



**Universidad del Valle**  
**Facultad de ingeniería**  
**Ingeniería en sistemas**

Cristian David Pacheco Torres  
2227437

Juan Sebastian Molina Cuellar  
2224491

November 9, 2023

Taller 4: Colecciones y Expresiones For:  
El problema de la subsecuencia incremental de longitud máxima

# Contents

<b>1</b>	<b>Introducción</b>	<b>3</b>
1.1	Preliminares . . . . .	3
1.2	Algoritmos de proporcionados de utilidad . . . . .	4
<b>2</b>	<b>Informe del taller - secciones</b>	<b>5</b>
2.1	Informe de corrección . . . . .	5
2.1.1	Argumentación de Corrección . . . . .	10
2.1.2	Explicación Teórica y Método de Inducción . . . . .	10
2.1.3	Conclusión de Corrección . . . . .	10
2.2	Informe de desempeño de las funciones secuenciales y de las funciones paralelas . . . . .	10
2.2.1	Resultados de Multiplicación de Matrices . . . . .	10
2.2.2	Metodología de Generación de Matrices de Prueba . . . . .	12
2.2.3	Análisis de Resultados . . . . .	12
2.2.4	Resultados de Producto Punto de Vectores . . . . .	12
2.2.5	Impacto de las Dimensiones de los Vectores . . . . .	12
2.2.6	Análisis de Resultados del Producto Punto . . . . .	12
2.3	Análisis comparativo de las diferentes soluciones . . . . .	12
2.3.1	Análisis Basado en Resultados . . . . .	12
2.3.2	Eficiencia del Algoritmo de Strassen . . . . .	13
2.3.3	Reflexiones sobre el Paralelismo . . . . .	13
<b>3</b>	<b>Conclusiones</b>	<b>13</b>
3.1	Síntesis de Hallazgos . . . . .	13
3.2	Implicaciones de los Resultados . . . . .	13
3.3	Recomendaciones . . . . .	13

# 1 Introducción

## 1.1 Preliminares

Con el proposito de implementar diferentes algoritmos de multiplicación de matrices, tanto secuenciales, recursivos y paralelos, se nos otorgó a través del *Taller 5: Multiplicación de matrices en paralelo*, la definicion matemática de las siguientes operaciones:

- Transpuesta de una matriz.

$$\begin{aligned} T : \mathbb{R}^{m \times n} &\rightarrow \mathbb{R}^{n \times m} \\ T(A) &= [a_{ji}]_{n \times m} \end{aligned} \quad (1)$$

Se denota como :  $\mathbf{A}^T$

- Producto punto de vectores.

$$\begin{aligned} \cdot : \mathbb{R}^n \times \mathbb{R}^n &\rightarrow \mathbb{R} \\ \mathbf{u} \cdot \mathbf{v} &= \sum_{i=1}^n u_i v_i \end{aligned} \quad (2)$$

Donde "·" denota la operacion binaria entre dos vectores  $\in \mathbb{R}^n$ .

- Multiplicación de matrices.

$$C_{ij} = \sum_{k=1}^n A_{ik} \cdot B_{kj} \quad (3)$$

- Suma de matrices.

$$\begin{aligned} + : \mathbb{R}^{m \times n} \times \mathbb{R}^{m \times n} &\rightarrow \mathbb{R}^{m \times n} \\ A + B &= C \\ C_{ij} &= A_{ij} + B_{ij} \end{aligned} \quad (4)$$

Donde "+" representa la operación de adición entre dos matrices  $\in \mathbb{R}^{m \times n}$ . A y B son las matrices a sumar. C es la matriz resultante de la suma.

- Resta de matrices.

$$\begin{aligned} - : \mathbb{R}^{m \times n} \times \mathbb{R}^{m \times n} &\rightarrow \mathbb{R}^{m \times n} \\ C_{ij} &= A_{ij} - B_{ij} \\ A - B &= C \end{aligned} \quad (5)$$

### Definiciones de utilidad:

Sea una tarea computacional  $T = (t, r)$ , donde  $t$  es el tiempo de ejecución y  $r$  el resultado.

