Taller Paralelización de Tareas

Estudiante: Juan Sebastian Getial Getial

Código: 202124644

Asignatura: Fundamentos de Programación Funcional y Concurrente

Docente: Juan Francisco Diaz Frias

Universidad del Valle

Santiago de Cali

2022

Informe de Corrección

Argumentación Sobre la Corrección

Preliminares

A continuación se tendrá en cuenta la siguiente definición:

Matriz.

$\forall\, m \in Matriz :\ m\ \ == Vector[Vector[Int]]$

Además, es importante resaltar que, todas las funciones se demostraran respecto a matrices cuadradas de tamaño 2^n, donde m1 y m2 son matrices de igual tamaño.
Por último, cabe recalcar que se utilizarán las definiciones brindadas por el profesor Juan Francisco Diaz Frias en el documento Taller 5: Multiplicación de matrices en paralelo. Por lo que recomiendo leer detalladamente dicho documento.

Función multMatriz

A continuación se demostrará que:
$\forall\, m1,\ m2 \in Matriz :\ multMatriz(m1,\ m2)\ \ ==\ \ m1\ \bullet\ m2.$
Para ello, daremos por hecho lo siguiente, debido a que son funciones ya definidas:

$\forall\, m \in Matriz : transpuesta(m)\ \ ==\ \ m^T$

$\forall\, v1,\ v2 \in Vector[Int] : prodEscalar(v1,\ v2)\ ==\ \ v1\ \bullet\ v2$

Demostración.

$multMatriz(m1,\ m2)$
== val l = m1.length
val m2T = transpuesta(m2)
Vector.tabulate(l, l)((i, j) => prodEscalar(m1(i), m2T(j)))
== val l = m1.length
val m2T = $m2^T$
Vector.tabulate(l, l)((i, j) => m1(i) • m2T(j)

Observe que m2T = $m2^T$, por lo que cada columna de m2 es una fila de m2T, por consiguiente, m1(i) • m2T(j) es el elemento a_{ij} de la matriz $m1 \bullet m2$.

Por lo que finalmente

→ Vector.tabulate(l, l)((i, j) => m1(i) • m2T(j)

== $m1 \bullet m2$ ♦

Función multMatrizPar

Esta función es una versión paralela de multMatriz, por lo que su demostración es bastante similar, ya que utiliza la misma lógica con algunas modificaciones para que utilice paralelismo. Por consiguiente, se da por hecho su buen funcionamiento.

Función multMatrizRec

Preliminares.

A continuación se demostrarán brevemente dos funciones auxiliares utilizadas en esta función.

Función subMatriz.

A continuación se demostrará que:

$\forall\, m \in Matriz,\ i,\ j,\ l \in Int : subMatriz(m, i,\ j,\ l) == m'_{ij}$, tal que m'_{ij} es una submatriz de m de tamaño lxl que empieza desde la posición m_{ij} de la matriz.

Demostración

$subMatriz(m, i,\ j,\ l)$

== $Vector.tabulate(l,\ l)((x,\ y) => m(i + x)(y + j))$

Observe que $(i + x)$ y $(y + j)$, son los valores que representan la fila y columna del elemento deseado de la matriz m, los cuales empiezan a aumentar desde los índices que se reciben como parámetro i y j hasta completar la submatriz de tamaño lxl.

Por consiguiente,

== m'_{ij} ♦

Función sumMatriz.

A continuación se demostrará que:

$\forall\, m1,\ m2 \in Matriz : sumMatriz(m1,\ m2) == m1 + m2.$

Demostración

$sumMatriz(m1,\ m2)$

== val l = m1.length

Vector.tabulate(l, l)((i, j) => m1(i)(j) + m2(i)(j))

Observe que "m1(i)(j) + m2(i)(j))" es la suma de los elementos que están en la misma posición de ambas matrices, por consiguiente,

$== m1 + m2$ ♦

Función unirMatriz.

A continuación se demostrará que:

$\forall\, m1,\ m2 \in Matriz : unirMatriz(m1,\ m2) == m12$, donde $m12$ es la matriz resultante de unir las filas de m1 con las filas de m2.

Demostración

$unirMatriz(m1,\ m2)$

== val l = m1.length

Vector.tabulate(l)((i) => m1(i) ++ m2(i))

Observe que "m1(i) ++ m2(i)" es la concatenación de las filas i de cada matriz, por consiguiente,

$== m12$ ♦

De acuerdo a lo anterior, se demostrará que:

$\forall\, m1,\ m2 \in Matriz :\ multMatrizRec(m1,\ m2) == m1 \bullet m2.$

Demostración.

