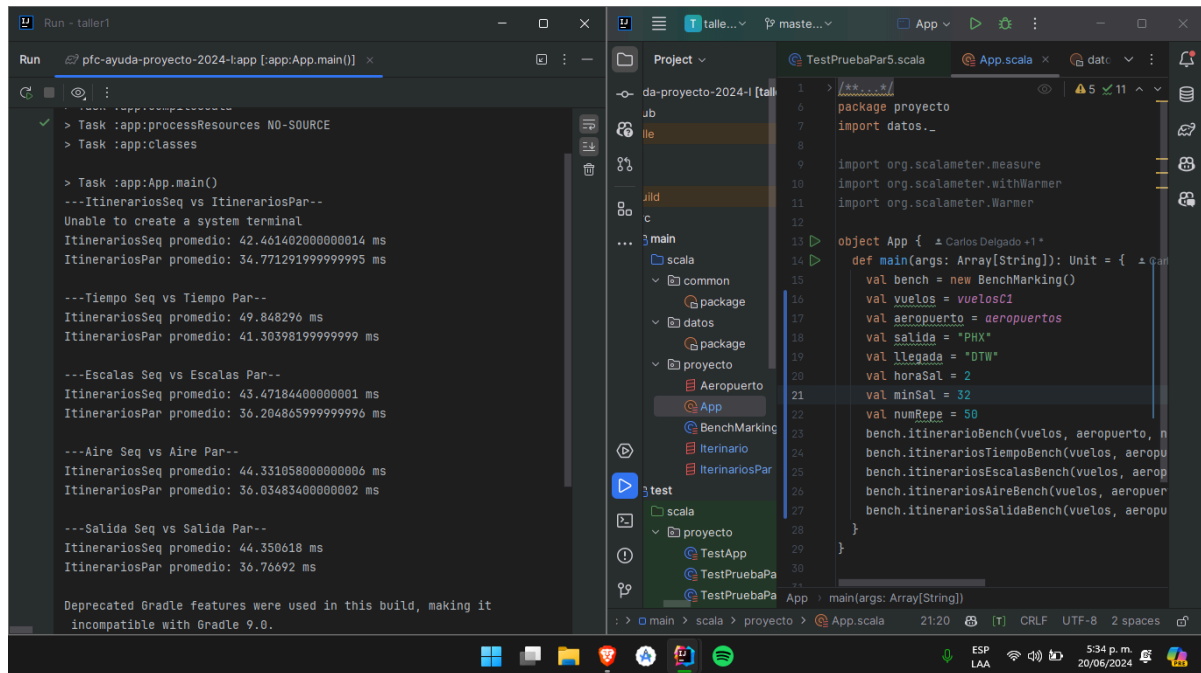
---Salida Seq vs Salida Par--
ItinerariosSeq promedio: 52.103703999999998 ms
ItinerariosPar promedio: 45.243672000000004 ms

Deprecated Gradle features were used in this build, making it incompatible with Gradle 9.0.
You can use '--warning-mode all' to show the individual deprecation warnings and determine if they come from your own scripts or plugins.
```

Scala Code (App.scala):

