# INFORME PROYECTO FINAL PROGRAMACIÓN FUNCIONAL Y CONCURRENTE - PROBLEMA DE LA RECONSTRUCCIÓN DE ADN

Víctor Manuel Hernández Ortiz
*2259520*
*Universidad Del Valle*

Jhon Alejandro Martinez Murillo
*2259565*
*Universidad Del Valle*

## I. CORRECCIÓN DE LAS FUNCIONES IMPLEMENTADAS

**Notitas: *Particularmente debe describir su implementacion de trie y por que son correctas las funciones pertenece, adicionar y arbolDeSufijos. No olvide senalar tambien que colecciones paralelas fueron utilizadas.**

### A. *Solución Ingenua*

Para la solución ingenua, se genera una lista de combinaciones posibles para cadenas del tamaño de la que se desea hallar, posteriormente se recorre cada cadena de las combinaciones posibles y se le pregunta al $oraculo$ por esta, continua hasta finalizar y por ultimo se retorna el resultado.

```
def reconstruirCadenaIngenuo(n: Int, oraculo: Oraculo): Seq
    [Char] = {
    val combinaciones = generarCombinaciones(n)
    val cadenaEncontrada = for {
        cadena <- combinaciones
        if oraculo(cadena) == true
    } yield cadena.toSeq

    cadenaEncontrada.head
}
```

### B. *Solución Ingenua Paralela*

Para la solución ingenua paralela tuvimos dos enfoques, paralelización de datos y paralelización de tareas. Inicialmente notamos que la instrucción $for$ (que hace el recorrido analizando con el $oraculo$ cual de las cadenas pertenecientes a la lista de todas las combinaciones posibles es la correcta) podía ser dividida en varias partes. Si, por ejemplo, tenemos un tamaño de cadena $n = 4$ eso nos arrojaría una lista de combinaciones $4^n$, para este ejemplo $4^4 = 256$ sería el tamaño de la lista, entonces podemos dividir esta lista de combinaciones en 4 partes, cada una de tamaño $64$, y con ello usar un $for$ para cada sub-lista de tamaño $64$ de forma paralela, de esta forma cada instrucción $for$ se ejecuta simultáneamente a las otras, lo que en teoría se traduce a menos tiempo para hallar la cadena correcta. Otro añadido es que se cambia la instrucción $for$ por un $filter$ que realiza la misma acción, hallar la cadena mediante el $oraculo$, pero gracias al $filter$ podemos usar paralelización de datos a la secuencia que se filtrara. De esta forma se usa una secuencia de datos paralela, aplicando paralelismo de datos, y cada comparación de las sub-listas de combinaciones se ejecuta de forma paralela.

```
//tarea de recorrido, recibe una porcion de la lista de
    combinaciones
def tareaRecorrido(bloqueCombinaciones: Seq[String]) : Seq[
    Char] = {
    val combinacionesFiltradas = bloqueCombinaciones.par.
        filter(oraculo(_) == true)
    if (combinacionesFiltradas.isEmpty) {
        Seq.empty[Char]
    } else {
        val combinacion = combinacionesFiltradas.head
        combinacion.toSeq
    }
}
// separo en bloques
val (bloque1, bloque2, bloque3, bloque4) =
    separarCombinaciones(combinaciones)

// Ejecuto las tareas sobre cada bloque
val (resultado1, resultado2, resultado3, resultado4) =
    parallel(tareaRecorrido(bloque1), tareaRecorrido(
    bloque2), tareaRecorrido(bloque3), tareaRecorrido(
    bloque4))
```

Finalmente elijo el resultado que no se encuentre vacío, ya que solo una $tareaRecorrido$ arrojará la cadena que se trata de reconstruir.

