INFORME TALLER N° 4 FUNCIONAL

Alejandro Marin Hoyos 2259353-3743

Yessica Fernanda Villa Nuñez 2266301-3743

Manuel Antonio Vidales Duran 2155481-3743

UNIVERSIDAD DEL VALLE SEDE TULUÁ

PROGRAMA ACADÉMICO DE INGENIERÍA DE SISTEMAS

2023

1.Informe corrección de las funciones implementadas.

1.1. Multiplicación estándar de matrices.

- **ProdPunto:** Esta función realiza el producto punto entre dos vectores de enteros. Combina los elementos correspondientes de los dos vectores, multiplica cada par de elementos y luego suma todos los productos resultantes. El tipo de retorno de la función es un entero, que representa el resultado del producto punto. Utiliza la función zip para combinar los elementos de los dos vectores, luego usa map para aplicar la multiplicación a cada par de elementos y finalmente suma los resultados. La implementación es concisa y aprovecha las funciones de orden superior de Scala de manera efectiva.

- **transpuesta:** Esta función crea una nueva matriz que es la transpuesta de la matriz de entrada m. Utiliza Vector.tabulate para generar la nueva matriz y utiliza una función anónima para intercambiar las coordenadas (i, j) para obtener los elementos de la matriz transpuesta. La función es concisa y utiliza funciones de orden superior proporcionadas por Scala para generar la matriz transpuesta de manera eficiente.

1.1.1. Versión estándar secuencial.

- **mulMatriz:**La función mulMatriz recibe dos matrices como entrada. Primero, calcula la transpuesta de la segunda matriz y luego realiza la multiplicación de matrices utilizando el producto escalar entre las filas de la primera matriz y las columnas (transpuestas) de la segunda matriz. El resultado es una nueva matriz que representa el producto de las dos matrices originales. Esta función ha sido creada específicamente para operar con matrices representadas como matrices bidimensionales.

1.1.2. Versión estándar paralela.

- **mulMatrizParalelo:** Esta función divide las matrices de entrada en dos partes, realiza la multiplicación de matrices para cada parte en paralelo utilizando tareas (task), y luego concatena los resultados para obtener la matriz final resultante de la multiplicación de matrices. La implementación utiliza la programación paralela para mejorar el rendimiento de la multiplicación de matrices, dividiendo la tarea en partes que pueden ejecutarse simultáneamente.

1.2. Multiplicación recursiva de matrices.

1.2.1. Extrayendo submatrices.

- **subMatriz:** Define la función subMatriz de entrada m y toma tres enteros como argumentos de entrada para devolver una matriz como resultado. Emplea el método **tabulate** de la clase **Vector** es utilizada para crear una nueva matriz 1*1, comenzando desde la posición (i, j). Ayuda cuando se necesita obtener una parte concreta de una matriz grande para realizar operaciones o interpretaciones de una parte específica.

1.2.2. Sumando matrices.

- **sumMatriz:** Se crea una función sumMatriz, utiliza como datos de entrada a m1 y m2 y devuelve una matriz que representa la suma de las 2 matrices.
  La suma de matrices nos permite combinar o agregar elementos correspondientes de dos matrices dadas.

1.2.3. Multiplicando matrices recursivamente, versión secuencial.

- **multMatrizRec:** La función mulMatrizRec opera inicialmente con dos matrices cuadradas, m1 y m2, como entrada, y produce el resultado de su multiplicación. En el caso base, cuando ambas matrices son de tamaño 1x1, se efectúa la multiplicación de sus elementos correspondientes, generando una matriz de igual tamaño con el resultado.

  El enfoque de "Divide y Vencerás" se emplea para matrices de mayor tamaño, dividiéndolas en cuatro submatrices más pequeñas (a11, a12, a21, a22, b11, b12, b21, b22). Posteriormente, se calculan submatrices intermedias mediante llamadas recursivas a la función, y estas submatrices se combinan de manera estratégica para obtener la matriz resultante. Durante la combinación, los resultados intermedios se organizan en las posiciones correctas de la matriz final.

