INTEGRANTES:
Nicolas Mauricio Rojas – 2259460
Juan Miguel Posso - 2259610
Esteban Revelo - 2067507

INFORME TALLER 4

1. Corrección de las funciones implementadas

1.1. Función Matriz al azar:

```
def matrizAlAzar(long: Int, vals: Int): Matriz
```

Esta función como su nombre nos indica, sirve para generar una matriz aleatoria y su funcionalidad consiste en recibir dos datos de tipo entero. La variable "long" será utilizada para generar una matriz cuadrada, mientras que la variable "vals" lo que hará es delimitar que rango de números tomara la matriz y dentro de las operaciones de la función se llenara la matriz con números aleatorios dentro del rango de 0 hasta vals.

1.2. Función Producto Punto:

```
def prodPunto(v1: Vector[Int], v2: Vector[Int]): Int
```

La función producto punto recibe dos vectores del mismo tamaño y retorna el producto punto entre ellos. La operación que realiza es la siguiente:

$$v1 \cdot v2 = v1_1 * v2_1 + v1_2 * v2_2 + \cdots + v1_n * v2_n$$

1.3. Función Transpuesta:

```
def transpuesta(m: Matriz): Matriz
```

La función transpuesta recibe una matriz y retorna una matriz en la que se cambia las filas por columnas de la matriz que recibe.

1.4. Función Multiplicar Matriz:

```
def multMatriz(m1: Matriz, m2: Matriz): Matriz
```

La función multiplicar matriz recibe dos matrices las cuales multiplicara entre sí, haciendo uso de las funciones anteriores, producto punto y transpuesta. Sabemos que la multiplicación de matrices funciona de la siguiente manera:

$$R_{IJ} = \sum_{k=1}^{n} m1_{ik} * m2_{kj}$$

Que en resumidas cuentas cada elemento de la matriz resultante “R” es el resultado de la suma de productos de los elementos correspondientes de la fila de la primera matriz “m1” y la columna de la segunda matriz “m2”. Entonces para lograr esto con las funciones que ya tenemos, lo que realizamos es sacar la matriz transpuesta de “m2” de modo que al utilizar la función producto punto enviamos “m1” y “m2” transpuesta, logrando que se multipliquen adecuadamente las filas de la matriz “m1” con las columnas de la matriz “m2”.

1.5. Función Multiplicar Matriz Paralelo:

```
def multMatrizParalelo(m1: Matriz, m2: Matriz): Matriz
```

Recibe dos matrices y retorna una matriz. Su funcionamiento consta de dividir la matriz “m1” en dos partes de modo que se crearan dos hilos para cada una de las mitades y para calcularlas se llamara a la función multiplicar matrices, en la que toma como parámetros una de las mitades de las matrices y la matriz “m2”. Luego de que se termine de hacer el calculo para cada una de las mitades se concatenan los resultados y se retorna la matriz resultante.

1.6. Función Sub Matriz :

```
def subMatriz(m: Matriz, i: Int, j: Int, l: Int): Matriz
```

Esta función recibirá una matriz de la cual se extraerá una sub matriz a partir de las coordenadas (i,j) y del tamaño deseado que es el parámetro “l”.

1.7. Función Suma Matriz:

```
def sumMatriz(m1: Matriz, m2: Matriz): Matriz
```

Esta función sumara las dos matrices que recibe y la operación que hace dentro de ella es sumar cada elemento de la matriz “m1” con la matriz “m2” la fórmula que define este comportamiento es la siguiente:

$$m1 + m2 = (m1_{ij} + m2_{ij})$$

1.8. Función Multiplicar Matriz Recursiva:

```
def multMatrizRec(m1: Matriz, m2: Matriz): Matriz
```

Esta función utiliza la técnica de dividir y conquistar. La función maneja matrices cuadradas. Si la dimensión de las matrices es 1, la función utiliza la multiplicación estándar de matrices que es la funcion (multMatriz). Si la dimensión es mayor a 1, la función divide las matrices en submatrices más pequeñas, realiza multiplicaciones recursivas y luego combina los resultados para formar la matriz resultante.

1.9. Función Multiplicar Matriz Recursiva Paralela:

```
def multMatrizrecPar(m1: Matriz, m2: Matriz): Matriz
```

Sus operaciones matemáticas son iguales a las de la función multiplicar matriz recursiva la única parte que cambia es que a la hora de realizar las multiplicaciones recursivas estas serán de manera paralela haciendo uso del paralelismo de tareas "task" y luego combina los resultados para formar la matriz resultante.