### C. *Solución Mejorada*

### D. *Solución Mejorada Paralela*

**Describo el funcionamiento que se implementa anteriormente** El añadido para la paralelización, consta de usar el mismo concepto de dividir el $for$ en varias tareas paralelas

### E. *Turbo Solución*

### F. *Turbo Solución Paralela*

### G. *Turbo Mejorada*

### H. *Turbo Mejorada Paralela*

### I. *Turbo Acelerada*

### J. *Turbo Acelerada Paralela*

## II. DESEMPEÑO Y COMPARACIÓN DE LAS SOLUCIONES SECUENCIALES Y PARALELAS

**Notita: Describir como se hicieron las pruebas de desempeño de los algoritmos** Para las pruebas de desempeño, usamos la función $desempenoFunciones$, la cual prueba cada algoritmo 100 veces con cadenas distintas y para distintos

tamaños, al final se promedian los resultados de tiempo arrojando el resultado que se verá a continuación.

### A. *Desempeño de todas las soluciones juntas*

Estas tres tablas representan una en realidad, pero por efectos visuales decidimos anexarla de esta manera

| Tamaño | Ingenua | Ingenua Paralela | Mejorada |
|---|---|---|---|
| 2 | 0.0381420016 | 0.093908998 | 0.03746996 |
| 3 | 0.0247579992 | 0.109804999 | 0.129887006 |
| 4 | 0.0386939998 | 0.486354006 | 0.215725003 |
| 5 | 0.0933949998 | 2.657260997 | 1.437317997 |
| 6 | 0.4390439993 | 21.21956999 | 9.674953 |
| 7 | 5.3342230015 | 177.01692393 | 67.98518801 |
| 8 | 43.635844 | 1463.531014998 | 506.11463989 |
| 9 | 43.635844 | 1463.531014998 | 506.11463989 |
| 10 | 43.635844 | 1463.531014998 | 506.11463989 |

| Tamaño | Mejorada Paralela | Turbo Solución | Turbo Solución Paralela |
|---|---|---|---|
| 2 | 0.0381420016 | 0.093908998 | 0.03746996 |
| 3 | 0.0247579992 | 0.109804999 | 0.129887006 |
| 4 | 0.0386939998 | 0.486354006 | 0.215725003 |
| 5 | 0.0933949998 | 2.657260997 | 1.437317997 |
| 6 | 0.4390439993 | 21.21956999 | 9.674953 |
| 7 | 5.3342230015 | 177.01692393 | 67.98518801 |
| 8 | 43.635844 | 1463.531014998 | 506.11463989 |
| 9 | 43.635844 | 1463.531014998 | 506.11463989 |
| 10 | 43.635844 | 1463.531014998 | 506.11463989 |

| Tamaño | Turbo Mejorada | Turbo Mejorada Paralela | Turbo Acelerada |
|---|---|---|---|
| 2 | 0.0381420016 | 0.093908998 | 0.03746996 |
| 3 | 0.0247579992 | 0.109804999 | 0.129887006 |
| 4 | 0.0386939998 | 0.486354006 | 0.215725003 |
| 5 | 0.0933949998 | 2.657260997 | 1.437317997 |
| 6 | 0.4390439993 | 21.21956999 | 9.674953 |
| 7 | 5.3342230015 | 177.01692393 | 67.98518801 |
| 8 | 43.635844 | 1463.531014998 | 506.11463989 |
| 9 | 43.635844 | 1463.531014998 | 506.11463989 |
| 10 | 43.635844 | 1463.531014998 | 506.11463989 |

| Tamaño | Turbo Acelerada Paralela |
|---|---|
| 2 | 0.0381420016 |
| 3 | 0.0247579992 |
| 4 | 0.0386939998 |
| 5 | 0.0933949998 |
| 6 | 0.4390439993 |
| 7 | 5.3342230015 |
| 8 | 43.635844 |
| 9 | 43.635844 |
| 10 | 43.635844 |

