Universidad del Valle

Taller No.4

William Alexander Franco Otero - 2259715
Jhojan Stiven Castaño Jejen - 2259476
David Santiago Velasco Triana - 2259479

1. Corrección De Funciones Implementadas

1.1.1 Función Multiplicar Matriz (MultMatriz)

La función MultMatriz de complejidad O(n^3) debido se encarga de implementar la multiplicación de matrices estándar. Donde la función comienza de agarrar la segunda matriz que es 'm2' para transponerla con la función 'transpuesta', es fundamental la transpuesta para que se alineen correctamente las filas y las columnas de ambas matrices para la multiplicación, luego se inicializa la variable 'l' con la longitud de la primera matriz 'm1', dando entender que ambas matrices son cuadradas; para luego utilizar el 'Vector.tabulate' para así generar la matriz resultante 'C', que se calcula por medio de la función 'prodPunto'.

Donde cada posición (i, j) en la matriz resultante 'C', se calcula el producto punto de cada fila i en la matriz m1 y la columna j de la matriz transpuesta m2t.Luego, la función 'prodPunto' toma dos vectores como entrada y realiza el producto punto entre ellos. Para así utilizar el 'zip' y combinar los elementos de ambos vectores, multiplica los elementos y luego se suma los resultados. Para finalizar la matriz resultante se construye con los valores calculados y devuelve como resultado de la función.

1.1.2 Función multiplicar Matriz Paralela (multMatrizParalelo)

La función multMatrizParalelo demuestra la implementación de la multiplicación paralela de matrices. La función comienza con transponer la segunda matriz 'm2' utilizando la función de 'transpuesta', se utiliza como ya se mencionó antes para alinear correctamente las filas y las columnas de ambas matrices. Luego, la matriz 'm1' se divide en dos partes que son 'm1a' y 'm1b' dividiendo las matrices a la mitad con el objetivo de distribuir el trabajo de carga de trabajo entre las tareas paralelas, provocando así que cada tarea calcula una cierta porción de la matriz resultante.

Además, se crean dos tareas paralelas (top y bot) utilizando la función task. Cada tarea se encarga de realizar la multiplicación de matrices de las submatrices de ‘m1’ con la matriz transpuesta ‘m2t’. El cálculo realizado se elabora mediante el map y la función ‘prodPunto’; para finalizar se utiliza un join( ) para la espera de que ambas tareas (top y bot) terminen. Después de terminar la tarea, los resultados se concatenan utilizando el operador ‘++’ para construir la matriz resultante ‘C’ completa

1.2 Función multiplicación recursiva de matrices (multMatrozRec)

Para la elaboración de la función de la multiplicación recursiva se necesita saber que se utilizara otra función llamada ‘MatrixAddition’ donde está función toma dos matrices de igual tamaño como entrada ‘mtx1’ y ‘mtx2’, donde ‘n’ es la longitud de una dimensión de matrices, luego se utiliza el vector.tabulate para crear una nueva matriz de tamaño ***n x n*** utilizando la función de tabulación. Para cada posición que hay (i, j) en la nueva matriz se realiza la operación de suma ‘mtx1(i) (j)’ + ‘mtx2(i) (j)’, dando como resultado el valor que corresponde a la matriz resultante ‘C’ .

Ahora en la función ‘multMatrozRec’ se implementa un caso base donde n es igual a una matriz 1x1, provocando así que la función retorna la multiplicación directa en esas mismas posiciones, en caso de que no sea una matriz 1x1 se dividen en cuatro submatrices más pequeñas que tienen etiquetas (a, b, c, d) para ‘mtx1’ y otras cuatro submatrices (e, f, g, h) para ‘mtx2’. Luego se realizan llamadas recursivas a ‘multMatrozRec’ para calcular las submatrices resultantes (leftup, rightup, leftlow, rightlow) y se suman las submatrices calculadas para obtener cuatro submatrices resultantes; por último, se concatenan las submatrices resultantes para luego formar la matriz final de salida.

1.2.1 Función sustracción de matrices (subMatriz)

La función ‘submatriz’ tiene como entrada una matriz m, una fila de inicio de la matriz original ‘i’ y una columna de inicio en la matriz original ‘j’, y el tamaño de la submatriz ‘l’. El ‘Vector.tabulate’ se encarga de crear un vector bidimensional de tamaño ***l x l***, que tiene como propósito construir el resultado de la submatriz. La función lambda se ejecuta para cada posición de (row, col) en la nueva submatriz y luego se extrae el elemento de la matriz original y se sitúa en la posición correcta de la submatriz, para así construir la submatriz utilizados de los elementos que fueron extraídos de la matriz original con el ‘Vector.tabulate’.