Además definase dos funciones sobre  $T$  como:

$$\rho(T) = \rho(t, r) = r \quad (6)$$

$$\phi(T) = \phi(t, r) = t \quad (7)$$

Definase una computación secuencial  $S$ :

$$S = \langle T_1, T_2, \dots, T_i, \dots, T_{n-1}, T_n \rangle \quad (8)$$

Donde  $i$  representa el orden de ejecución de la tarea  $T_i \mid 0 \leq i \leq n$ .

Definase una computación paralela  $P$ :

$$P = \{T_1, T_2, \dots, T_i, \dots, T_{n-1}, T_n\} \quad (9)$$

Donde  $i$  identifica cada tarea que a posteriori se le extraerá el tiempo de ejecución y el resultado.

Sea  $\phi(S)$  el tiempo de ejecución de una secuencia  $S$ :

$$\phi(S) = \sum_{i=1}^n \phi(T_i) \quad (10)$$

Sea  $\phi(P)$  el tiempo de ejecución del conjunto de tareas  $P$ :

$$\phi(P) = \max(\phi_i(T_i)) \mid 1 \leq i \leq n \wedge \phi_i(T_i) \in P \quad (11)$$

## 1.2 Algoritmos de proporcionados de utilidad

En el listing 1, se definen dos tipos de datos esenciales: `Matriz` y `MatrizD`. Estos tipos representan matrices de enteros, donde `MatrizD` está diseñada para un procesamiento paralelo en base al tipo `ParVector` de *Scala*.

```
1 type Matriz = Vector[Vector[Int]]
2 type MatrizD = ParVector[ParVector[Int]]
```

Listing 1: Definiciones tipos de datos

```
1 def matrizAlAzar(long:Int, vals:Int) = {
2     //Crea una matriz de enteros cuadrada de long x long,
3     //con valores entre 0 y vals
4     val v = Vector.fill(long, long){random.nextInt(vals)}
5     v
6 }
```

Listing 2: matriz al azar

```
1 def vectorAlAzar(long:Int, vals:Int): Vector[Int] = {
2     //Crea un vector de enteros de longitud long,
3     //con valores aleatorios entre 0 y vals
4     val v = Vector.fill(long){random.nextInt(vals)}
5     v
6 }
```

Listing 3: vector al azar

```

1 def prodPunto(v1: Vector[Int], v2: Vector[Int]): Int = {
2   //Calcula el producto punto entre dos vectores
3   (v1 zip v2).map({case (i,j) => i*j}).sum
4 }

```

Listing 4: producto punto

```

1 def transpuesta(m: Matriz): Matriz = {
2   //Calcula la transpuesta de una matriz
3   val l = m.length
4   Vector.tabulate(l,l)((i,j) => m(j)(i))
5 }

```

Listing 5: transpuesta de una matriz

Los algoritmos anteriores fueron sugeridos en el *Taller5* por parte del profesor, para la implementación de las funciones a desarrollar en este informe. Estos algoritmos fueron de utilidad para generar matrices aleatorias y operaciones fundamentales entre vectores.

## 2 Informe del taller - secciones

### 2.1 Informe de corrección

**multMatriz**

```

1 def multMatriz(m1: Matriz, m2: Matriz): Matriz = {
2   val l = m1.length
3   val m = m2.length
4   val v = Vector.tabulate(l,m)((i,j) => prodPunto(m1(i),
5     transpuesta(m2)(j)))
6   v
7 }