De esta manera se calcularon los desempeños

```
// Primera parte de la funcion
 def desempenoDeFunciones(tamanoCadena: Int): Vector[Double
     ] = {
    println("Tamanio: " + tamanoCadena)

    // repido 100 veces una reconstruccion para una cadena
        distinta y al final promedio los resultados, esto
        se repite para todas las funciones
    val tiemposIngenua = (1 to 100).map(_ => 0.0).toArray
    for (i <- 0 until 100) {
    val time = withWarmer(new Warmer.Default) measure {
        val cadenaAleatoria = crearADN(tamanoCadena)
        val oraculo = oraculoFunc(cadenaAleatoria)
        reconstruirCadenaIngenuo(cadenaAleatoria.length,
            oraculo)
    }
    tiemposIngenua(i) = time.value
    }
    ...
    ...

    // resultado, promedios de todas las funciones
     Vector(tiemposIngenua.sum / 100, tiemposIngenuaPar.sum
         / 100, tiempoMejorado.sum / 100,
        tiempoMejoradoPar.sum / 100, tiempoTurbo.sum /
        100, tiempoTurboPar.sum / 100, tiempoTurboMejorada
        .sum / 100, tiempoTurboMejoradaPar.sum / 100,
        tiempoTurboAcelerada.sum / 100,
        tiempoTurboAceleradaPar.sum / 100)
}
```

### B. *Desempeño de las soluciones secuenciales*

| Tamaño | Ingenua | Mejorada | Turbo | Turbo Mejo-rada | Turbo Aceler-ada |
|---|---|---|---|---|---|
| 2 | 0.0381 | 0.09390 | 0.03746 | 0.09390 | 0.0374 |
| 3 | 0.02475 | 0.10980 | 0.12988 | 0.09390 | 0.0374 |
| 4 | 0.03869 | 0.48635 | 0.2157 | 0.09390 | 0.0374 |
| 5 | 0.0933 | 2.65726 | 1.4373 | 0.09390 | 0.0374 |
| 6 | 0.43904 | 21.2195 | 9.6749 | 0.09390 | 0.03746 |
| 7 | 5.33422 | 177.0169 | 67.9851 | 0.09390 | 0.03746 |
| 8 | 43.635 | 1463.5310 | 506.1146 | 0.09390 | 0.03746 |
| 9 | 43.635 | 1463.5310 | 506.1146 | 0.09390 | 0.03746 |
| 10 | 43.635 | 1463.5310 | 506.1146 | 0.09390 | 0.03746 |

### C. *Desempeño de las soluciones paralelas*

| Tamaño | Ingenua Paralela | Mejorada Paralela | Turbo Paralela | Turbo Mejorada Paralela | Turbo Acelerada Paralela |
|---|---|---|---|---|---|
| 2 | 0.0381 | 0.09390 | 0.03746 | 0.09390 | 0.0374 |
| 3 | 0.02475 | 0.10980 | 0.12988 | 0.09390 | 0.0374 |
| 4 | 0.03869 | 0.48635 | 0.2157 | 0.09390 | 0.0374 |
| 5 | 0.0933 | 2.65726 | 1.4373 | 0.09390 | 0.0374 |
| 6 | 0.43904 | 21.2195 | 9.6749 | 0.09390 | 0.03746 |
| 7 | 5.33422 | 177.0169 | 67.9851 | 0.09390 | 0.03746 |
| 8 | 43.635 | 1463.5310 | 506.1146 | 0.09390 | 0.03746 |
| 9 | 43.635 | 1463.5310 | 506.1146 | 0.09390 | 0.03746 |
| 10 | 43.635 | 1463.5310 | 506.1146 | 0.09390 | 0.03746 |

De esta manera se calcularon los desempeños para las algoritmos secuenciales y paralelos:

```
// codigo del calculo de algoritmos secuenciales y
    paralelos
```

## III. Comparación de las distintas soluciones y sus paralelas

### A. *Comparación de la solución ingenua y la ingenua Paralela*

| Tamaño | Ingenua | Ingenua Paralela | Aceleración |
|---|---|---|---|
| 2 | 0.1941 | 0.3109 | 0.624316500482 |
| 3 | 0.0969 | 0.1253 | 0.773343974461 |
| 4 | 0.1558 | 0.512 | 0.304296874997 |
| 5 | 16.8657 | 0.4238 | 39.7963662104 |
| 6 | 1.0945 | 0.8826 | 1.24008610922 |
| 7 | 9.3597 | 2.6625 | 3.51538028169 |
| 8 | 74.3937 | 23.9784 | 3.1025297767 |
| 9 | 582.3829 | 198.7335 | 2.93047171 |
| 10 | 4811.1526 | 1559.8417 | 3.0843851 |

