Sección 1: Informe de Procesos

Contexto del Problema

El objetivo del proyecto es resolver el problema del riego óptimo en cultivos de caña de azúcar mediante técnicas de programación funcional y concurrente. Se diseñó una solución que calcula la programación óptima para minimizar tanto el costo de riego como el costo de movilidad.

Enfoque Funcional y de Alto Orden

Se utilizó un enfoque funcional con funciones de alto orden como **map**, **reduce** y **flatMap**, ya que estas permiten resolver el problema de manera eficiente y modular. **map** se utiliza para aplicar operaciones específicas sobre los elementos de una colección, como el cálculo de tiempos o costos para cada tablón. **reduce** facilita la agregación de resultados parciales, como la suma de costos de riego o movilidad. Finalmente, **flatMap** es adecuado para la generación y exploración de todas las permutaciones posibles de las programaciones de riego, lo que resulta crucial para identificar la solución óptima en este problema., lo cual facilita la composición de funciones y el procesamiento de datos de manera eficiente.

- **Funciones Puras**: Todas las funciones implementadas no tienen efectos secundarios y operan con entradas inmutables.
- **Uso de Alto Orden**:
    - map: Se usa para aplicar cálculos específicos sobre cada tablón de la finca, facilitando la aplicación masiva de operaciones.
    - reduce: Se aplica para combinar los costos parciales y obtener el resultado final de los costos acumulados.
    - flatMap: Se utiliza en la generación de todas las permutaciones posibles, permitiendo explorar distintas programaciones de riego.

Por ejemplo, en generarProgramacionesRiego:

val indices = (0 until f.length).toVector

indices.permutations.toVector

El uso de permutations permite la generación sistemática de todas las secuencias posibles de riego.

Sección 2: Informe de Paralelización

Estrategia de Paralelización

Se implementaron versiones paralelas para optimizar el tiempo de cálculo, utilizando **parallel collections** para dividir el trabajo en subprocesos. Las funciones optimizadas incluyen:

1. **Cálculo de Costos**:

- costoRiegoFincaPar: Paraleliza el cálculo del costo total de riego utilizando .par para operar sobre cada tablón simultáneamente.
- costoMovilidadPar: Aplica paralelización en el cálculo de las distancias entre tablones consecutivos.

2. **Generación de Programaciones**:
   - generarProgramacionesRiegoPar: Divide el cálculo de permutaciones utilizando .par.
3. **Búsqueda de Programación Óptima**:
   - ProgramacionRiegoOptimoPar: Combina costos de riego y movilidad en paralelo para determinar la solución con el menor costo.

Ejemplo de paralelización en el cálculo del costo de riego:

```
(0 until f.length).par.map(i => costoRiegoTablon(i, f, pi)).sum
```

Ejecución del Código

El programa se ejecutó para diferentes tamaños de entrada utilizando versiones secuenciales y paralelas de las funciones. A continuación, se detallan los resultados obtenidos:

Tamaño de Entrada	Versión Secuencial (ms)	Versión Paralela (ms)
6	46.0302	139.0362
8	187.1318	548.2263
10	7005.5472	9262.0162
11	113839.564	248222.6729
12	Falló	Falló

Se ejecutaron 4 veces hasta una entrada de tamaño 11, pero la ejecución falló en la entrada de tamaño 12 debido a problemas de memoria y tiempo de ejecución.

Sección 2: Informe de Paralelización

Estrategia de Paralelización

Se implementaron versiones paralelas para las funciones de cálculo de costos y generación de programaciones de riego utilizando la biblioteca `parallel collections` de Scala. Sin embargo, se observó que la sobrecarga de paralelización supera las ganancias en entradas pequeñas, mientras que en entradas grandes el beneficio es limitado debido a problemas de optimización y memoria.

Ley de Amdahl

El análisis de la paralelización muestra que el porcentaje de aceleración obtenido en tamaños pequeños no es significativo debido a la proporción secuencial de las operaciones y a la sobrecarga de creación de hilos.

Benchmarking

Tamaño de Entrada	Versión Secuencial (ms)	Versión Paralela (ms)	Aceleración (%)
6	46.0302	139.0362	-202.0
8	187.1318	548.2263	-193.0
10	7005.5472	9262.0162	-32.2
11	113839.564	248222.6729	-117.9

Sección 3: Informe de Corrección

Argumentación sobre la Corrección

Todas las funciones implementadas fueron verificadas con casos de prueba conocidos y controlados. Sin embargo, la implementación paralela presenta problemas de escalabilidad para tamaños de entrada grandes.