1.2.2 Función sumando matrices (matrixAddition)

La función matrixAddition toma dos matrices como entrada en este caso “mtx1” y “mtx2” estas matrices deben ser del mismo tamaño, se obtiene la longitud n de la primera matriz “mtx1” asumiendo que las matrices son cuadradas, luego se utiliza la función vector.tabulate para generar una nueva matriz de tamaño ***n x n***, para cada elemento de la matriz se realiza la suma de estos elementos por medio de la función de alto orden ***(i, j) => mtx1(i)(j) + mtx2(i)(j)***, en cuanto complejidad computacional esta función tiene una complejidad de O(n^2), donde n es el longitud de las matrices de entrada

1.2.3 Función multiplicación de matrices recursivas, versión secuencial (multMatrizRecSec)

La función multMatrizRecSec se obtiene una longitud de las matrices ‘m1’ o ‘m2’ donde se almacenas el valor en la variable ‘n’. En caso de que la n es igual a 1, se realizará la multiplicación directa y devuelve una matriz resultante.

Las matrices se dividen en cuatro submatrices para cada matriz, donde m1 tiene 4 submatrices (a, b, c, d) y m2 maneja otras cuatro submatrices (e, f, g, h), se realizan llamadas recursivas a mulmatrizrecsec para calcular las submatrices resultantes (c11,c12,c21,c22), luego se suman las submatrices utilizando la función matrixAddition para combinarlas y construir la matriz final resultante con ‘top’ y ‘bottom’

1.2.4. Multiplicando matrices recursivamente, versión paralela

La función multMatrizRecPar se implementa la versión paralela con respecto a multMatrizRec en la cual se uso la técnica de “Divide y Vencerás”,

1. El algoritmo funciona con la entrada de ***m1*** **y** ***m2*** en cuatro submatrices cada una las cuales se definen en las líneas ***a11, a12, a21, a22, b11, b12, b21, b22***
2. Se realiza la multiplicación de las submatrices y se suma en paralelo, esto se realiza con la función **parallel** la cual nos permite realizar estas operaciones de forma paralela
3. Finalmente se combinan las submatrices resultantes en una matriz final con la ayuda de la parte del Código donde usamos Vector.tabulate

Esta función tiene una complejidad funcional de O(n^log2(7)) gracias al uso de la técnica “Divide y Vencerás” la cual muestra una mayor eficiencia frente a el algoritmo de multiplicación de matrices estándar la cual es de complejidad O(n^3)

1.3.1 Restando matrices

La función matrixsubtraction toma dos matrices como entrada en este caso “mtx1” y “mtx2” estas matrices deben ser del mismo tamaño, se obtiene la longitud n de la primera matriz “mtx1” asumiendo que las matrices son cuadradas, luego se utiliza la función vector.tabulate para generar una nueva matriz de tamaño ***n x n***, para cada elemento de la matriz se realiza la suma de estos elementos por medio de la función de alto orden ***(i, j) => mtx1(i)(j) - mtx2(i)(j)***, en cuanto complejidad computacional esta función tiene una complejidad de O(n^2), donde n es el longitud de las matrices de entrada

1.3.2 Función de multiplicación de matrices usando el algoritmo de Strassen (multStrassen)

En esta función se implementa la multiplicación de matrices con el algoritmo de Strassen donde se toman dos parámetros ‘m1’ y ‘m2’, donde representan matrices cuadradas que poseen la misma dimensión. En el primer caso se toma de que n es igual a 1, dejando así que la multiplicación de matrices se realice de forma directa y devuelva el resultado. Si las matrices no sean de tamaño 1, entonces se divide cada matriz en 4 submatrices más pequeñas, donde m1 son las submatrices a11, a12, a21, a22 y la matriz m2 con las submatrices b11, b12, b21, b22. Se realizan siete llamadas recursivas que son p1 a p7 utilizando las matrices que se calcularon anteriormente, luego se combinan los resultados de las llamadas recursivas para obtener las submatrices resultantes (c11, c12, 21, c22) y se construye la matriz con el vector.tabulate utilizando las submatrices mencionadas y devolver la función de la matriz resultante.

1.3.3 Función de multiplicación de matrices paralelas usando el algoritmo de Strassen (multStrassenPar)

Este código representa una versión optimizada y paralelizada del algoritmo de multiplicación de matrices de Strassen. Al recibir dos matrices cuadradas de la misma dimensión como entrada (asumiendo que esta dimensión es una potencia de 2), primero verifica si la dimensión es 1. En caso afirmativo, realiza directamente la multiplicación de los elementos y devuelve el resultado en una matriz de 1x1.En el caso de matrices de mayor dimensión, se procede a dividir las matrices originales en submatrices más pequeñas mediante la función subMatriz. Cada una de estas submatrices se calcula simultáneamente en paralelo mediante el uso de la función task, generando tareas para calcular las submatrices individuales (a11, a12, ..., b22).Luego, se realizan siete llamadas recursivas para calcular los productos p1 a p7, donde cada llamada recursiva también se ejecuta en paralelo mediante task. Estas operaciones recursivas involucran

tanto operaciones de suma y resta de matrices como la multiplicación de submatrices. Después de que las tareas paralelas p1 a p7 han concluido, se combinan los resultados para calcular las submatrices c11, c12, c21 y c22. Estas operaciones de combinación también se llevan a cabo de manera paralela mediante la función join, que asegura que la función principal espere a que todas las tareas paralelas se completan antes de continuar.

