INFORME

Descripción del Problema:

El objetivo es implementar y optimizar el algoritmo de Strassen para la multiplicación de matrices cuadradas. Este algoritmo divide las matrices en submatrices más pequeñas y realiza multiplicaciones de manera eficiente usando menos operaciones que el método estándar.El proyecto incluye dos implementaciones:

Strassen: Implementación secuencial.

StrassenPar: Implementación paralelizada usando Parallel.

Ambas implementaciones permiten calcular el producto de dos matrices cuadradas de tamaño **2^n × 2^n**, donde n ≥ 1.

Introducción:

La multiplicación de matrices es una operación fundamental en múltiples campos, como la inteligencia artificial, el análisis de datos, la computación científica y la informática gráfica. Sin embargo, su elevado costo computacional, especialmente para matrices grandes, ha motivado el desarrollo de algoritmos optimizados que reduzcan el tiempo de ejecución manteniendo la precisión de los resultados. Entre estos algoritmos, el método de Strassen se destaca por ser más eficiente que la multiplicación matricial convencional.

En este proyecto, se implementan dos versiones del algoritmo de Strassen: una secuencial (**Strassen**) y otra paralelizada (**StrassenPar**). El objetivo principal es evaluar su funcionamiento, eficiencia y escalabilidad en distintos escenarios. Además, se incluyen pruebas exhaustivas para validar la corrección de las implementaciones y garantizar su adaptabilidad a diferentes casos de uso.

La estructura del informe abarca una descripción detallada del proceso de desarrollo, pruebas realizadas y un análisis teórico que respalda la corrección de las soluciones. Finalmente, se incluyen reflexiones sobre los aprendizajes obtenidos, los desafíos superados y las posibles aplicaciones prácticas de las implementaciones.

Con este trabajo, se busca no solo demostrar la eficacia del algoritmo de Strassen en la multiplicación de matrices, sino también resaltar la importancia de las técnicas de paralelización en la resolución de problemas computacionales complejos.

1.1 Strassen

Descripción de la Función:

La función Strassen.multiply realiza la multiplicación utilizando 7 multiplicaciones y 10 sumas/restas en lugar de las 8 multiplicaciones del método estándar, lo que mejora su complejidad.

Casos de Prueba:

Multiplicación básica de matrices 2x2:

```
val A = Vector(Vector(2, 4), Vector(6, 8))

val B = Vector(Vector(1, 3), Vector(5, 7))

val resultado = Strassen.multiply(A, B)
```

Resultado esperado:

```
Vector(Vector(22, 34), Vector(46, 74))
```

Matrices con valores negativos:

```
val A = Vector(Vector(-1, -2), Vector(-3, -4))

val B = Vector(Vector(2, 0), Vector(0, 2))

val resultado = Strassen.multiply(A, B)
```

Resultado esperado:

```
Vector(Vector(-2, -4), Vector(-6, -8))
```

Matrices con ceros y valores mixtos:

```
val A = Vector(Vector(0, 1), Vector(2, 3))

val B = Vector(Vector(4, 5), Vector(0, 0))

val resultado = Strassen.multiply(A, B)
```

Resultado esperado:

```
Vector(Vector(0, 0), Vector(8, 10))
```

Multiplicación de matrices cuadradas de tamaño 3x3:

```
val A = Vector(Vector(1, 2, 3), Vector(4, 5, 6), Vector(7, 8, 9))

val B = Vector(Vector(9, 8, 7), Vector(6, 5, 4), Vector(3, 2, 1))
```

val resultado = Strassen.multiply(A, B)

Resultado esperado:

Vector(Vector(30, 24, 18), Vector(84, 69, 54), Vector(138, 114, 90))

Multiplicación de una matriz consigo misma:

val A = Vector(Vector(1, 2, 3), Vector(4, 5, 6), Vector(7, 8, 9))

val resultado = Strassen.multiply(A, A)

Resultado esperado:

Vector(Vector(30, 36, 42), Vector(66, 81, 96), Vector(102, 126, 150))

1.2 StrassenPar

Descripción de la Función:

La función StrassenPar.multiply es una extensión paralelizada del algoritmo de Strassen, utilizando la librería Parallel para dividir las tareas de multiplicación entre subprocesos.

Casos de Prueba:

Multiplicación básica de matrices 2x2:

val A = Vector(Vector(1, 2), Vector(3, 4))

val B = Vector(Vector(5, 6), Vector(7, 8))

val resultado = StrassenPar.multiply(A, B)

Resultado esperado:

Vector(Vector(19, 22), Vector(43, 50))

Matrices con valores negativos:

val A = Vector(Vector(-1, 0), Vector(0, -1))

val B = Vector(Vector(1, 1), Vector(1, 1))

val resultado = StrassenPar.multiply(A, B)

Resultado esperado:

Vector(Vector(-1, -1), Vector(-1, -1))

Matrices con ceros en algunas filas:

```
val A = Vector(Vector(0, 0), Vector(1, 1))
val B = Vector(Vector(1, 1), Vector(1, 1))
resultado = StrassenPar.multiply(A, B)
```

Resultado esperado:

Vector(Vector(0, 0), Vector(2, 2))

Matrices de tamaño 4x4 con valores aleatorios:

```
val A = Vector(
Vector(1, 2, 3, 4),
Vector(5, 6, 7, 8),
Vector(9, 10, 11, 12),
Vector(13, 14, 15, 16)
)
val B = Vector(
Vector(16, 15, 14, 13),
Vector(12, 11, 10, 9),
Vector(8, 7, 6, 5),
Vector(4, 3, 2, 1)
)
resultado = StrassenPar.multiply(A, B)
```

Resultado esperado:

```
Vector(
Vector(80, 70, 60, 50),
Vector(240, 214, 188, 162),
```

Vector(400, 358, 316, 274),

Vector(560, 502, 444, 386)

)

Matrices cuadradas con valores pequeños y grandes:

val A = Vector(Vector(1000, 2000), Vector(3000, 4000))

val B = Vector(Vector(5000, 6000), Vector(7000, 8000))

resultado = StrassenPar.multiply(A, B)

Resultado esperado:

Vector(Vector(19000000, 22000000), Vector(43000000, 50000000))

Conclusiones:

Las implementaciones de Strassen y StrassenPar cumplen con los objetivos planteados al abordar de manera eficiente la multiplicación de matrices cuadradas. Ambos algoritmos demuestran ser funcionales y correctos, y la incorporación de pruebas exhaustivas asegura su fiabilidad en escenarios variados.

La versión secuencial (**Strassen**) es adecuada para casos en los que la paralelización no es viable o cuando se trabaja con matrices pequeñas. Por otro lado, la versión paralelizada (StrassenPar) destaca por su capacidad para manejar matrices grandes, aprovechando los recursos de hardware modernos para mejorar el rendimiento, especialmente en sistemas multi-nucleo.

El proceso de desarrollo permitió identificar y resolver desafíos clave, como garantizar que las matrices cumplieran con el tamaño requerido **(2^n × 2^n)** y ajustar el algoritmo a las peculiaridades de Scala. Las pruebas realizadas abarcaron una variedad de casos, desde matrices pequeñas y simples hasta matrices grandes con valores negativos y ceros. Esto demostró la robustez de las implementaciones y su capacidad para adaptarse a diferentes condiciones.

Finalmente, este trabajo subraya la relevancia de algoritmos como el de Strassen en el contexto de aplicaciones prácticas, desde el análisis de datos hasta la simulación científica, donde la multiplicación de matrices es un componente fundamental.