```

Listing 6: mult matriz

$$\begin{aligned}
 m1 * m2 &= \text{multMatriz}(m1, m2) \\
 &= \text{Vector.tabulate}(l, n)((i, j) \Rightarrow \text{prodPunto}(m1[i], \text{transpose}(m2)[j])) \\
 &= \begin{bmatrix} m1[0][0] \cdot \text{transpose}(m2)[0][0] + m1[0][1] \cdot \text{transpose}(m2)[1][0] + \\ \dots + m1[0][m-1] \cdot \text{transpose}(m2)[m-1][0] & \dots & m1[0][0] \cdot \\ \dots + m1[0][m-1] \cdot \text{transpose}(m2)[m-1][n-1] & & \vdots \\ m1[l-1][0] \cdot \text{transpose}(m2)[0][0] + m1[l-1][1] \cdot \text{transpose}(m2)[1][0] + \\ \dots + m1[l-1][m-1] \cdot \text{transpose}(m2)[m-1][0] & \dots & m1[l-1][0] \cdot \\ \dots + m1[l-1][m-1] \cdot \text{transpose}(m2)[m-1][n-1] & & \vdots \end{bmatrix}
 \end{aligned}$$

## multMatrizPar

```
1 def multMatrizPar(m1: Matriz, m2: Matriz): Matriz = {
2   val l = m1.length
3   val m = m2.length
4   val parRows = for (k <- 0 until l )
5     yield {
6       (k, task(Vector.tabulate(1,m)((i,j)=>
7         prodPunto(m1(k), transpuesta(m2)(j)))))
8     }
9   val v = parRows.map({case (i,j) =>
10     (i,j.join())}).sortBy(_._1).map(_._2)
11   v.reduce(_+_ )
12 }
```

Listing 7: mult matriz paralela

## multMatrizRec

```
1 def multMatrizRec(m1:Matriz, m2:Matriz): Matriz ={
2   //recibe m1 y m2 matrices cuadradas de la misma dimension,
3   //potencia de 2
4   //y devuelve la multiplicacion de las 2 matrices
5   val n = m1.length
6   if(n == 1) {
7     Vector(Vector(m1(0)(0)*m2(0)(0)))
8   }
9   else {
10    val l = n/2
11    val (m1_11, m1_12, m1_21, m1_22) =
12      (subMatriz(m1,0,0,l),subMatriz(m1,0,l,l),
13       subMatriz(m1,l,0,l),subMatriz(m1,l,l,l))
14    val (m2_11, m2_12, m2_21, m2_22) =
15      (subMatriz(m2,0,0,l),subMatriz(m2,0,l,l),
16       subMatriz(m2,l,0,l),subMatriz(m2,l,l,l))
17
18    val c_11 = sumMatriz(multMatrizRec(m1_11,m2_11),
19      multMatrizRec(m1_12,m2_21))
20    val c_12 = sumMatriz(multMatrizRec(m1_11,m2_12),
21      multMatrizRec(m1_12,m2_22))
22    val c_21 = sumMatriz(multMatrizRec(m1_21,m2_11),
23      multMatrizRec(m1_22,m2_21))
24    val c_22 = sumMatriz(multMatrizRec(m1_21,m2_12),
25      multMatrizRec(m1_22,m2_22))
26
27    Vector.tabulate(n,n)((i,j)=>
28      if(i<l && j<l) c_11(i)(j)
29      else if(i<l && j>=l) c_12(i)(j-l)
30      else if(i>=l && j<l) c_21(i-l)(j)
31      else c_22(i-l)(j-l))
32  }
```

```

26         else c_22(i-1)(j-1))
27     }
28 }