En última instancia, se construye la matriz resultante combinando las submatrices calculadas de acuerdo con la estructura de la matriz de salida. Este enfoque paralelo tiene como objetivo mejorar la eficiencia computacional, especialmente en sistemas con múltiples núcleos, al realizar cálculos concurrentes.

1.4 Evaluación Comparativa

1.4.1 Evaluación Comparativa MultMatriz – multMatrizParalelo

Dimension	MultMatriz	multMatrizParalelo	Aceleracion
2x2	0.1059	0.0764	1.386125654
4x4	0.1467	0.0772	1.900259067
8x8	0.1617	0.1702	0.950058754
16x16	0.3055	0.3749	0.814883969
32x32	1.313	1.5999	0.820676292
64x64	3.767	2.2364	1.684403506
256x256	239.8767	132.2865	1.813312016
512x512	2450.2178	1450.0905	1.689699919
1024x1024	20552.5115	11359.6116	1.809261815

¿Cuál de las implementaciones es más rápida?

R./ La implementación más rápida en este caso de estudio fue multMatriz Paralelo

¿De qué depende que la aceleración sea mejor?

R./ multMatrizParalelo y multMatriz, aunque tengan la misma complejidad O(n^3) el tiempo de ejecución de multMatrizParalelo va a ser menor debido que este se ejecuta de manera paralela mientras que la otra se ejecuta de manera secuencial, hay que tener en cuenta que la paralelización acelera el tiempo de ejecución, pero también aumenta la complejidad de la programación y tiene un costo de memoria mas alto

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

R./ con respecto a el reporte de los tiempos de ejecución podemos notar que los mejores casos para usar cada uno serian

Versión secuencial:

Tamaño de los datos: La versión secuencial puede ser más rápida para conjuntos de datos pequeños. Esto se debe a que el costo de la creación y coordinación de múltiples tareas en paralelo puede superar el beneficio de hacer cálculos en paralelo.

Capacidad de procesamiento: Si se está ejecutando en una máquina con un solo núcleo o con pocos núcleos, la versión secuencial puede ser más rápida. Esto se debe a que no hay suficientes núcleos para ejecutar tareas en paralelo de manera efectiva.

Versión paralela:

Tamaño de los datos: La versión paralela puede ser más rápida para conjuntos de datos grandes. Esto se debe a que puede dividir el trabajo entre múltiples tareas y realizar cálculos en paralelo.

Capacidad de procesamiento: Si se está ejecutando en una máquina con muchos núcleos, la versión paralela puede ser más rápida. Esto se debe a que puede aprovechar todos los núcleos disponibles para realizar cálculos en paralelo.

1.4. Evaluación Comparativa

multMatrizRec vs **multMatrizRecSec** vs **MultiplicaciónRecPar**

Dimension	multMatrizRec	multMatrizRecSec	multMatrizRecPar
2x2	0.0995	0.1229	0.0346
4x4	0.1032	0.1299	0.1065
8x8	0.8094	0.5195	0.2541
16x16	2.8663	2.3532	0.9634
32x32	5.6327	6.0887	5.7342
64x64	47.272	52.7671	52.4875
256x256	3389.3227	3673.7768	3666.9243
512x512	27903.2818	28421.8635	29324.7129

¿Cuál de las implementaciones es más rápida?

R./ La implementación multmatrizrecpar desde 2x2 hasta 64x64 es ligeramente mas rápida que las otras y de 256x256 hasta 512x512 fue multMatrizRec, podemos notar que la recursión con paralelismo fue la más rápida entre todos los campos probados

¿De qué depende que la aceleración sea mejor?

R./ En este caso la aceleración de multmatrizrecpar fue mejor debido a el uso de paralelización y aprovechamiento eficiente de los recursos del sistema

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

R./ El uso de la paralelización en las multiplicaciones recursivas, como en multMatrizRecPar, puede ser beneficioso en ciertos casos:

Tamaño de los datos: La paralelización puede ser más eficiente para conjuntos de datos grandes. Esto se debe a que puede dividir el trabajo entre múltiples tareas y realizar cálculos en paralelo.

Capacidad de procesamiento: Si se está ejecutando en una máquina con muchos núcleos, la paralelización puede ser más rápida. Esto se debe a que puede aprovechar todos los núcleos disponibles para realizar cálculos en paralelo.