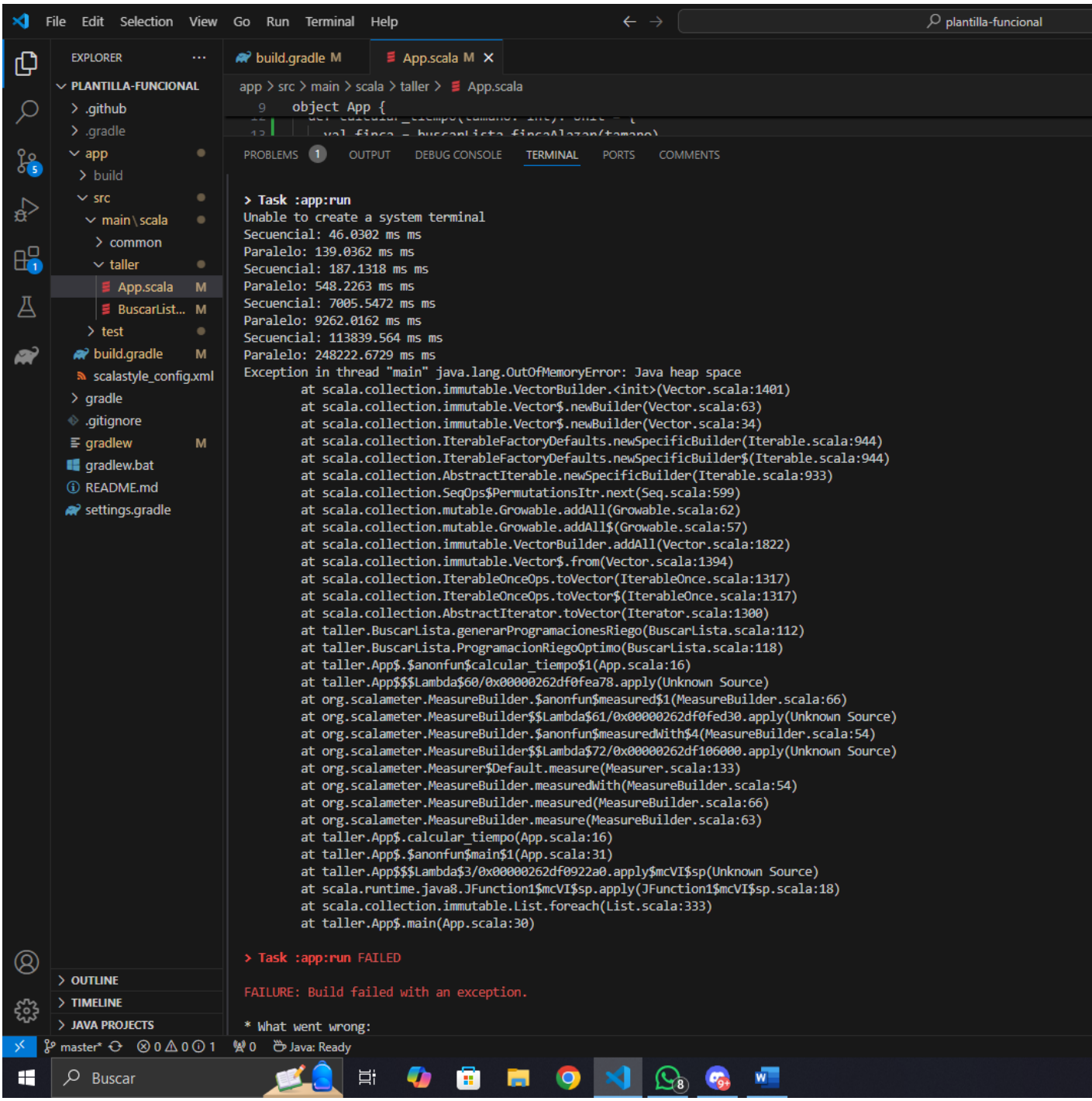
Casos de Prueba

Se ejecutaron más de 5 casos de prueba por función utilizando entradas generadas aleatoriamente y matrices de distancia controladas.