```

Listing 8: mult matriz recursiva

### multMatrizRecPar

```

1 def multMatrizRecPar(m1:Matriz, m2:Matriz): Matriz ={
2     //recibe m1 y m2 matrices cuadradas de la misma dimension,
3     //y devuelve la multiplicacion de las 2 matrices,
4     //paralelizando tareas
5     val n = m1.length
6     if(n == 1) {
7         Vector(Vector(m1(0)(0)*m2(0)(0)))
8     }
9     else {
10        val l = n/2
11        val (m1_11, m1_12, m1_21, m1_22) =
12            (subMatriz(m1,0,0,l),subMatriz(m1,0,l,l),
13             subMatriz(m1,l,0,l),subMatriz(m1,l,l,l))
14        val (m2_11, m2_12, m2_21, m2_22) =
15            (subMatriz(m2,0,0,l),subMatriz(m2,0,l,l),
16             subMatriz(m2,l,0,l),subMatriz(m2,l,l,l))
17        val (c_11, c_12, c_21, c_22) = parallel(
18            sumMatriz(multMatrizRec(m1_11,m2_11),
19                      multMatrizRec(m1_12,m2_21)),
20            sumMatriz(multMatrizRec(m1_11,m2_12),
21                      multMatrizRec(m1_12,m2_22)),
22            sumMatriz(multMatrizRec(m1_21,m2_11),
23                      multMatrizRec(m1_22,m2_21)),
24            sumMatriz(multMatrizRec(m1_21,m2_12),
25                      multMatrizRec(m1_22,m2_22)))
26        Vector.tabulate(n,n)((i,j)=>
27            if(i<l && j<l) c_11(i)(j)
28            else if(i<l && j>=l) c_12(i)(j-l)
29            else if(i>=l && j<l) c_21(i-l)(j)
30            else c_22(i-l)(j-l))
31    }
32 }

```

Listing 9: mult matriz recursiva paralela

### multStrassen

```

1 def multStrassen(m1:Matriz, m2:Matriz): Matriz ={
2     //recibe m1 y m2 matrices cuadradas de la misma dimension,
3     //y devuelve la multiplicacion de las 2 matrices

```

```

4    val n = m1.length
5    if(n == 1) {
6        Vector(Vector(m1(0)(0)*m2(0)(0)))
7    }
8    else {
9        val l = n/2
10       val (s1, s2, s3, s4, s5, s6, s7, s8, s9, s10) = (
11           restaMatriz(subMatriz(m2,0,l,l),subMatriz(m1,l,l,l)),
12           sumMatriz(subMatriz(m1,0,0,l), subMatriz(m1,0,l,l)),
13           sumMatriz(subMatriz(m1,l,0,l), subMatriz(m1,l,l,l)),
14           restaMatriz(subMatriz(m2,l,0,l), subMatriz(m2,0,0,l))
15           ,
16           sumMatriz(subMatriz(m1,0,0,l), subMatriz(m1,l,l,l)),
17           sumMatriz(subMatriz(m2,0,0,l), subMatriz(m2,l,l,l)),
18           restaMatriz(subMatriz(m1,0,l,l), subMatriz(m1,l,l,l))
19           ,
20           sumMatriz(subMatriz(m2,l,0,l), subMatriz(m2,l,l,l)),
21           restaMatriz(subMatriz(m1,0,0,l), subMatriz(m1,l,0,l))
22           ,
23           sumMatriz(subMatriz(m2,0,0,l), subMatriz(m2,0,l,l))
24       )
25       val (p1, p2, p3, p4, p5, p6, p7) = (
26           multStrassen(subMatriz(m1,0,0,l), s1),
27           multStrassen(s2, subMatriz(m2,l,l,l)),
28           multStrassen(s3, subMatriz(m2,0,0,l)),
29           multStrassen(subMatriz(m1,l,l,l), s4),
30           multStrassen(s5, s6),
31           multStrassen(s7, s8),
32           multStrassen(s9, s10)
33       )
34       val (c_11, c_12, c_21, c_22) = (
35           restaMatriz(sumMatriz(p5, p4), sumMatriz(p6, p2)),
36           sumMatriz(p1, p2),
37           sumMatriz(p3, p4),
38           restaMatriz(sumMatriz(p1, p5), restaMatriz(p3, p7))
39       )
40       Vector.tabulate(n,n)((i,j)=>
41           if(i<l && j<l) c_11(i)(j)
42           else if(i<l && j>=l) c_12(i)(j-l)
43           else if(i>=l && j<l) c_21(i-l)(j)
44           else c_22(i-l)(j-l))
45     }
46 }