Naturaleza del problema: Algunos problemas se prestan bien para la paralelización. Por ejemplo, si los cálculos son independientes entre sí, entonces la paralelización puede ser más rápida

1.4.3

Evaluación Comparativa MultStrassen VS MultStrassenPar

Dimension	MultStrassen	MultStrassenPar	Aceleracion
2x2	0.522601	0.313	1.669651757
4x4	0.2771	0.3948	0.701874367
8x8	1.0612	1.212	0.875577558
16x16	1.3533	2.6689	0.507062835
32x32	7.6268	6.1307	1.244034123
64x64	69.3839	47.6033	1.45754391
256x256	3783.8505	2722.9398	1.389619594
512x512	21009.5319	18310.8597	1.147380966
1024x1024	150042.3355	79924.5285	1.877300227

¿Cuál de las implementaciones es más rápida?

R./ Podemos observar que entre los 2 algoritmos en las matrices mas pequeñas multStrassen es más rápida que multStrassenPar debido a que el costo de la creación de las tareas paralelas se ve superado, pero en caso de matrices grandes multStrassen consigue sacar un gran tiempo de ventaja en maquinas con múltiples núcleos de procesador ya que puede realizar varias operaciones al mismo tiempo

¿De qué depende que la aceleración sea mejor?

R./ La aceleración de ejecución de algoritmos depende de

Número de núcleos de procesador: La cantidad de núcleos en una máquina afecta la aceleración potencial, ya que más núcleos permiten realizar más cálculos simultáneamente.

Tamaño de los datos: Los algoritmos paralelos son más eficientes para conjuntos de datos grandes, ya que el costo de crear y coordinar tareas paralelas se amortiza con el tiempo ahorrado al realizar cálculos en paralelo.

Naturaleza del problema: La eficiencia de la paralelización depende de la naturaleza del problema. Problemas con cálculos independientes entre sí son más propensos a beneficiarse de la paralelización.

Balance de carga: Si las tareas no están equilibradas y algunas toman mucho más tiempo que otras, algunos núcleos pueden quedar inactivos, reduciendo la aceleración general. Un buen balance de carga es crucial para maximizar la eficiencia de la paralelización.

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

R./ Como hemos visto anteriormente lo mas recomendable sobre que versión de el algoritmo usar es que si las matrices no son extremadamente grandes la versión secuencial es buena elección, pero si son matrices mucho mas grandes y necesitamos un rápido mas rápido de ejecución el algoritmo paralelo es mejor debido a la distribución de tareas en el equipo

1.5. Implementando el producto punto usando paralelismo de datos

Dimension	ProdPunto	ProdPuntoParD	Aceleracion
2x2	0.0287	0.4132	0.06945789
4x4	0.0404	0.7268	0.055586131
8x8	0.031	1.0721	0.028915213
16x16	0.0454	1.0478	0.04332888
32x32	0.062	0.9205	0.067354699
64x64	0.0479	1.2381	0.038688313
256x256	0.1432	0.8554	0.167407061
512x512	0.1688	0.9077	0.185964526
100000x100000	7.5497	3.4327	2.199347452

Cuando es mejor usar ProdPunto/ProdPuntoParD:

ProdPunto:

Tamaño de los datos: prodPunto puede ser más eficiente para vectores pequeños. Esto se debe a que el costo de la creación y coordinación de tareas paralelas en prodPuntoParD puede superar el beneficio de realizar cálculos en paralelo.

Capacidad de procesamiento: Si se está ejecutando en una máquina con un solo núcleo o con pocos núcleos, prodPunto puede ser más eficiente. Esto se debe a que no hay suficientes núcleos para ejecutar tareas en paralelo de manera efectiva.

prodPuntoParD:

Tamaño de los datos: prodPuntoParD puede ser más eficiente para vectores grandes. Esto se debe a que puede dividir el trabajo entre múltiples tareas y realizar cálculos en paralelo.

Capacidad de procesamiento: Si se está ejecutando en una máquina con muchos núcleos, prodPuntoParD puede ser más eficiente. Esto se debe a que puede aprovechar todos los núcleos disponibles para realizar cálculos en paralelo

¿Sera practico construir versiones de los algoritmos de multiplicación de matrices del enunciado, para la coleccion ParVector y usar prodPuntoParD en lugar de prodPunto?

La implementación en paralelo del cálculo del producto escalar no demostró ser ventajosa, excepto bajo una condición particular. Esta excepción ocurre cuando el tamaño del vector alcanza un valor elevado de elementos, por ejemplo, cien mil elementos; en tal escenario, se observa un incremento en la eficacia. No obstante, para las demás evaluaciones, la versión secuencial superó en rendimiento a la paralela. Los datos correspondientes a este experimento se encuentran documentados en la tabla.