Caso Base.

$multMatrizRec(Vector(Vector(a)),\ Vector(Vector(b)))$

$== Vector(Vector(a * b))$

Demostración

$multMatrizRec(Vector(Vector(a)),\ Vector(Vector(b)))$

$== Vector(Vector(\ m1(0)(0) * m2(0)(0)\ ))$

$==$ $Vector(Vector(a * b))$ ♦

Caso Inducción.

Hipótesis: $multMatrizRec(m1, m2) == m1 \bullet m2$, donde m1 y m2 son de tamaño lxl, tal que $l < L$.

$Hipótesis \Rightarrow multMatrizRec(A, B) == A \bullet B$, donde A y B son de tamaño LxL.

Demostración

$multMatrizRec(A, B)$

Para limpieza del documento procederemos a sustituir los valores correspondientes en cada caso, teniendo en cuenta que tan solo es azúcar sintáctico y las funciones auxiliares ya fueron probadas. Si desea ver el código más a detalle, por favor revise el código fuente.

$==$ $val\ multsLeft =$

$Vector($

$multMatrizRec(A_{11}, B_{11}),$

$multMatrizRec(A_{11}, B_{12}),$

$multMatrizRec(A_{21}, B_{11}),$

$multMatrizRec(A_{21}, B_{12})$

$)$

$val\ multsRight =$

$Vector($

$multMatrizRec(A_{12}, B_{21}),$

$multMatrizRec(A_{12}, B_{22}),$

$multMatrizRec(A_{22}, B_{21}),$

$multMatrizRec(A_{22}, B_{22})$

$)$

Observe que cada A_{ij} y B_{ij} es una submatriz de A y B respectivamente, las cuales tienen un tamaño menor de L, por consiguiente

Hip$\rightarrow$ $val\ multsLeft =$

$Vector($

$A_{11} \bullet B_{11}$,

$A_{11} \bullet B_{12}$,

$A_{21} \bullet B_{11}$,

$A_{21} \bullet B_{12}$

)

$val\ multsRight\ =$

$Vector($

$A_{12} \bullet B_{21}$,

$A_{12} \bullet B_{22}$,

$A_{22} \bullet B_{21}$,

$A_{22} \bullet B_{22}$

)

== $val\ subMTotales\ =$

$Vector($

$A_{11} \bullet B_{11} + A_{12} \bullet B_{21}$,

$A_{11} \bullet B_{12} + A_{12} \bullet B_{22}$,

$A_{21} \bullet B_{11} + A_{22} \bullet B_{21}$,

$A_{21} \bullet B_{12} + A_{22} \bullet B_{22}$

)

== $val\ subMTotales\ =$

$Vector($

C_{11},

C_{12},

C_{21},

C_{22}

)

Finalmente luego de ejecutar la última orden de la función, se obtiene la matriz

== (

$$C_{11}\ C_{12}$$
$$C_{21}\ C_{22}$$

) ♦

Por lo que finalmente se tiene que:

$\forall\, m1,\ m2 \in Matriz:\ multMatrizRec(m1,\ m2)\ ==\ m1\ \bullet\ m2$ ♦

Función multMatrizRecPar

Esta función es una versión paralela de multMatrizRec, por lo que su demostración es bastante similar, ya que utiliza la misma lógica con algunas modificaciones para que utilice paralelismo. Por consiguiente, se da por hecho su buen funcionamiento.

Función multStrassen

A continuación se demostrará que:

$\forall\, m1,\ m2 \in Matriz:\ multStrassen(m1,\ m2)\ ==\ m1\ \bullet\ m2.$

En primer lugar, cabe recalcar que la función auxiliar ***restarMatriz*** presenta una lógica prácticamente igual a la función sumMatriz, con la única diferencia de que esta resta en vez de sumar. Por consiguiente, se da por hecho su buen funcionamiento.

Demostración.

Caso Base.

$multStrassen(Vector(Vector(a)),\ Vector(Vector(b)))$

$==\ Vector(Vector(a\ *\ b))$

Demostración

$multStrassen(Vector(Vector(a)),\ Vector(Vector(b)))$

$==\quad Vector(Vector(\ m1(0)(0)\ *\ m2(0)(0)\ ))$

$==\quad Vector(Vector(a\ *\ b))$ ♦

Caso Inducción.

Hipótesis: $multStrassen(m1,\ m2)\ ==\ m1\ \bullet\ m2$, donde m1 y m2 son de tamaño lxl, tal que $l < L$.

$Hipótesis \Rightarrow multStrassen(A,\ B)\ ==\ A\ \bullet\ B$, donde A y B son de tamaño LxL.