```

Listing 10: mult Strassen

## multStrassenPar

```

1    def multStrassenPar(m1:Matriz, m2:Matriz): Matriz = {
2        //recibe m1 y m2 matrices cuadradas de la misma dimension
3        , potencia de 2

```



```

3 //y devuelve la multiplicacion de las 2 matrices
4 val n = m1.length
5
6 /*if(umbral <= n){
7     multMatrizRec(m1,m2)
8 }*/
9 if(n == 1) {
10     Vector(Vector(m1(0)(0)*m2(0)(0)))
11 }
12 else {
13     val l = n/2
14     val (s1, s2, s3, s4, s5, s6, s7, s8, s9, s10) = (
15         restaMatriz(subMatriz(m2,0,l,l),
16             subMatriz(m1,l,l,l)),
17         sumMatriz(subMatriz(m1,0,0,l),
18             subMatriz(m1,0,l,l)),
19         sumMatriz(subMatriz(m1,l,0,l),
20             subMatriz(m1,l,l,l)),
21         restaMatriz(subMatriz(m2,l,0,l),
22             subMatriz(m2,0,0,l)),
23         sumMatriz(subMatriz(m1,0,0,l),
24             subMatriz(m1,l,l,l)),
25         sumMatriz(subMatriz(m2,0,0,l),
26             subMatriz(m2,l,l,l)),
27         restaMatriz(subMatriz(m1,0,l,l),
28             subMatriz(m1,l,l,l)),
29         sumMatriz(subMatriz(m2,l,0,l),
30             subMatriz(m2,l,l,l)),
31         restaMatriz(subMatriz(m1,0,0,l),
32             subMatriz(m1,l,0,l)),
33         sumMatriz(subMatriz(m2,0,0,l),
34             subMatriz(m2,0,l,l))
35     )
36     val (p1, p2, p3, p4, p5, p6, p7) = (
37         task(multStrassenPar(subMatriz(m1,0,0,l), s1)),
38         task(multStrassenPar(s2, subMatriz(m2,l,l,l))),
39         task(multStrassenPar(s3, subMatriz(m2,0,0,l))),
40         task(multStrassenPar(subMatriz(m1,l,l,l), s4)),
41         task(multStrassenPar(s5, s6)),
42         task(multStrassenPar(s7, s8)),
43         task(multStrassenPar(s9, s10))
44     )
45     val (c_11, c_12, c_21, c_22) = (
46         restaMatriz(sumMatriz(p5.join(), p4.join()),
47             sumMatriz(p6.join(), p2.join())),
48         sumMatriz(p1.join(), p2.join()),
49         sumMatriz(p3.join(), p4.join()),
50         restaMatriz(sumMatriz(p1.join(), p5.join()),
51             restaMatriz(p3.join(), p7.join()))
52     )
53     Vector.tabulate(n,n)((i,j)=>

```

```

52         if(i<1 && j<1) c_11(i)(j)
53     else if(i<1 && j>=1) c_12(i)(j-1)
54     else if(i>=1 && j<1) c_21(i-1)(j)
55     else c_22(i-1)(j-1)
56     }
57 }

```

Listing 11: mult Strassen paralela

### 2.1.1 Argumentación de Corrección

Argumentación detallada de la corrección para cada uno de los algoritmos de multiplicación de matrices y productos punto.

### 2.1.2 Explicación Teórica y Método de Inducción

Explicación teórica de la corrección y método de inducción o inducción estructural utilizado.

### 2.1.3 Conclusión de Corrección

Conclusión sobre la corrección de cada función.

## 2.2 Informe de desempeño de las funciones secuenciales y de las funciones paralelas

### 2.2.1 Resultados de Multiplicación de Matrices

Tabla comparativa y análisis de desempeño.

