TALLER 4 – MULTIPLICACIÓN DE MATRICES EN PARALELO

LAURA SOFIA PEÑALOZA LÓPEZ - 2259485 - 3743

ESMERALDA RIVAS GUZMÁN - 2259580 - 3743

LAURA TATIANA COICUE POQUIGUEGUE – 2276652-3743

DIRECTOR

CARLOS ANDRES DELGADO

UNIVERSIDAD DEL VALLE SEDE TULUÁ

FACULTAD DE INGENIERÍA

PROGRAMA ACADÉMICO DE INGENIERÍA DE SISTEMAS

CURSO FUNDAMENTOS DE PROGRAMACIÓN FUNCIONAL Y CONCURRENTE (750013C)

TULUÁ – VALLE DEL CAUCA

2023

PRIMERA PARTE

1. Funciones implementadas

1.1 versión estándar secuencial

- la función def multMatriz recibe dos parámetros m1, m2 de tipo matriz y devuelve una matriz. La función crea dos variables la primera llamada t_m2, calcula la transpuesta de la matriz m2, en la segunda variable n, calcula la dimensión de la matriz m1. Vector. Tabule crea un vector de tamaño n x n esta función se aplica a cada posición de la matriz el i y j son los índices de la fila y la columna. La función prodPunto calcula el producto punto entre los elementos de las matrices m1, m2.

2. Multiplicación recursiva de matrices

2.1 Extrayendo submatrices clase recursiva

- La función subMatriz recibe tres parámetros m de tipo obj.Matriz, (i, j) de tipo entero y de tipo entero, devuelve una nueva matriz obj.Matriz. (i, j) son los índices de inicio y (l) representa el tamaño de la subMatriz extraída de la matriz original. La función vector.tabulate crea una nueva matriz (x, y) define las coordenadas de la nueva matriz la función devuelve un elemento ubicado en la fila (i + x) y la columna (j + y) de la matriz original m.

2.2 Sumando matrices

- La función sumMatriz recibe dos parámetros m1, m2 de tipo obj.Matriz y devuelve una nueva matriz obj.Matriz, se crea una variable n la cual almacena el resultado final de la funcion m1.map la cual transforma cada elemento de la colección aplicando una función dada (_ => 1), a cada elemento de m1 se le asigna un valor 1 creando una nueva colección, el foldLeft combina los elementos de la colección utilizando la función (_+_) y un valor inicial que es cero. La función vector.tabulate crea la nueva matriz con sus dimensiones n x n, donde se sumarán ambas matrices.

2.3 Multiplicando matrices recursivamente, versión secuencial

- La función multMatrizRec recibe como parámetro m1, m2 de tipo obj.Matriz devuelve una nueva matriz obj.Matriz. Esta función divide recursivamente las matrices en submatrices más pequeñas realizando operaciones de suma y multiplicación, se combinan estas sumas para formar la matriz resultante.

- la función multMatrizPar recibe m1,m2 de tipo matriz y devuelve una matriz, crea tres variables t_m2 calcula la transpuesta de la matriz m2, n calcula la longitud de m1 la variable m crea la nueva matriz con dimensiones n x n, para cada par de índices (i, j) se ejecuta un trabajo 'task' que calcula el producto punto entre las filas de m1, m2, el vector.tabulate crea un vector bidimensional de tamaño n x n donde para cada índice (i, j) se ejecuta un trabajo que calcula el producto punto entre las matrices m1 y m2.

2.4 Multiplicando matrices recursivamente, versión paralela

- La función multMatrizRecPar recibe como parámetro dos matrices m1, m2 de tipo obj.Matriz que devuelve una nueva matriz de tipo obj.matriz, divide las matrices en submatrices, se crean casos el primero es si el valor coincide con 1 se crea un vector bidimensional que contiene el resultado de multiplicar el primer elemento de la fila de la matriz m1 con el primer elemento de la fila de la matriz m2, el segundo caso es cuando haya cualquier cosa se crean variables donde se subdividen las matrices, la última variable de este caso crea cuatro procesos los cuales ejecutan tareas que contienen resultados de suma y multiplicación de las matrices m1 y m2, son tareas que son por separadas pero que se ejecutan al mismo tiempo. La función vector.tabulate crea la nueva matriz con su dimensión de n x n y con los resultados de los procesos.

