INFORME TALLER 4

Pedro Bernal Londoño - 2259548

Jota Emilio López Ramírez – 2259394

Docente:

Carlos delgado

Universidad del Valle sede Tuluá

Facultad de ingeniería

Ingeniería de sistemas

Funciones implementadas:

multMatriz:

La función multMatriz multiplica dos matrices numéricas, que son m1 y m2. Se usa la longitud de m1, la transpuesta de m2 y el producto escalar de dos vectores para calcular el valor de cada elemento del resultado. De igual forma, la función crea y devuelve una nueva matriz usando el método tabulate, que recibe una función que indica cómo calcular cada elemento. Con esto, logramos que devuelva la matriz final, correspondientes a la multiplicación de las matrices de entrada.

multMatrizPar:

La función multMatrizPar multiplica dos matrices numéricas de forma paralela usando el método tabulate para crear dos matrices, la palabra clave task para crear tareas asíncronas, el método length para obtener la longitud de la primera matriz, la función utils.transpuesta para obtener la transpuesta de la segunda matriz, la función utils.prodPunto para obtener el producto escalar de dos vectores, y el método join para esperar el resultado de las tareas. Con esto, logramos que devuelva la matriz final, correspondientes a la multiplicación de las matrices de entrada.

multMatrizRecursiva:

La función multMatrizRecursiva recibe dos matrices como argumentos, llamadas m1 y m2, y devuelve una nueva matriz como resultado, que es el producto de las dos matrices originales. Para esto se realizan en los siguientes pasos:

- Comprueba si la longitud de la primera matriz m1 es igual a 1. Esto significa que las matrices son de tamaño 1x1, y por lo tanto se pueden multiplicar directamente.
- Si la longitud de la primera matriz m1 no es igual a 1, entonces la función asume que las matrices son de tamaño nxn, donde n es una potencia de 2.
- La función se llama a sí misma recursivamente para multiplicar las submatrices entre sí, y luego suma o resta los resultados para obtener las submatrices de la matriz producto, usando la función sumMatriz.
- La función combina las submatrices en una sola matriz, usando la función combinarMatrices, que recibe cuatro matrices y devuelve una matriz que las une en forma de cuadrado. Esta matriz es la matriz producto que devuelve la función.

multMatrizRecursivaPar:

multMatrizRecursivaPar multiplica dos matrices numéricas de forma recursiva y paralela usando una variante del algoritmo de Strassen, que divide las matrices en submatrices, calcula productos y sumas de submatrices, ejecuta productos de forma simultánea, y combina las sumas en una matriz final. La función recibe dos matrices como argumentos, llamadas m1 y m2, y devuelve una nueva matriz como resultado, que es el producto de las dos matrices originales. Para esto, realiza lo siguiente:

- Comprueba si la longitud de la primera matriz m1 es igual a 1. Esto significa que las matrices son de tamaño 1x1, y por lo tanto se pueden multiplicar directamente.

- Si la longitud de la primera matriz m1 no es igual a 1, entonces la función asume que las matrices son de tamaño nxn, donde n es una potencia de 2. En este caso, se aplican los siguientes pasos con el algoritmo de Strassen:
    - Divide cada matriz en cuatro submatrices de tamaño n/2 x n/2, usando una función auxiliar llamada utils.subMatriz.
    - Calcula cuatro productos de submatrices usando la misma función multMatrizRecursiva, de forma recursiva.
    - Calcula estos cuatro productos de forma paralela, usando la palabra clave parallel.
    - Calcula cuatro sumas de productos de submatrices usando una función auxiliar llamada utils.sumMatriz.
    - Combina las cuatro sumas de productos de submatrices en una sola matriz de tamaño nxn, usando una función auxiliar llamada utils.combinarMatrices.

Con estas tareas ejecutadas se devuelve la matriz AB de dimensión n×n.

multStrassen:

La función recibe dos matrices como argumentos, llamadas m1 y m2, y devuelve una nueva matriz como resultado, que es el producto de las dos matrices originales. Para cumplir con lo que se pide, la función hace lo siguiente:

- Si las matrices son de tamaño 1x1, devuelve el producto de sus únicos elementos.
- Si las matrices son de tamaño nxn, donde n es una potencia de 2, divide cada matriz en cuatro submatrices de tamaño n/2 x n/2, calcula siete sumas y restas de submatrices, calcula siete productos de submatrices de forma recursiva y paralela, calcula cuatro sumas y restas de productos de submatrices de forma paralela, y combina las cuatro sumas y restas en una matriz final de tamaño nxn.