Demostración

$multStrassen(A,\ B)$

Para limpieza del documento procederemos a sustituir los valores correspondientes en cada caso, teniendo en cuenta que tan solo es azúcar sintáctico y las funciones auxiliares ya fueron probadas. Si desea ver el código más a detalle, por favor revise el código fuente.

== $val\ subMatricesM1\ = Vector(A_{11},\ A_{12},\ A_{21},\ A_{22})$

$val\ subMatricesM2\ = Vector(B_{11},\ B_{12},\ B_{21},\ B_{22})$

Observe que a partir de aquí las matrices con las que se empieza a trabajar son submatrices de A y B, por lo que su tamaño es inferior a L.

== $val\ s1\ = B_{12}\ -\ B_{22}$

$val\ s2\ = A_{11}\ +\ A_{12}$

$val\ s3\ = A_{21}\ +\ A_{22}$

$val\ s4\ = B_{21}\ -\ B_{11}$

$val\ s5\ = A_{11}\ -\ A_{22}$

$val\ s6\ = B_{11}\ +\ B_{22}$

$val\ s7\ = A_{12}\ -\ A_{22}$

$val\ s8\ = B_{21}\ +\ B_{22}$

$val\ s9\ = A_{11}\ -\ A_{21}$

$val\ s10\ = B_{11}\ +\ B_{12}$

Hip→ $val\ p1\ = A_{11}\ \bullet\ s1$

$val\ p2\ = s2\ \bullet B_{22}$

$val\ p3\ = s3\ \bullet\ B_{11}$

$val\ p4\ = A_{22}\ \bullet\ s4$

$val\ p5\ = s5\ \bullet s6$

$val\ p6\ = s7\ \bullet\ s8$

$val\ p7\ = s9\ \bullet\ s10$

== $val\ c11\ = p5\ +\ p4\ -\ p2\ +\ p6$

$val\ c12\ = p1 + p2$

$val\ c21\ = p3 + p4$

$val\ c22\ = p5 + p1 - p3 - p7$

Finalmente luego de ejecutar la última orden de la función, se obtiene la matriz

== (

$C_{11}\ C_{12}$

$C_{21}\ C_{22}$

) ♦

Finalmente, se tiene que:

$\forall\ m1,\ m2 \in Matriz:\ multStrassen(m1,\ m2)\ ==\ m1\ \bullet\ m2$ ♦.

Función multStrassenPar

Esta función es una versión paralela de multStrassen, por lo que su demostración es bastante similar, ya que utiliza la misma lógica con algunas modificaciones para que utilice paralelismo. Por consiguiente, se da por hecho su buen funcionamiento.

Informe de Desempeño de las Funciones Secuenciales y Paralelas

En cuanto a los casos de prueba, por comodidad del lector, se recomienda revisar los casos de prueba directamente en el código. Dichos casos de prueba fueron generados especialmente para que su comprobación fuera sencilla, ya que comprobar, por ejemplo, que una multiplicación de matrices 1024x1024 está correcta sería muy engorroso, sin embargo, con el informe de corrección ya se da por sabido que los algoritmos funcionan para matrices de cualquier tamaño potencia de 2.

Por otra parte, en cuanto a los tiempos de ejecución de cada función se obtuvo lo siguiente:

Tamaño Matriz	Tiempo de Ejecucion [ms]					
[nxn]	multMatriz	multMatrizPar	multMatrizRec	multMatrizRecPar	multStrassen	multStrassenPar
2	0.1519	0.0311	0.1382	0.1136	0.0373	0.1526
4	0.0497	0.1391	0.4008	8.7493	0.6651	0.2394
8	0.1038	0.2949	2.0348	1.0661	1.0353	0.7816
16	0.6899	0.4601	9.8331	2.5509	10.8616	4.0711
32	0.8025	0.9382	21.3458	14.0126	12.33	9.183
64	4.9639	4.8017	174.728	77.4579	94.6767	47.8826
128	56.091	40.7494	1234.7042	583.6371	706.8497	347.4492
256	282.3377	286.5665	9645.3296	4868.6725	4902.8252	2443.7612
512	2338.221	2563.2955	82620.5975	38971.3008	35080.4636	18802.8096
1024	20858.0451	25792.7674	639860.0098	311204.1869	243646.6289	120707.447

Los resultados anteriores se obtuvieron a partir de evaluar cada función en el mismo par de matrices de dimensión nxn con ayuda de la función probarAlgoritmo, la cual fue definida personalmente en el paquete Benchmark y está basada en la función compararAlgoritmos del mismo paquete.