```
1  /**...*/
2
3  package proyecto
4
5  import ...
6
7
8
9
10
11
12
13  object App { @santiagocarrilloT +1*
14
15  def main(args: Array[String]): Unit = { @santiagocarrilloT +1*
16
17  val bench = new BenchMarking()
18
19  val salida = "MIA"
20  val llegada = "ORD"
21
22  val horaSal = 12
23  val minSal = 30
24  val numRepe = 50
25
26  bench.itinerarioBench(vuelosC1, aeropuertos, numRepe)(salida,
27  bench.itinerariosTiempoBench(vuelosC1, aeropuertos, numRepe)(s
28  bench.itinerariosEscalasBench(vuelosC1, aeropuertos, numRepe)(
29  bench.itinerariosAireBench(vuelosC1, aeropuertos, numRepe)(sal
30  bench.itinerariosSalidaBench(vuelosC1, aeropuertos, numRepe)(s
31
32  }
33
34  }
```

Prueba 2: “PHX”, “DTW” y 50 repeticiones:



The screenshot displays an IDE with a Scala application and its test results. The left pane shows the output of the application, which includes performance metrics for various scenarios. The right pane shows the source code of the application, which is a simple benchmarking tool. The bottom status bar indicates the current file is `App.scala` and the project is `App`.

Run - taller1

Run: pfc-ayuda-proyecto-2024-tapp [app.App.main()] x

Task :app:processResources NO-SOURCE

Task :app:classes

Task :app:App.main()

---ItinerariosSeq vs ItinerariosPar--

Unable to create a system terminal

ItinerariosSeq promedio: 42.461402000000014 ms

ItinerariosPar promedio: 34.771291999999995 ms

---Tiempo Seq vs Tiempo Par--

ItinerariosSeq promedio: 49.848296 ms

ItinerariosPar promedio: 41.303981999999999 ms

---Escalas Seq vs Escalas Par--

ItinerariosSeq promedio: 43.471844000000001 ms

ItinerariosPar promedio: 36.204865999999996 ms

---Aire Seq vs Aire Par--

ItinerariosSeq promedio: 44.331058000000006 ms

ItinerariosPar promedio: 36.834834000000002 ms

---Salida Seq vs Salida Par--

ItinerariosSeq promedio: 44.350618 ms

ItinerariosPar promedio: 36.76692 ms

Deprecated Gradle features were used in this build, making it incompatible with Gradle 9.0.

Project

- da-proyecto-2024-I [taller1]
- sub
- ile
- ild
- c
- main
 - scala
 - common
 - package
 - datos
 - package
 - proyecto
 - Aeropuerto
 - App
 - BenchMarking
 - Itinerario
 - ItinerariosPar
 - test
 - scala
 - proyecto
 - TestApp
 - TestPruebaPa
 - TestPruebaPa

TestPruebaPa5.scala

```
1 //**...*/
2 package proyecto
3 import datos._
4
5 import org.scalameter.measure
6 import org.scalameter.withWarmer
7 import org.scalameter.Warmer
8
9 object App {
10   @ Carlos Delgado +1*
11   def main(args: Array[String]): Unit = {
12     val bench = new Benchmarking()
13     val vuelos = vuelosC1
14     val aeropuerto = aeropuertos
15     val salida = "PHX"
16     val llegada = "DTW"
17     val horaSal = 2
18     val minSal = 32
19     val numRepe = 50
20     bench.itinerarioBench(vuelos, aeropuerto, n
21     bench.itinerariosTiempoBench(vuelos, aeropu
22     bench.itinerariosEscalasBench(vuelos, aerop
23     bench.itinerariosAireBench(vuelos, aeropuer
24     bench.itinerariosSalidaBench(vuelos, aeropu
25   }
26 }
```

App.scala

```
1 //**...*/
2 package proyecto
3 import datos._
4
5 import org.scalameter.measure
6 import org.scalameter.withWarmer
7 import org.scalameter.Warmer
8
9 object App {
10   @ Carlos Delgado +1*
11   def main(args: Array[String]): Unit = {
12     val bench = new Benchmarking()
13     val vuelos = vuelosC1
14     val aeropuerto = aeropuertos
15     val salida = "PHX"
16     val llegada = "DTW"
17     val horaSal = 2
18     val minSal = 32
19     val numRepe = 50
20     bench.itinerarioBench(vuelos, aeropuerto, n
21     bench.itinerariosTiempoBench(vuelos, aeropu
22     bench.itinerariosEscalasBench(vuelos, aerop
23     bench.itinerariosAireBench(vuelos, aeropuer
24     bench.itinerariosSalidaBench(vuelos, aeropu
25   }
26 }
```

main > scala > proyecto > App.scala 21:20 ESP LAA 534 p. m. 20/06/2024