Gracias a esto se cumple el multiplicar dos matrices numéricas de forma recursiva, usando el algoritmo de Strassen optimizado.

multStrassenPar:

La función recibe dos matrices como argumentos, llamadas m1 y m2, y devuelve una nueva matriz como resultado, que es el producto de las dos matrices originales. Para esto, la función hace lo siguiente:

- Si las matrices son de tamaño 1x1, devuelve el producto de sus únicos elementos.
- Si las matrices son de tamaño nxn, donde n es una potencia de 2, divide cada matriz en cuatro submatrices de tamaño n/2 x n/2, calcula siete sumas y restas de submatrices, calcula siete productos de submatrices de forma recursiva y paralela, usando la palabra clave parallel y la clase task, calcula cuatro sumas y restas de productos de submatrices de forma paralela, usando la palabra clave parallel y la clase task, y combina las cuatro sumas y restas en una matriz final de tamaño nxn.

Cumpliendo con multiplicar dos matrices numéricas de forma paralela, usando el algoritmo de Strassen optimizado.

sumMatriz:

La función recibe dos parámetros; que son, dos matrices m1 y m2, en donde se asume que tienen el mismo tamaño. La función devuelve una matriz que es la suma de las dos matrices originales, es decir, que cada elemento de la matriz resultante es la suma de los elementos correspondientes de las matrices m1 y m2. Para crear la matriz suma, la función usa la función tabulate del objeto Vector, que crea un vector en dos dimensiones a partir de una función que recibe dos índices (i, j) y devuelve el elemento correspondiente.

submatriz:

Esta función recibe cuatro parámetros que son, una matriz m, dos enteros i y j, y un entero l. La función devuelve una submatriz de la matriz m, que tiene un tamaño de l x l, y que empieza desde la posición (i, j) de la matriz original. Para crear la submatriz, la función usa la función tabulate del objeto Vector, que crea un vector de dos dimensiones a partir de una función que recibe dos índices (x, y) y devuelve el elemento correspondiente.

restaMatriz:

La función recibe dos matrices m1 y m2, que se asume que tienen el mismo tamaño. La función devuelve una matriz que es la resta de las dos matrices originales, es decir, que cada elemento de la matriz resultante es la resta de los elementos correspondientes de las matrices m1 y m2. Para crear la matriz resta, la función usa la función tabulate del objeto Vector, que crea un vector en dos dimensiones a partir de una función que recibe dos índices (i, j) y devuelve el elemento correspondiente.

Informe de desempeño de las funciones secuenciales y de las funciones paralelas:

Existen 3 implementaciones para comparar distintos casos:

1. Multiplicación de matrices de forma secuencial vs Multiplicación de matrices de forma concurrente
2. Producto punto de vectores de forma secuencial vs Producto punto de vectores de forma concurrente
3. Multiplicación de matrices de forma secuencial vs Multiplicación de matrices de forma concurrente utilizando ParVector.

puede elegir entre las siguientes implementaciones:

- new MultiplicacionMatriz().multMatriz,
- new MultiplicacionMatriz().multMatrizPar,
- new MultiplicacionMatriz().multMatrizRecursiva,
- new MultiplicacionMatriz().multMatrizRecursivaPar,
- new MultiplicacionMatriz().multStrassen,
- new MultiplicacionMatriz().multStrassenPar

Para los casos 1 y 3 puede ajustar el tamaño que se van a multiplicar, es importante tener en cuenta que el tamaño de la matriz debe ser potencia de 2.

- i <- 1 to 10
- m1 = utils.matrizAlAzar(math.pow(2, i).toInt, 2)
- m2 = utils.matrizAlAzar(math.pow(2, i).toInt, 2)

Para el caso 2 puede ajustar el tamaño de los vectores que se calcular el producto punto, en este caso no es necesario que el tamaño sea potencia de 2.