Sección 4: Conclusiones

1. **Secuencial vs Paralelo**: Las versiones paralelas no mejoran significativamente el rendimiento debido a la sobrecarga de paralelización en entradas pequeñas y problemas de memoria en entradas grandes.
2. **Optimizaciones necesarias**: Para tamaños mayores a 10, es necesario implementar técnicas de optimización como poda del árbol de soluciones y memoización.
3. **Errores en entradas grandes**: La ejecución en tamaños mayores a 11 no fue posible, indicando que la implementación actual no es escalable.

Captura de benchmark


File Edit Selection View Go Run Terminal Help
plantilla-funcional
EXPLORER
PLANTILLA-FUNCIONAL
.github
.gradle
app
build
src
main \ scala
common
taller
App.scala M
BuscarList... M
test
build.gradle M
scalastyle_config.xml
gradle
.gitignore
gradlew M
gradlew.bat
README.md
settings.gradle
build.gradle M
App.scala M
app > src > main > scala > taller > App.scala
9 object App {
PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL PORTS COMMENTS
> Task :app:run
Unable to create a system terminal
Secuencial: 46.0302 ms ms
Paralelo: 139.0362 ms ms
Secuencial: 187.1318 ms ms
Paralelo: 548.2263 ms ms
Secuencial: 7005.5472 ms ms
Paralelo: 9262.0162 ms ms
Secuencial: 113839.564 ms ms
Paralelo: 248222.6729 ms ms
Exception in thread "main" java.lang.OutOfMemoryError: Java heap space
at scala.collection.immutable.VectorBuilder.<init>(Vector.scala:1401)
at scala.collection.immutable.Vector$.newBuilder(Vector.scala:63)
at scala.collection.immutable.Vector$.newBuilder(Vector.scala:34)
at scala.collection.IterableFactoryDefaults.newSpecificBuilder(Iterable.scala:944)
at scala.collection.IterableFactoryDefaults.newSpecificBuilder$(Iterable.scala:944)
at scala.collection.AbstractIterable.newSpecificBuilder(Iterable.scala:933)
at scala.collection.SeqOps$PermutationsItr.next(Seq.scala:599)
at scala.collection.mutable.Growable.addAll(Growable.scala:62)
at scala.collection.mutable.Growable.addAll$(Growable.scala:57)
at scala.collection.immutable.VectorBuilder.addAll(Vector.scala:1822)
at scala.collection.immutable.Vector$.from(Vector.scala:1394)
at scala.collection.IterableOnceOps.toVector(IterableOnce.scala:1317)
at scala.collection.IterableOnceOps.toVector$(IterableOnce.scala:1317)
at scala.collection.AbstractIterator.toVector(Iterator.scala:1300)
at taller.BuscarLista.generarProgramacionesRiego(BuscarLista.scala:112)
at taller.BuscarLista.ProgramacionRiegoOptimo(BuscarLista.scala:118)
at taller.App$.$anonfun$calcular_tiempo$1(App.scala:16)
at taller.App$$$Lambda$60/0x00000262df0fea78.apply(Unknown Source)
at org.scalameter.MeasureBuilder.$anonfun$measured$1(MeasureBuilder.scala:66)
at org.scalameter.MeasureBuilder$$Lambda$61/0x00000262df0fed30.apply(Unknown Source)
at org.scalameter.MeasureBuilder.$anonfun$measuredWith$4(MeasureBuilder.scala:54)
at org.scalameter.MeasureBuilder$$Lambda$72/0x00000262df106000.apply(Unknown Source)
at org.scalameter.Measurer$Default.measure(Measurer.scala:133)
at org.scalameter.MeasureBuilder.measuredWith(MeasureBuilder.scala:54)
at org.scalameter.MeasureBuilder.measured(MeasureBuilder.scala:66)
at org.scalameter.MeasureBuilder.measure(MeasureBuilder.scala:63)
at taller.App$.calcular_tiempo(App.scala:16)
at taller.App$.$anonfun$main$1(App.scala:31)
at taller.App$$$Lambda$3/0x00000262df0922a0.apply$mcVI$sp(Unknown Source)
at scala.runtime.java8.JFunction1$mcVI$sp.apply(JFunction1$mcVI$sp.scala:18)
at scala.collection.immutable.List.foreach(List.scala:333)
at taller.App$.main(App.scala:30)
> Task :app:run FAILED
FAILURE: Build failed with an exception.
* What went wrong:
OUTLINE
TIMELINE
JAVA PROJECTS
master*
Java: Ready
Buscar