  El algoritmo utiliza la técnica "Divide y Vencerás" con el objetivo de mejorar la eficiencia en el cálculo de la multiplicación de matrices, lo que se traduce en una reducción de la complejidad computacional, especialmente beneficiosa para matrices de dimensiones considerables.

1.2.4. Multiplicando matrices recursivamente, versión paralela.

- **multMatrizRecPar:** Este código introduce una variante paralela del algoritmo de multiplicación de matrices basado en la estrategia "**Divide y Vencerás**". La función **multMatrizRecPar** toma como entrada dos matrices cuadradas, m1 y m2, y genera el resultado de su multiplicación. Utiliza la función **parallel**, presumiblemente perteneciente a algún marco de concurrencia o paralelismo, para ejecutar las llamadas recursivas simultáneamente, aprovechando así la capacidad de procesamiento paralelo. A pesar de la paralelización, la lógica subyacente del algoritmo sigue siendo coherente con la versión no paralela.

  La función **multMatrizRecPar** representa una adaptación paralela del algoritmo de multiplicación de matrices mediante la técnica de "**Divide y Vencerás**". Esta versión divide las matrices en submatrices más pequeñas y realiza los cálculos de multiplicación de manera concurrente, buscando mejorar la eficiencia computacional, especialmente cuando se trata con matrices de considerable tamaño.

1.3. Multiplicación de matrices usando el algoritmo de Strassen.

1.3.1. Restando matrices.

- **restaMatriz:**El código define una función llamada **restaMatriz**. Esta función toma dos matrices, **m1** y **m2**, como argumentos y devuelve una nueva matriz que resulta de restar cada elemento de **m2** al elemento correspondiente de **m1**. La longitud de las filas o columnas de las matrices se almacena en la variable **n**. Se utiliza el método tabulate de la clase **Vector** para crear la nueva matriz, especificando la lógica de la resta en la función proporcionada. Finalmente, la función **restaMatriz** realiza la operación de resta entre dos matrices, proporcionando una matriz resultante con las diferencias correspondientes.

1.3.2. Algoritmo de Strassen, versión secuencial.

- **multStrassen:** En la práctica, se ha implementado el algoritmo de multiplicación de matrices de **multStrassen**, que divide la multiplicación de matrices en operaciones más pequeñas y minimiza el número de cálculos de multiplicación básicos en comparación con el método tradicional.
  Detalles del proceso: Si las dimensiones de las matrices m1 y m2 son 1x1, se realiza el producto de sus elementos correspondientes, dando como resultado una matriz de 1x1.

Divide y vencerás: Tanto la matriz m1 como la matriz m2 se dividen en cuatro submatrices más pequeñas (a11, a12, a21, a22, b11, b12, b21, b22).
Submatrices intermedias (p1-p7): Los siete productos intermedios (p1 a p7) se calculan mediante llamadas recursivas a la función multStrassen, que combina operaciones de suma y resta entre submatrices.
Las submatrices resultantes (c11-c22): Las cuatro submatrices resultantes (c11, c12, c21, c22) se calculan a partir de las interpolaciones resultantes (p1 a p7) utilizando operaciones de suma y resta.

Las submatrices resultantes se combinan en una matriz final que representa la multiplicación de la matriz original. Esta estrategia de divide y vencerás, junto con el cálculo de productos intermedios y submatrices resultantes, permite minimizar el número de multiplicaciones básicas en la multiplicación de matrices.

1.3.3. Algoritmo de Strassen, versión paralela.

- **multStrassenPar:**Primero se obtienen las dimensiones de tablas:
  val n = m1.head.count(_ =>; true) cuenta dimensiones de tablas m1 y m2.

Caso base (matrices 1x1): Si la medida es 1, se insertan directamente los elementos correspondientes y se devuelve una matriz de 1x1 con el resultado.