- i <- 1 to 10
- ...
- benchmark.compararProdPunto(math.pow(10, i).toInt)

Tablas:

multMatriz vs multMatrizPar:

1. multMatriz vs multMatrizPar

Dimension	Secuencial	Paralelo	Aceleración
2x2	0.141129	0.431482	0.32707969277976834
4x4	0.123936	0.480266	0.2580569934161486
8x8	0.265727	1.299917	0.20441843594629502
16x16	0.565938	0.522054	1.08406026962728
32x32	2.767554	2.5659	1.0785899684321292
64x64	6.285167	4.272541	1.471060663900007
128x128	55.105048	32.384131	1.7016065059766465
256x256	457.417766	243.14107	1.8812854858292758
512x512	3534.412609	2526.817195	1.3987607081326672
1024x1024	29265.957697	16304.400746	1.7949729127076255

multMatrizRecursiva vs multMatrizRecursivaPar:

Dimension	secuencial	paralelo	Aceleracion
2x2	0.3086	0.422969	0.7296042972416418
4x4	0.382943	1.141895	0.33535745405663386
8x8	1.669153	2.745051	0.6080590123826479
16x16	5.041975	6.63384	0.7600386804625978
32x32	18.606131	30.370736	0.6126335232705589
64x64	182.632	84.309855	2.166199906286163
128x128	1364.688522	677.243049	2.015064641881618
256x256	8310.428088	4824.307931	1.7226155972753971
512x512	57097.969473	29017.092737	1.9677357063477892
1024x1024	398216.434726	224919.118202	1.7704872663086006

multStrassen vs multStrassenPar:

Dimension	Secuencial	Paralelo	Aceleración
2x2	0.212623	0.266244	0.7986020342242455
4x4	0.344653	0.663266	0.5196301333100144
8x8	0.555522	0.821276	0.6764132885899502
16x16	3.737419	3.846555	0.9716275992413992
32x32	13.186677	20.495109	0.6434060438517307
64x64	102.131656	55.481477	1.84082438901185
128x128	729.914031	391.808502	1.8629356618708597
256x256	4843.947296	2613.450395	1.8534682369588271
512x512	32472.626935	18973.739096	1.711451115180866
1024x1024	237095.441557	136106.809686	1.74198074368198

multMatrizParD vs multMatrizParParD - Utilizando ParVector:

Dimension	Secuencial	Paralelo	Aceleración
2x2	1.558168	0.584826	2.6643275093788583
4x4	2.026636	1.369114	1.480253653092438
8x8	9.181567	1.947989	4.713356697599422
16x16	21.560716	4.496052	4.795477454442253
32x32	69.17674	19.190857	3.6046717455088113
64x64	313.617093	202.881212	1.5458163420277675
128x128	1421.377187	1354.408526	1.0494449493741596
256x256	6643.26411	16012.428625	0.4148817312839076
512x512	28793.905265	92478.731769	0.3113570516616023
1024x1024			

Comparar prodPunto y prodPuntoPar con diferentes longitudes de vectores:

Tamaño del Vector	Secuencial	Paralelo	Aceleración
10	0.059793	1.084469	0.05513573924197004
100	0.083038	0.573799	0.1447161811017447
1000	0.609687	0.692174	0.8808290978857918
10000	0.929695	2.090869	0.4446452647200757
100000	5.890912	5.725828	1.0288314633272253
1000000	34.775707	31.030398	1.1206980651682261
10000000	268.999511	848.385006	0.3170724483548923
100000000			

A la hora de la compilación de los resultados para esta tabla, se lanzó una excepción que no nos permitió obtener los valores correspondientes para el ultimo campo.

Aquí la excepción:

Exception in thread "main" java.lang.OutOfMemoryError: Java heap space

Análisis comparativo de las diferentes soluciones:

multMatriz vs multMatrizPar:

Se puede observar que la aceleración varía entre 0.239 y 0.421 y la aceleración más alta se da cuando las matrices son de 1024x1024, lo que significa que la versión paralela no es tan rápida como la versión secuencial. A medida que disminuye el tamaño de las matrices, la aceleración disminuye hasta un mínimo de 0.239 cuando las matrices son de 8x8, lo que significa que la versión paralela es más de cuatro veces más lenta que la versión secuencial.

multMatrizRecursiva vs multMatrizRecursivaPar:

Para este caso, notamos que la aceleración varía entre 0.239 y 0.729, lo que indica que la versión paralela no es siempre más rápida que la versión secuencial. De hecho, la aceleración más alta se da cuando las matrices son de 2x2, lo que significa que la versión paralela es solo un 27.9% más rápida que la versión secuencial. De igual forma, a medida que aumenta el tamaño de las matrices, la aceleración disminuye hasta un mínimo de 0.239 cuando las matrices son de 8x8, lo que significa que la versión paralela es más de cuatro veces más lenta que la versión secuencial. Teniendo en cuenta esto, a partir de ese punto, la aceleración aumenta ligeramente hasta un máximo de 0.421 cuando las matrices son de 1024x1024, lo que significa que la versión paralela es solo un 42.1% más rápida que la versión secuencial.

multStrassen vs multStrassenPar

Se puede observar que el algoritmo paralelo es mucho más rápido que el algoritmo secuencial, especialmente cuando la dimensión de las matrices es grande. Por ejemplo, para matrices de 1024x1024, el algoritmo paralelo tarda 136106.80 segundos, mientras que el algoritmo secuencial tarda 3274.24 segundos. Pudiendo deberse a que el algoritmo paralelo aprovecha el poder de cómputo de múltiples recursos, reduciendo así la complejidad temporal del problema.

multMatrizParD vs multMatrizParParD - Utilizando ParVector:

Se puede observar que el algoritmo Secuencial tiene un mayor rendimiento que el algoritmo Paralelo cuando la dimensión del arreglo es pequeña, es decir, menor o igual a 16. Esto se debe a que el algoritmo Paralelo tiene un costo adicional de dividir y combinar los subarreglos, lo que hace que su tiempo de ejecución sea mayor que el algoritmo Secuencial para arreglos pequeños. Sin embargo, cuando la dimensión del arreglo es

grande, es decir, mayor o igual a 32, el algoritmo Paralelo tiene un mayor rendimiento que el algoritmo Secuencial. Esto se debe a que el algoritmo Paralelo aprovecha el poder de cómputo de múltiples recursos, reduciendo así la complejidad temporal del problema.

Comparar prodPunto y prodPuntoPar con diferentes longitudes de vectores:

El algoritmo Secuencial tiene un mayor tiempo de ejecución que los otros dos algoritmos, especialmente cuando el tamaño del vector es grande y se podría decir que este es el mas simple y el menos eficiente en este caso. Por el contrario, el algoritmo Paralelo es más complejo y más eficiente que el algoritmo Secuencial. Por ende, se puede concluir que el algoritmo paralelo es el más eficiente y escalable de los tres algoritmos para resolver el problema.

Análisis general para los paralelos y secuencial:

Los algoritmos paralelos muestran ser más rápido y eficientes que los algoritmos secuenciales para ordenar arreglos grandes, pero tiene un costo adicional y una complejidad mayor que pueden afectar su calidad y fiabilidad. En contraste, los algoritmos Secuenciales son más lentos e ineficientes que los algoritmos Paralelos para ordenar arreglos grandes, pero eficientes cuando estos son pequeños, además que tienen un costo bajo y una simplicidad mayor que pueden favorecer su calidad y fiabilidad.

Conclusiones:

- La implementación de la multiplicación de matrices de forma concurrente no siempre es más rápida que la implementación secuencial, esto se debe a que el costo de crear los hilos y sincronizarlos es mayor que el costo de realizar la operación de forma secuencial.
- Algunas implementaciones de la multiplicación de matrices de forma concurrente son más rápidas que otras, esto se debe a que algunas implementaciones utilizan más hilos que otras, por lo que el costo de crear los hilos y sincronizarlos es mayor que el costo de realizar la operación de forma secuencial.
- Si la aceleración es mayor que 1, significa que el algoritmo paralelo es más rápido que el algoritmo secuencial. Si la aceleración es menor que 1, significa que el algoritmo paralelo es más lento que el algoritmo secuencial. Si la aceleración es igual a 1, significa que el algoritmo paralelo y el algoritmo secuencial tienen el mismo tiempo de ejecución.

Pequeña definición:

El algoritmo Secuencial es aquel que busca el elemento en el vector de forma lineal, recorriendo cada posición hasta encontrarlo o llegar al final. El algoritmo Paralelo es aquel

que divide el vector en subvectores y los busca simultáneamente usando varios procesadores o hilos de ejecución, y luego combina los resultados.