Matrix size	mulMatriz(ms)	multMatrizPar(ms)	speedup
$2^0$	0.0208	0.0126	1.65
$2^1$	0.054	0.0899	0.60
$2^2$	0.1658	0.2217	0.75
$2^3$	0.6989	1.003	0.70
$2^4$	2.2913	4.7724	0.48
$2^5$	2.8232	5.416	0.52
$2^6$	22.1308	36.1349	0.61
$2^7$	172.5227	249.1038	0.69
$2^8$	1391.6189	1747.6074	0.80
$2^9$	11624.2295	13095.9631	0.88
$2^{10}$	92733.1786	92184.1319	1.005

Table 1: mulMatriz vs multMatrizPar

La tabla proporcionada muestra los tiempos de ejecución de dos implementaciones de multiplicación de matrices, una secuencial (**mulMatriz**) y otra paralela (**multMatrizPar**), para diferentes tamaños de matrices, expresados como potencias de 2. Además, se presenta el *speedup* de la versión paralela en comparación con la secuencial. A continuación, se analizan diversos aspectos de estas implementaciones:

1. **¿Cuál de las implementaciones es más rápida?**

Para matrices de tamaño pequeño (hasta  $2^6$ ), la versión secuencial (`mulMatriz`) muestra un mejor rendimiento. Sin embargo, a partir de matrices de tamaño  $2^7$ , la implementación paralela (`multMatrizPar`) empieza a ser más eficiente, superando ligeramente a la versión secuencial en tamaños de  $2^{10}$ .

2. **¿De qué depende que la aceleración sea mejor?**

La aceleración se ve influenciada principalmente por el tamaño de la matriz. Para tamaños pequeños, el *overhead* de la paralelización puede no compensar sus beneficios. A medida que el tamaño aumenta, los beneficios del paralelismo superan este *overhead*, resultando en un mejor *speedup*.

3. **¿En qué casos es mejor usar la versión secuencial o paralela de cada algoritmo?**

- **Versión Secuencial (`mulMatriz`):** Ideal para matrices pequeñas (hasta  $2^6$ ), donde el *overhead* del paralelismo no se justifica.
- **Versión Paralela (`multMatrizPar`):** Recomendada para matrices de gran tamaño (a partir de  $2^7$ ), donde el coste del paralelismo se ve compensado por la eficiencia en el procesamiento de grandes volúmenes de datos.

Matrix size	mulMatrizRecPar(ms)	multStrassen(ms)	speedup
$n^0$	0.0882	0.0248	3.56
$n^1$	0.2109	0.1308	1.61
$n^2$	0.3769	0.4245	0.89
$n^3$	0.9086	1.5833	0.57
$n^4$	0.8598	4.6825	0.18
$n^5$	17.6741	30.5887	0.58
$n^6$	51.2801	232.0896	0.22
$n^7$	389.73	1692.9611	0.23
$n^8$	3114.4097	5199.9472	0.60

Table 2: mulMatrizRecPar vs multStrassen

Matrix size	mulMatrizRec(ms)	multStrassenPar(ms)	speedup
$n^0$	0.0541	0.035	1.55
$n^1$	0.033	0.2113	0.16
$n^2$	0.1192	0.33	0.36
$n^3$	0.5392	2.0168	0.27
$n^4$	1.1699	5.0879	0.23
$n^5$	8.4393	11.5936	0.73
$n^6$	67.8253	64.9989	1.04
$n^7$	567.7783	451.8261	1.26

Table 3: mulMatrizRec vs multStrassenPar

Matrix size	multMatrizRec(ms)	multStrassen(ms)	Speedup
$n^0$	0.0195	0.0129	1.51
$n^1$	0.0584	0.0403	1.45
$n^2$	0.0926	0.3343	0.28
$n^3$	0.6009	0.9823	0.61
$n^4$	1.2007	2.0959	0.57
$n^5$	5.8948	10.1365	0.58
$n^6$	49.869	70.9033	0.70
$n^7$	415.2096	529.4741	0.78
$n^8$	3099.9929	3523.037	0.88
$n^9$	25379.6906	25235.8456	1.01