Divide y conquistarás: Divide los arreglos m1 y m2 en cuatro subarreglos más pequeños (a11, a12, a21, a22, b11, b12, b21, b22). Cada subconjunto se procesa en paralelo mediante tareas, lo que permite que las tareas se ejecuten de forma asincrónica y en paralelo.

Cálculo de productos intermedios (p1-p7): Calcula siete productos intermedios (p1 - p7) utilizando llamadas recursivas a la función multStrassenPar y operaciones de suma y resta entre submatrices. Todos estos cálculos se realizan en paralelo.

Calcular las submatrices resultantes (c11-c22): Calcula las cuatro submatrices resultantes (c11, c12, c21, c22) utilizando las operaciones de suma y resta de las interpolaciones calculadas.

Ensamblando la matriz resultante: Combine las submatrices resultantes para formar una matriz final que represente la multiplicación de la matriz original.

2. Informe de desempeño de las funciones secuenciales y paralelas.

- **Tabla comparativa de multiplicación de matriz versión secuencial y paralela:**

Tamaño de la matriz	mulMatriz (Tiempo)	mulMatrizParalelo (Tiempo)	Aceleración
2x2	0.2363	0.4642	0.509047824213701
4x4	0.1749	0.348	0.5025862068965518
8x8	0.2622	0.5317	0.4931352266315592
16x16	1.1264	0.7381	1.5260804769001493
32x32	1.8058	2.8584	0.6317520291071929
64x64	13.4328	3.3178	4.048706974501176
128x128	50.8725	56.522	0.9000477690102969
256x256	931.8295	428.4266	2.1750038396308726
512x512	5695.1551	2709.8084	2.101681838465037
1024x1024	57651.4013	24392.6326	2.3634759824980924

Ejemplos de prueba: Se define una secuencia (Seq) llamada dimensiones que contiene diferentes tamaños de matrices. Estos tamaños van desde 2 hasta 1024.

Se imprime un encabezado para la tabla de comparación que se generará en el siguiente bucle.

Bucle de pruebas

Para cada dimensión de la secuencia, se crean dos matrices aleatorias **(matriz1 y matriz2)** utilizando la función **RandomMatrix(dim, 10)**. Estas matrices se generan con dimensiones débiles y valores aleatorios entre 0-10.
Se utiliza la función compararAlgoritmos para comparar los tiempos de ejecución de los algoritmos mulMatriz y mulMatrizParalelo con las matrices generadas. El resultado se desestructura en tres variables: tiempoMulMatriz, tiempoMulMatrizParalelo, y aceleración.
Finalmente, se imprime una fila de la tabla con las dimensiones de la matriz y los tiempos de ejecución para la multiplicación y aceleración de matrices en serie y en paralelo.

- **Tabla comparativa de multiplicación de matriz recursiva versión secuencial y paralela:**

Tamaño de la matriz	multMatrizRec (Tiempo)	multMatrizRecPa (Tiempo)	Aceleración
2x2	0.5425	0.4045	1.3411619283065512
4x4	1.2017	0.6204	1.936976144422953
8x8	0.7796	1.0324	0.755133669120496
16x16	7.1856	5.0533	1.421961886292126
32x32	31.2245	5.6694	5.507549299749532
64x64	94.4191	68.3501	1.3814039774630908
128x128	1016.9697	1007.5422	1.0093569281763086
256x256	6476.8956	7825.1859	0.8276986237477117
512x512	37703.6096	22867.3506	1.6487965859936569
1024x1024	306492.1518	181600.2841	1.687729473106039

Ejemplos de prueba: Un bucle for repite i en 1 a 10 (1 a 10). En cada iteración se generan dos matrices aleatorias de dimensiones fijas (m1 y m2). Las dimensiones se calculan como 2 elevado a la potencia i.