### B. *Comparación de la solución Mejorada y la Mejorada Paralela*

| Tamaño | Mejorada | Mejorada Paralela | Aceleración |
|---|---|---|---|
| 2 | 0.4017 | 1.3084 | 0.3070162029 |
| 3 | 0.3693 | 1.4949 | 0.2470399357 |
| 4 | 1.4204 | 6.1935 | 0.2293372083 |
| 5 | 3.5153 | 18.4654 | 0.1903722638 |
| 6 | 128.1462 | 81.0312 | 1.5814427035 |
| 7 | 289.9554 | 532.3358 | 0.5446851404 |
| 8 | 2244.0366 | 2965.9211 | 0.756606977 |
| 9 | 18234.863 | 21556.7414 | 0.845900716 |
| 10 | 91069.015 | 110813.2341 | 0.821824358 |

### C. *Comparación de la solución Turbo con la Turbo Paralela*

| Tamaño | Turbo | Turbo Paralela | Aceleración |
|---|---|---|---|
| 2 | 0.7311 | 0.6296 | 1.16121346886 |
| 3 | 0.7454 | 0.3804 | 1.95951629863 |
| 4 | 1.4337 | 0.8821 | 1.62532592676 |
| 5 | 9.8697 | 2.7326 | 3.61183488252 |
| 6 | 27.3466 | 6.5672 | 4.16411865026 |
| 7 | 149.8199 | 38.782 | 3.86312980248 |
| 8 | 1187.6159 | 339.3482 | 3.4996970663 |
| 9 | 8264.3622 | 2360.687 | 3.5008292924 |
| 10 | 54366.4652 | 18073.1638 | 3.00813215 |

### D. *Comparación de la solución Turbo Mejorada con la Turbo Mejorada Paralela*

| Tamaño | Turbo Mejorada | Turbo Mejorada Paralela | Aceleración |
|---|---|---|---|
| 2 | 0.7311 | 0.6296 | 1.16121346886 |
| 3 | 0.7454 | 0.3804 | 1.95951629863 |
| 4 | 1.4337 | 0.8821 | 1.62532592676 |
| 5 | 9.8697 | 2.7326 | 3.61183488252 |
| 6 | 27.3466 | 6.5672 | 4.16411865026 |
| 7 | 149.8199 | 38.782 | 3.86312980248 |
| 8 | 1187.6159 | 339.3482 | 3.4996970663 |
| 9 | 8264.3622 | 2360.687 | 3.5008292924 |
| 10 | 54366.4652 | 18073.1638 | 3.00813215 |

### E. *Comparación de la solución Turbo Acelerada con la Turbo Acelerada Paralela*

| Tamaño | Turbo Acelerada | Turbo Acelerada Paralela | Aceleración |
|---|---|---|---|
| 2 | 0.7311 | 0.6296 | 1.16121346886 |
| 3 | 0.7454 | 0.3804 | 1.95951629863 |
| 4 | 1.4337 | 0.8821 | 1.62532592676 |
| 5 | 9.8697 | 2.7326 | 3.61183488252 |
| 6 | 27.3466 | 6.5672 | 4.16411865026 |
| 7 | 149.8199 | 38.782 | 3.86312980248 |
| 8 | 1187.6159 | 339.3482 | 3.4996970663 |
| 9 | 8264.3622 | 2360.687 | 3.5008292924 |
| 10 | 54366.4652 | 18073.1638 | 3.00813215 |

De esta manera se calcularon las comparaciones

```
// codigo
```

## IV. Análisis comparativo de las diferentes soluciones y conclusiones