Table 4: multMatrizRec vs multStrassen

Matrix size	multMatrizRec(ms)	multMatrizRecPar(ms)	Speedup
$n^0$	0.0535	0.0495	1.08
$n^1$	0.0632	0.1054	0.60
$n^2$	0.1286	0.1518	0.85
$n^3$	0.403	0.4911	0.82
$n^4$	5.5323	3.9688	1.39
$n^5$	8.5952	4.7139	1.82
$n^6$	67.7342	37.6179	1.80
$n^7$	551.341	300.7862	1.83
$n^8$	4441.5969	2399.4469	1.85

Table 5: multMatrizRec vs multMatrizRecPar

### 2.2.2 Metodología de Generación de Matrices de Prueba

Descripción de cómo se generaron las matrices de prueba.

### 2.2.3 Análisis de Resultados

Análisis profundo de los resultados obtenidos.

### 2.2.4 Resultados de Producto Punto de Vectores

Tabla comparativa y análisis de desempeño para las implementaciones de producto punto.

### 2.2.5 Impacto de las Dimensiones de los Vectores

Discusión sobre cómo las dimensiones de los vectores afectan al desempeño.

### 2.2.6 Análisis de Resultados del Producto Punto

Análisis detallado de los resultados del producto punto.

## 2.3 Análisis comparativo de las diferentes soluciones

### 2.3.1 Análisis Basado en Resultados

Comparación entre las versiones secuenciales y paralelas de los algoritmos y su desempeño.

Matrix size	multMatriz(ms)	multMatrizPar(ms)	Speedup
$n^0$	0.157074	0.508098	0.31
$n^1$	0.058667	0.259741	0.23
$n^2$	0.06202	0.304369	0.20
$n^3$	0.241862	0.186058	1.30
$n^4$	1.196248	0.536174	2.23
$n^5$	15.896863	2.565981	6.20
$n^6$	234.5659	21.393817	10.96
$n^7$	3373.980202	374.861453	9.00
$n^8$	52778.116937	10691.328672	4.94

Table 6: multMatriz vs multMatrizPar

Matrix size	multMatriz(ms)	multMatrizRec(ms)	Speedup
$n^0$	0.197233	0.124877	1.58
$n^1$	0.098057	0.02626	3.73
$n^2$	0.072356	0.128369	0.56
$n^3$	0.128857	0.153372	0.84
$n^4$	1.26574	0.863941	1.47
$n^5$	15.580685	6.217719	2.51
$n^6$	233.389835	57.153132	4.08
$n^7$	3756.499596	466.273799	8.06
$n^8$	74007.560303	3649.418623	20.28

Table 7: multMatriz vs multMatrizRec

### 2.3.2 Eficiencia del Algoritmo de Strassen

Discusión específica sobre la eficiencia del algoritmo de Strassen en comparación con otros.

### 2.3.3 Reflexiones sobre el Paralelismo

Evaluación crítica del paralelismo de tareas y de datos y su efecto en la eficiencia general.

## 3 Conclusiones

### 3.1 Síntesis de Hallazgos

Resumen de los hallazgos más importantes del informe.

### 3.2 Implicaciones de los Resultados

Discusión sobre las implicaciones de los resultados para futuras investigaciones y aplicaciones prácticas.

### 3.3 Recomendaciones

Recomendaciones basadas en el análisis y desempeño de las funciones estudiadas.