3. Multiplicación de matrices usando el algoritmo de Strassen

3.1 Restando matrices

- La función restaMatriz recibe dos matrices de tipo obj.Matriz y devuelve una obj.Matriz, se crea una variable n la cual almacena el resultado final de la funcion m1.map la cual transforma cada elemento de la colección aplicando una función dada (_ => 1), a cada elemento de m1 se le asigna un valor 1 creando una nueva colección, el foldLeft combina los elementos de la colección utilizando la función (_+_) y un valor inicial que es cero. La función vector.tabulate se crea una nueva matriz con sus dimensiones e índices los cuales son las filas y las columnas en esta función se realiza la resta de las matrices.

3.2 Algoritmo de Strassen, versión secuencial

- La función multStrassen recibe como parámetro dos matrices de tipo obj.Matriz, calcula la longitud de la matriz m1, divide las matrices en submatrices, luego con cada una de ellas realiza operaciones de multiplicación, resta y suma, con la función vector.tabulate se

crea la nueva matriz con base a los resultados de las operaciones que ya se realizaron con las submatrices.

3.3 Algoritmo de Strassen, versión paralela

- La función multStrassenPar recibe m1, m2 de tipo obj.Matriz que devuelve una nueva matriz de tipo obj.matriz, crea una variable n de tipo entero la cual almacena una nueva colección, z calcula la división n/s, se realizan dos casos el primero que si el dato es igual a 1 crea un vector bidimensional que contiene el resultado de multiplicar el primer elemento de la fila de la matriz m1 con el primer elemento de la fila de la matriz m2, en el segundo caso cuando hay cualquier cosa se crean variables, las primeras variables 'm1_1' contienen las submatrices de m1, m2 las segundas variables 'p_1' calculan la multiplicación, resta y suma de más matrices para al final en una variable realizar cuatro procesos los cuales ejecutan 4 tareas donde se calculan la suma y resta de las variables 'p_1'. La función vector.tabulate crea la nueva matriz con sus dimensiones y sus índices (i, j).

4. Implementación de los test

4.1 Para la creación de los test lo que se realizó fue una variación de matrices de distintos tamaño, respetando la condición de que solo tuviera 0´s y 1´s; partiendo de esto utilizamos excel para calcular de forma mas rapida y verídica la respuesta de la multiplicación de las matrices, donde se ingresaban los valores de las matrices a multiplicar, y excel nos arrojaba el resultado. Luego de esto pusimos los test en los diferentes algoritmos, y lo que hicimos fue colocar el llamado de las diversas funciones, ya sean paralelas o secuenciales para verificar y estar seguras de que los algoritmos programados estuvieran funcionando de forma correcta.

SEGUNDA PARTE

Las pruebas para los algoritmos se realizaron gracias a la implementación del algoritmo de BenchMarck, basándose en generar matrices de potencia 2, con estas pruebas donde obtuvimos el tiempo secuencial, paralelo y la aceleración. Se pudo evidenciar la diferencia en los comportamiento de los dos algoritmos, en las tablas se puede observar que los tiempos son aleatorios ya que en algunas matrices el tiempo secuencial es más eficiente que el paralelo y en otras matrices es lo contrario,aunque en la gran medida el paralelismo representó mejoras en los tiempos de ejecución; sumado a esto, la aceleración en todos los casos fue diferente, en donde la aceleración fue menor a 1 nos indica que la implementación secuencial es mas rapida, y si es mayor nos indica que la paralela es mas rapida. A la hora de la compilación se vio muy reflejado la variación de tiempo en el cual se ejecutaban los algoritmos, en la matriz donde hubo mayor demora y dificultad a la hora de la compilación fue en la de 1024 x 1024.