Llamada a función y salida de retorno: La función Comparar Algoritmos se llama para comparar los tiempos de ejecución de las funciones multMatrizRec y multMatrizRecPar con las matrices generadas. El resultado de esta comparación se imprime con el valor 2 elevado a la potencia i. La declaración de devolución indica que estos resultados se recopilaron en una colección. La función println se utiliza para imprimir la salida a la consola durante cada iteración. En resumen, este código ejecuta las iteraciones 1 a 10, generar matrices aleatorias en cada iteración y compara los tiempos de ejecución de las funciones multMatrizRec y multMatrizRecPar con esas matrices.

Los resultados de estas comparaciones y las dimensiones de las matrices (hasta la potencia de 2 i) se imprimen en la consola.

Tabla comparativa de multiplicación de matriz del método Strassen versión secuencial y paralela:

Tamaño de la matriz	multStrassen (Tiempo)	multStrassenPar (Tiempo)	Aceleración
2x2	0.3175	0.2223	1.4282501124606388
4x4	0.2937	0.2997	0.97997997997998
8x8	0.6274	0.7042	0.8909400738426582
16x16	5.3317	4.9783	1.0709880883032359
32x32	25.6238	10.0176	2.5578781344833095
64x64	86.0847	80.1849	1.0735774441322494
128x128	641.781	595.7334	1.0772956493626176
256x256	7945.654	4179.2379	1.9012207943462611
512x512	32393.0507	31646.8297	1.0235796446934462
1024x1024	222408.638	145914.1267	1.5242433548416667

Ejemplos de prueba: Especifica una secuencia (Seq) llamada dimensión que contiene matrices de diferentes tamaños. Estos tamaños van del 2 al 1024.

Se imprime un encabezado para la tabla de comparación que incluye las dimensiones de la matriz, los tiempos de ejecución para multStrassen y multStrassenPar, y la aceleración.

Para cada dimensión de la secuencia, se crean dos matrices aleatorias (matriz1 y matriz2) utilizando la función RandomMatrix(dim, 100). Estos arreglos se generan con dimensiones débiles y valores aleatorios entre 0 y 100.

La función CompleteAlgoritms se utiliza para comparar los tiempos de ejecución de los algoritmos multStrassen y multStrassenPari con las matrices generadas. El resultado se divide en tres variables: timeMultStrassen, timeMultStrassenPar y aceleración.

Las dimensiones de la matriz y los tiempos de ejecución para la multiplicación y aceleración de matrices en serie y en paralelo están impresos en una fila de la tabla. Sandquot;| entrada $tenue | $timeMultStrassen | $timeMultStrassenPar | $aceleracion |; Esto se utiliza para un formato más legible de la salida.

3. Análisis comparativo de las diferentes funciones.

- **comparativa de multiplicación de matriz versión secuencial y paralela:**

1. **¿Cúal de las implementaciones es más rápida?**
   Para matrices pequeñas (2x2, 4x4, 8x8), la versión paralela **mulMatrizParalelo** es más lenta que la versión secuencial **mulMatriz**. Para el resto de las matrices la versión **mulMatrizParalelo** resulta ser mas rapida.

2. **¿De que depende que la aceleración sea mejor?**
   La aceleración depende de la naturaleza del algoritmo, en general, los algoritmos paralelos son más efectivos cuando se aplican a conjuntos de datos lo suficientemente grandes y cuando la sobrecarga de la paralelización es menor que la ganancia obtenida por la ejecución en paralelo.

3. **¿Puede caracterizar los casos en que es mejor usar la versión secuencial/paralela de cada algoritmo de multiplicación de matrices?**

   Versión Secuencial:
   Mejor para matrices pequeñas o medianas, donde la sobrecarga de la paralelización puede ser significativa.
   Puede ser preferible en situaciones donde la memoria compartida es limitada y la gestión de concurrencia agrega complejidad sin proporcionar beneficios significativos.

   Versión Paralela:
   Mejor para matrices grandes donde la paralelización puede aprovechar eficazmente los recursos.
   Puede ser preferible en sistemas con múltiples núcleos o clústeres, donde la paralelización puede distribuir la carga de trabajo y mejorar el rendimiento.