Matrix size	multMatriz(ms)	multMatrizRecPar(ms)	Speedup
$n^0$	0.213017	0.054686	3.90
$n^1$	0.05727	0.261487	0.22
$n^2$	0.093867	0.193461	0.49
$n^3$	0.138426	0.931548	0.15
$n^4$	1.035612	0.553565	1.87
$n^5$	15.601226	4.07065	3.83
$n^6$	260.689531	30.724248	8.48
$n^7$	4699.357314	223.874599	20.99
$n^8$	94123.683973	1796.040497	52.41

Table 8: multMatriz vs multMatrizRecPar

Matrix size	multMatriz(ms)	multStrassenPar(ms)	Speedup
$n^0$	0.0271	0.0113	2.40
$n^1$	0.05	0.0886	0.56
$n^2$	0.0298	0.1949	0.15
$n^3$	0.1785	0.687	0.26
$n^4$	0.9352	2.2692	0.41
$n^5$	10.9968	4.8675	2.26
$n^6$	199.8272	38.1879	5.23
$n^7$	3721.3193	240.8355	15.45
$n^8$	77156.0959	1724.9942	44.73

Table 9: multMatriz vs multStrassenPar

Matrix size	multStrassen(ms)	multMatrizPar(ms)	Speedup
$n^0$	0.0187	0.0708	0.26
$n^1$	0.0472	0.0979	0.48
$n^2$	0.1763	0.1098	1.61
$n^3$	1.4824	0.1596	9.29
$n^4$	1.3893	0.2277	6.10
$n^5$	10.0845	2.1428	4.71
$n^6$	71.4526	39.3428	1.82
$n^7$	524.8294	660.7993	0.79
$n^8$	3715.7102	13099.9935	0.28

Table 10: multStrassen vs multMatrizPar

Matrix size	multStrassen(ms)	multStrassenPar(ms)	Speedup
$n^0$	0.094566	0.018089	5.23
$n^1$	0.164617	0.266935	0.62
$n^2$	0.082203	0.486237	0.17
$n^3$	0.551889	0.492383	1.12
$n^4$	2.002918	2.132335	0.94
$n^5$	11.121369	9.672645	1.15
$n^6$	83.202037	56.999489	1.46
$n^7$	591.982072	397.681511	1.49
$n^8$	4214.366236	2770.456839	1.52

Table 11: multStrassen vs multStrassenPar

Matrix size	multMatrizRec(ms)	multMatrizPar(ms)	Speedup
$n^0$	0.069562	0.198001	0.35
$n^1$	0.046654	0.318967	0.15
$n^2$	0.501463	0.312541	1.60
$n^3$	0.171252	0.516339	0.33
$n^4$	0.914017	0.475552	1.92
$n^5$	6.9931	2.383554	2.93
$n^6$	52.731112	44.138359	1.19
$n^7$	459.95615	668.513945	0.69
$n^8$	3509.126971	13464.049988	0.26

Table 12: multMatrizRec vs multMatrizPar

Matrix size	multStrassenPar(ms)	multMatrizPar(ms)	Speedup
$n^0$	0.0183	0.0677	0.27
$n^1$	0.068	0.0845	0.80
$n^2$	0.1879	0.1407	1.34
$n^3$	0.6789	0.2121	3.20
$n^4$	2.3105	0.4184	5.52
$n^5$	5.0187	2.1041	2.39
$n^6$	35.2597	41.5603	0.85
$n^7$	254.7461	671.1146	0.38
$n^8$	1759.3622	13238.1516	0.13

Table 13: multStrassenPar vs multMatrizPar

Matrix size	multMatrizRecPar(ms)	multMatrizPar(ms)	Speedup
$n^0$	0.0702	0.0311	2.26
$n^1$	0.0424	0.1279	0.33
$n^2$	0.0463	0.1546	0.30
$n^3$	0.2225	0.7962	0.28
$n^4$	1.8466	2.8966	0.64
$n^5$	19.7595	14.4537	1.37
$n^6$	325.1826	104.5296	3.11
$n^7$	5908.247	727.7273	8.12
$n^8$	114646.5989	5121.9016	22.38

Table 14: multMatrizRecPar vs multMatrizPar