Nota: Se intentó probar cada función con 5 parejas de matrices por cada dimensión, sin embargo el tiempo de espera era excesivamente largo(20 horas fue el límite de espera), y la máquina(Portátil) corría el riesgo de sufrir algún daño al estar tanto tiempo activa, por lo que se optó por realizar lo anterior.

Análisis Comparativo

Todos los datos a continuación están basados en la tabla presentada anteriormente.

En primer lugar, se procederá a comparar los algoritmos secuenciales con su versión paralela.

Al comparar el tiempo de ejecución de la función multMatriz con el tiempo de ejecución de su versión paralela y calculando la aceleración, se obtuvieron los siguientes resultados:

multMatriz vs multMatrizPar			
Tamaño	Tiempo de Ejecucion [ms]		Aceleración
Matriz [nxn]	multMatriz	multMatrizPar	
2	0.1519	0.0311	4.884244373
4	0.0497	0.1391	0.3572969087
8	0.1038	0.2949	0.3519837233
16	0.6899	0.4601	1.49945664
32	0.8025	0.9382	0.8553613302
64	4.9639	4.8017	1.033779703
128	56.091	40.7494	1.376486525
256	282.3377	286.5665	0.9852432158
512	2338.221	2563.2955	0.9121933074
1024	20858.0451	25792.7674	0.8086780599

En dichos resultados, se puede apreciar que no se obtuvo una aceleración considerable en ninguno de los casos(exceptuando el primero), e incluso, se obtuvo una desaceleración al utilizar la versión paralela. Por lo que en este caso podemos concluir que la paralelización no sirvió.

Por otro lado, al comparar el tiempo de ejecución de la función multMatrizRec con el tiempo de ejecución de su versión paralela y calculando la aceleración, se obtuvieron los siguientes resultados:

multMatrizRec vs multMatrizRecPar			
Tamaño	Tiempo de Ejecucion [ms]		Aceleracion
Matriz [n*n]	multMatrizRec	multMatrizRecPar	
2	0.1382	0.1136	1.216549296
4	0.4008	8.7493	0.04580937904
8	2.0348	1.0661	1.908638964
16	9.8331	2.5509	3.854757145
32	21.3458	14.0126	1.523329004
64	174.728	77.4579	2.255780237
128	1234.7042	583.6371	2.115534122
256	9645.3296	4868.6725	1.981100516

512	82620.5975	38971.3008	2.120036945
1024	639860.0098	311204.1869	2.056077767

Al analizar estos resultados, se puede observar que si se obtuvo una aceleración considerable al usar la versión paralela, la cual tiende a ser el doble en matrices de tamaño superior a 4. Por lo que en este caso se puede concluir que la paralelización si sirvió.

Por otra parte, al comparar el tiempo de ejecución de la función multStrassen con el tiempo de ejecución de su versión paralela y calculando la aceleración, se obtuvieron los siguientes resultados:

multStrassen vs multStrassenPar			
Tamaño	**Tiempo de Ejecucion [ms]**		**Aceleracion**
Matriz [n*n]	**multStrassen**	**multStrassenPar**	
2	0.0373	0.1526	0.244429882
4	0.6651	0.2394	2.778195489
8	1.0353	0.7816	1.324590583
16	10.8616	4.0711	2.667976714
32	12.33	9.183	1.342698465
64	94.6767	47.8826	1.977267316
128	706.8497	347.4492	2.034397259
256	4902.8252	2443.7612	2.006261987
512	35080.4636	18802.8096	1.865703283
1024	243646.6289	120707.447	2.018488792

De manera similar al caso anterior, se obtuvo una aceleración considerable al utilizar la versión paralela, la cual tiende a ser el doble en la mayoría de matrices. Por consiguiente, se puede concluir que la paralelización si sirvió.

En segundo lugar, se procederá a comparar los algoritmos secuenciales entre sí.

Al comparar el tiempo de ejecución de la función multMatrizRec con el tiempo de ejecución de de la función multStrassen y calculando la aceleración, se obtuvieron los siguientes resultados:

multMatrizRec vs multStrassen			
Tamaño	**Tiempo de Ejecucion [ms]**		**Aceleración**
Matriz [n*n]	**multMatrizRec**	**multStrassen**	
2	0.1382	0.0373	3.705093834
4	0.4008	0.6651	0.6026161479
8	2.0348	1.0353	1.965420651
16	9.8331	10.8616	0.9053086101
32	21.3458	12.33	1.731208435

64	174.728	94.6767	1.84552271
128	1234.7042	706.8497	1.746770494
256	9645.3296	4902.8252	1.967300323
512	82620.5975	35080.4636	2.355174049
1024	639860.0098	243646.6289	2.626180435

Como se puede observar, en general el tiempo de ejecución de multStrassen resultó ser mucho menor en la mayoría de los casos, con una aceleración que tiende a ser el doble, por lo que resultó ser más eficiente.