1.10. Función Resta Matriz:

```
def restaMatriz(m1: Matriz, m2: Matriz): Matriz
```

Funciona de la misma manera que la suma de matrices solo que restando la fórmula que representa esto es la siguiente:

$$m - m2 = (m1_{ij} - m2_{ij})$$

1.11. Función Producto Punto Paralelo:

```
def prodPuntoParD(v1: ParVector[Int], v2: ParVector[Int]): Int
```

En cuanto a la operación matemática que se realiza en esta función es igual a la de producto punto normal, lo único en lo que se diferencia es que los vectores son de tipo ParVector lo que permite paralelizar el cálculo de modo que se mejora el rendimiento en las operaciones que se realizan.

1.12. Función unir:

```
def unir(a: Matriz, b: Matriz, c: Matriz, d: Matriz): Matriz
```

Esta función fue implementada por nosotros para facilitar la concatenación de matrices y básicamente lo que realiza es combinar cuatro matrices bidimensionales en una sola matriz.

1.13. Función Multiplicacion Strassen:

```
def multMatrizParalelo(m1: Matriz, m2: Matriz): Matriz
```

La función realiza la multiplicación de dos matrices cuadradas m1 y m2 utilizando el algoritmo de Strassen, que es una técnica de dividir y conquistar. La función maneja matrices cuadradas de tamaño potencia de 2. Si la dimensión de las matrices es 1, la función realiza la multiplicación estándar de matrices. En caso contrario, la función divide las matrices en submatrices más pequeñas, aplica recursivamente el algoritmo de Strassen y luego combina los resultados para formar la matriz resultante.

1.14. Función Multiplicacion Strassen Paralela:

```
def multMatrizParalelo(m1: Matriz, m2: Matriz): Matriz
```

Sus operaciones matemáticas son exactamente iguales a las de la funcion de multiplicación de Strassen solo que esta vez los llamados recursivos se ejecutaran de manerea paralela.

2. Desempeño:

2.1. Evaluación comparativa:

Tamaño	multMatriz	multMatrizPa	Aceleracion
2	0,0706	0,0886	0,796839729
4	0,1221	0,0644	1,89596273
8	0,0948	0,1138	0,833040422
16	0,3632	0,1811	2,005521811
32	0,5177	0,4079	1,26918362
64	3,0191	1,7864	1,69004702
128	24,2118	12,6158	1,91916486
256	200,67	119,5624	1,67837046
512	1672,7479	1054,9739	1,58558226
1024	14550,6934	7665,7297	1,89814851

Tamaño	multMatrizRe	multMatrizRe	Aceleracion2
2	0,063	0,0936	0,67307692
4	0,1757	0,2046	0,85874878
8	1,08	0,795	1,35849057
16	1,0785	0,8299	1,29955416
32	9,7925	7,0959	1,38002227
64	80,3598	59,4762	1,35112532
128	677,7629	429,0726	1,57959958
256	5869,7214	3482,8713	1,6853110
512	43547,2566	28900,2042	1,50681484
1024	361742,493	246318,362	1,46859735

Tamaño	prodPunto	prodPuntoPa	Aceleracion3
2	0,0039	0,3907	0,00998208
4	0,0028	0,7236	0,00386954
8	0,0053	0,8198	0,0064650
16	0,0017	1,257	0,00135243
32	0,0019	0,9285	0,00204631
64	0,0035	1,4046	0,00249181
128	0,0039	1,2946	0,00301251
256	0,0057	1,0269	0,00555069
512	0,0087	1,1308	0,00769367

1024	0,0221	1,9701	0,0112177

Tamaño	multStrassen	multStrassen	Aceleracion4
2	0,00998208	0,1043	0,66059444
4	0,1209	0,151	0,80066225
8	0,5347	0,6726	0,79497472
16	0,9172	0,6801	1,3486252
32	7,0183	4,1533	1,68981292
64	47,4823	28,7454	1,65182255
128	334,9754	187,6878	1,78474786
256	2305,371	1463,8208	1,57489974
512	16733,9708	10020,8641	1,6699130
1024	120826,262	76824,68	1,57275321

- Preguntas:

¿Cuál de las implementaciones es más rápida?