1. Desempeño de la función MultMatriz

Dimensiones matrices	Tiempo secuencial	Tiempo paralelo	Aceleración
2x2	0.1481	0.1481	0.8555748
4x4	0.109	0.1483	0.7349966
8x8	0.191	1.1854	0.1611270
16x16	62.6244	0.6308	99.277742
32x32	1.3637	1.034	1.3188588
64x64	8.1499	4.5285	1.7996908
128x128	64.9013	53.3299	1.2169777
256x256	523.8756	353.3875	1.4824395
512x512	4439.9091	2763.9727	1.6063505
1024x1024	31592.7181	22687.1303	1.3992539

2. Desempeño de las funcion Recursiva

Dimensiones matrices	Tiempo secuencial	Tiempo paralelo	Aceleración
2x2	0,1921	0.1525	1.259671
4x4	0.2057	0.1859	1.106508
8x8	1.1187	0.6797	1.645873
16x16	2.6117	1.0567	2.471562
32x32	9.5711	4.9561	1.931175
64x64	65.9793	57.2068	1.153347
128x128	540.3458	354.4884	1.524297
256x256	4945.2046	3318.6283	1.490135
512x512	47390.1503	26007.981	1.822138
1024x1024	379817.0042	209959.8733	1.80899

3. Desempeño de las funcione de Strassen

Dimensiones matrices	Tiempo secuencial	Tiempo paralelo	Aceleración
2x2	0.2326	0.2396	0.9707846
4x4	0.434	0.169	2.5680473
8x8	0.6607	1.5474	0.4269742
16x16	5.4582	3.5752	1.5266838
32x32	19.0338	30.5018	0.6240221
64x64	193.9364	330.2502	0.5872408
128x128	1256.1488	1234.5834	1.0174677
256x256	7928.6208	7614.8251	1.0412085
512x512	77069.1125	78790.9566	0.9781466
1024x1024	379817.0042	209959.8733	1.80899806

4. ProductoPunto

Dimensiones matrices	Producto punto Vector	Producto punto ParVector	Aceleración
2x2	0.057	1.2806	0,044510
4x4	0.0259	1.5894	0.01629545
8x8	0.0274	1.3118	1.02088
16x16	0.0899	0.9119	0.09858
32x32	0.0371	0.736	0.05040
64x64	0.0887	0.4434	0.200045
128x128	0.087	0.35	0.2485
256x256	0.0993	0.4641	0.2139
512x512	0.153	2.0383	0.07506
1024x1024	0.2644	0.6445	0.41024

TERCERA PARTE

1. Análisis comparativo de las diferentes soluciones, versión secuencial versión paralela

PREGUNTAS

- ***¿Cuál de las implementaciones es más rápida?***

Luego de implementar y compilar todas las versiones de multiplicación de matrices, podemos afirmar que la primera función(multMatriz), es la más eficiente, tanto la versión secuencial como la versión paralela, son más rápidas a comparación de los resultados de los otros algoritmos. Ahora, analizando más a fondo los resultados, las matrices de tamaño pequeño se resolvieron de forma más eficaz con el algoritmo secuencial, pero cuando son medianas(16x16) su eficiencia empieza a ser mayor con el algoritmo paralelo.

- ***¿De qué depende que la aceleración sea mejor?***

La aceleración depende netamente de la diferencia de valores entre los tiempos secuenciales y paralelos; entre más cercanos estén los valores, su aceleración será menor, pero si sus valores son muy distintos, su aceleración será más alta. Podríamos interpretar, que entre la aceleración sea más baja los valores comparados son cercanos, pero si la aceleración es un número considerable quiere decir que el tiempo secuencial y paralelo están bastante alejados.

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

La primera función implementada(multMatriz) las matrices de tamaño pequeño(2x2,4x4 y 8x8) los compila de forma más rápida con la función secuencial, pero desde la matriz de tamaño 16x16 empieza a ser más eficiente el algoritmo paralelo

La segunda función implementada es la función recursiva, en este caso, para todas las matrices la versión paralela es más eficiente que la versión secuencial

En la tercera función (Strassen) se puede evidenciar que tanto los tiempos como la aceleración son variantes; las matrices con tamaños más pequeños y más grandes son más

eficientes con el algoritmo paralelo, pero las matrices de tamaño medio son eficientes con el algoritmo secuencial.

Por último, en la comparación del producto punto se evidencia que en la mayoría de casos el uso del Vector era más eficiente que el ParVector, en los tiempos de las primeras matrices, su diferencia es notoria, a medida que la matriz evaluada se hacia mas grande su diferencia se reducía, el comportamiento donde el Vector era más eficiente que el ParVector solo cambio en 1 matriz(64x64), luego de esta siguió el mismo comportamiento