Por otra parte, al comparar el tiempo de ejecución de la función multStrassen con el tiempo de ejecución de de la función multMatriz y calculando la aceleración, se obtuvieron los siguientes resultados:

multStrassen vs multMatriz			
Tamaño	Tiempo de Ejecucion [ms]		Aceleración
Matriz [n*n]	multStrassen	multMatriz	
2	0.0373	0.1519	0.245556287
4	0.6651	0.0497	13.38229376
8	1.0353	0.1038	9.973988439
16	10.8616	0.6899	15.74373098
32	12.33	0.8025	15.36448598
64	94.6767	4.9639	19.0730474
128	706.8497	56.091	12.60183808
256	4902.8252	282.3377	17.36510994
512	35080.4636	2338.221	15.00305728
1024	243646.6289	20858.0451	11.68118238

Como se puede apreciar a simple vista, la función multMatriz presenta un tiempo de ejecución bastante bajo a comparación de la función multStrassen, lo cual también se ve reflejado en la aceleración obtenida, la cual es mayor a 10 en la mayoría de los casos. Por lo que, teniendo en cuenta que multStrassen es mas eficiente que multMatrizRec, multMatriz es el algoritmo secuencial más eficiente de todos.

Por último, se procederá a comparar los algoritmos paralelos entre sí.

Al comparar el tiempo de ejecución de la función multMatrizRecPar con el tiempo de ejecución de la función multStrassenPar y calculando la aceleración, se obtuvieron los siguientes resultados:

multMatrizRecPar vs multStrassenPar			
Tamaño	Tiempo de Ejecucion [ms]		Aceleración
Matriz [nxn]	multMatrizRecPar	multStrassenPar	
2	0.1136	0.1526	0.744429882
4	8.7493	0.2394	36.54678363
8	1.0661	0.7816	1.363996929
16	2.5509	4.0711	0.6265874088
32	14.0126	9.183	1.525928346
64	77.4579	47.8826	1.617662784
128	583.6371	347.4492	1.679776785
256	4868.6725	2443.7612	1.992286521
512	38971.3008	18802.8096	2.072631784
1024	311204.1869	120707.447	2.578168909

Como se puede observar, multStrassenPar presenta tiempos de ejecución menores en la mayoría de los casos, lo cual se puede ver reflejado en las aceleraciones obtenidas. Por lo que multStrassenPar resultó ser más eficiente.

Por otra parte, al comparar el tiempo de ejecución de la función multStrassenPar con el tiempo de ejecución de de la función multMatrizPar y calculando la aceleración, se obtuvieron los siguientes resultados:

multStrassenPar vs multMatrizPar			
Tamaño	Tiempo de Ejecucion [ms]		Aceleración
Matriz [n*n]	multStrassenPar	multMatrizPar	
2	0.1526	0.0311	4.906752412
4	0.2394	0.1391	1.721063983
8	0.7816	0.2949	2.650389963
16	4.0711	0.4601	8.848293849
32	9.183	0.9382	9.787891708
64	47.8826	4.8017	9.972009913
128	347.4492	40.7494	8.52648628
256	2443.7612	286.5665	8.527728119
512	18802.8096	2563.2955	7.335404599
1024	120707.447	25792.7674	4.679895148

Como se puede apreciar, la función multMatrizPar tiene tiempos de ejecución mucho menores a los tiempos de la función multStrassenPar, lo cual se puede observar claramente en las

aceleraciones obtenidas. Por lo que, teniendo en cuenta que multStrassenPar es más eficiente que multMatrizRecPar, la función multMatrizPar es la función paralela más eficiente de todas.

En base a todo lo anterior, se puede concluir que el algoritmo más eficiente de todos es multMatriz, en el cual no es recomendable paralelizar. Además, aunque se creía que el algoritmo multStrassen sería el más eficiente de todos, resultó estando por detrás de multMatriz tanto en su versión secuencial como en su versión paralela. Y por último, el algoritmo mulMatrizRec resultó ser el más ineficiente de todos tanto en su versión secuencial como en su versión paralela.
Por último, es importante mencionar que estos resultados están basados meramente en la manera en que las funciones fueron implementadas en este caso, por lo que no se descarta que existan mejores implementaciones que cambien por completo los resultados anteriores.