La implementación paralela es la indiscutible ganadora a la hora de mostrar rendimiento, mostrando más rapidez que sus versiones secuenciales y sacándoles menos de la mitad del tiempo que les toma a algunas secuenciales

¿De qué depende que la aceleración sea mejor?

Depende del tipo de algoritmo que estamos implementando, también, podemos analizar que en algunos casos, los algoritmos funcionan mejor con matrices que tienen gran tamaño, aquí se empieza a notar la diferencia de tiempos, o aceleración con respecto a un algoritmo secuencial y paralelo

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

En el caso de la función prodPunto es mucho más efectivo usar la versión secuencial, pero en el caso de la multiplicación de matrices pudimos observar que la versión paralela es mucho mejor en las funciones de miltMatriz,mulMatrizRec,multStrassen.

2.2 Implementación del producto punto usando paralelismo de datos.

¿Será práctico construir versiones de los algoritmos de multiplicación de matrices del enunciado, para la colección ParVector y usar prodPuntoParD en lugar de prodPunto?

Podemos observar que no hubo mejoría en la implementación en el producto paralelo, un algoritmo ineficiente, lo que nos hace deducir que, si queremos mejorar los tiempos y ser más eficientes, no sería practico construir estas versiones tomando en cuenta este algoritmo.

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

Estos resultados fueron generados con las funciones tiempoAlgoritmosParalelos, tiempoAlgoritmosSecuenciales y CompararProdPunto.

Funciones Secuenciales:

Tamaño	multMatriz	multMatrizRec	multStrassen
2	0,0706	0,063	0,0099821
4	0,1221	0,1757	0,1209
8	0,0948	1,08	0,5347
16	0,3632	1,0785	0,9172
32	0,5177	9,7925	7,0183
64	3,0191	80,3598	47,4823
128	24,2118	677,7629	334,9754
256	200,67	5869,7214	2305,371
512	1672,7479	43547,257	16733,971
1024	14550,693	361742,49	120826,26

Funciones paralelas:

Tamaño	multMatrizParalelo	multMatrizRecPar	multStrassenPar
2	0,0886	0,0936	0,1043
4	0,0644	0,2046	0,151
8	0,1138	0,795	0,6726
16	0,1811	0,8299	0,6801
32	0,4079	7,0959	4,1533
64	1,7864	59,4762	28,7454
128	12,6158	429,0726	187,6878
256	119,5624	3482,8713	1463,8208
512	1054,9739	28900,204	10020,864
1024	7665,7297	246318,36	76824,68

Funciones de Vectores:

Tamaño	prodPunto	prodPuntoParD
2	0,0039	0,3907
4	0,0028	0,7236
8	0,0053	0,8198
16	0,0017	1,257
32	0,0019	0,9285
64	0,0035	1,4046
128	0,0039	1,2946
256	0,0057	1,0269
512	0,0087	1,1308
1024	0,0221	1,9701

3. Análisis comparativo:

3.1. Análisis comparativo de las diferentes soluciones

Al obtener los resultados en las tablas pudimos evidenciar que hay un comportamiento claro a la hora de ejecutar cada función y su versión paralela, en los datos pudimos observar que las versiones paralelas ganan en tiempo de ejecución, algunas iban el doble de rápidas que sus versiones secuenciales, en resumen, las funciones paralelas para trabajos de alto rendimiento son mucho más rápidas, pero esto siempre dependerá de cómo se construya la implementación de la paralelización.

¿Las paralizaciones sirvieron?

En nuestro caso sí sirvieron para la mayoría de funciones, a excepción de la función prodPunto, donde su versión secuencial siempre ganaba a la paralela, pero en las otras vimos un muy buen rendimiento de la paralelización.

¿Es realmente más eficiente el algoritmo de Strassen?

El algoritmo de Strassen fue mucho más efectivo que la función multMatrizRe, sacándole un resultado de más rápido que la de la mitad de tiempo que tardó la versión recursiva, pero por otro lado la multMatriz saco muchos mejores resultados que las otras dos funciones de multiplicación de matrices, con esto podemos decir que Strassen es un buen algoritmo que no se queda atrás en términos de rendimiento y eficiencia.

¿No se puede concluir nada al respecto?

Al final podemos concluir que las implementaciones de paralelización son mucho más efectivas que las funciones secuenciales, dándonos eficiencia y rendimiento a la hora de elaborar tareas de alto costo de rendimiento.