CLASE 2-2:
Programación Paralela
Una Introducción

Instructor Rina Surós

29 de Septiembre 2022

Facultad de Ingeniería. UDELAR

Montevideo

Paradigmas de Programación Paralela
Continuación

Los tiempos de ejecución

Cuando ejecutamos el código para el cálculo de potencias, nos preguntábamos como saber si el código paralelo había acelerado nuestros procesos.

Para saber si el código paralelo es más rápido que el código en secuencial, debemos hacer un estudio comparativo de los tiempos de ejecución. Asumamos que:

$\mathbf{T_S}$ = Tiempo de ejecución secuencial, es el tiempo total que tarda el código secuencial en ser ejecutado. Obviamente utilizando **un solo nodo** del computador.

- $\mathbf{T_1}$ es el tiempo que registra el computador al inicio del proceso
- $\mathbf{T_f}$ es el tiempo que registra el computador al final del proceso

Entonces

$$\mathbf{T_S = T_f - T_1}$$

La función Sys.time() permite registrar el tiempo utilizando el reloj del computador

```
# Al inicio de la ejecución, almaceno este valor
t1 = Sys.time() # tiempo inicial del sistema

# Al final de la ejecución, almaceno este valor
t11 = Sys.time() # tiempo inicial del sistema

Tt = t1 - t11
```

El resultado es el tiempo total de ejeción del código secuencial

Cálculo de Normas

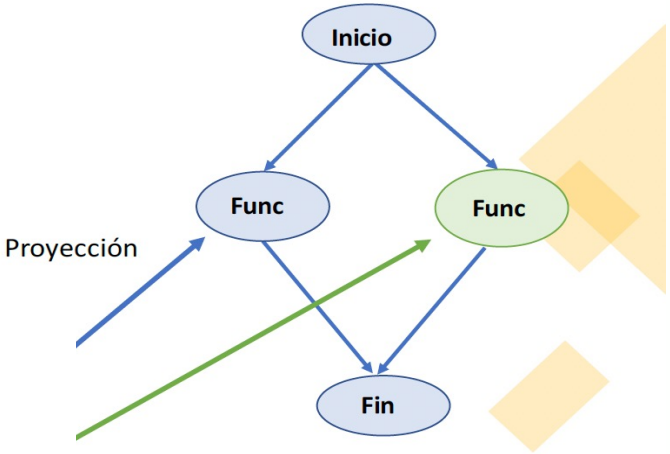
En la primera clase vimos un ejemplo de código paralelo, el **scripts** en R

2 Normas 2 Nodos.r

En este código obtenemos valores aleatorios utilizando **rnorm(n, μ , σ)** que genera n valores aleatorios, con distribución normal de media μ y y desviación estándar **σ.**

- Generamos dos conjuntos distintos de valores aleatorios.
- Cada conjunto de datos es independiente del otro, entonces pueden ser ejecutados en dos nodos distintos. Necesitamos solo 2 nodos.

```
# rnorm(z,a,b) produce z valores aleatorios normales(a,b)

######## Secuencial #######
# Parámetros para la normal
a = 2
b = 7

t1 = Sys.time() # tiempo inicial del sistema
y1 = rnorm(55000000,a,b)
t11 = Sys.time()  # tiempo final del sistema para la primera norma

y2 = rnorm(60000000,a,b)
t12 = Sys.time() # tiempo final del sistema para la segunda norma

t2 = Sys.time() # tiempo final del sistema
tt = t2-t1

res_secuencial = c(secuencial=tt,parte_1=t11-t1,parte_2=t12-t11) #

res_secuencial
```



Dos procesos independientes que se pueden ejecutar en dos procesadores distintos.

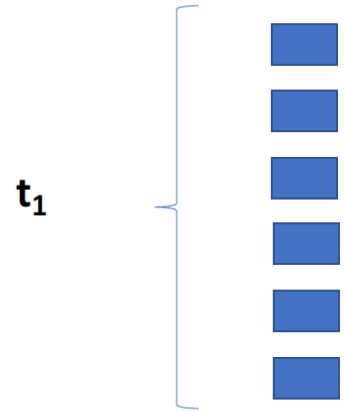
Paradigma Maestro-Esclavo

Diagrama de los tiempos de ejecución en el Paradigma Maestro-Esclavo

Caso secuencial: un único nodo

t_1

Para calcularlo medimos el tiempo de ejecución del programa

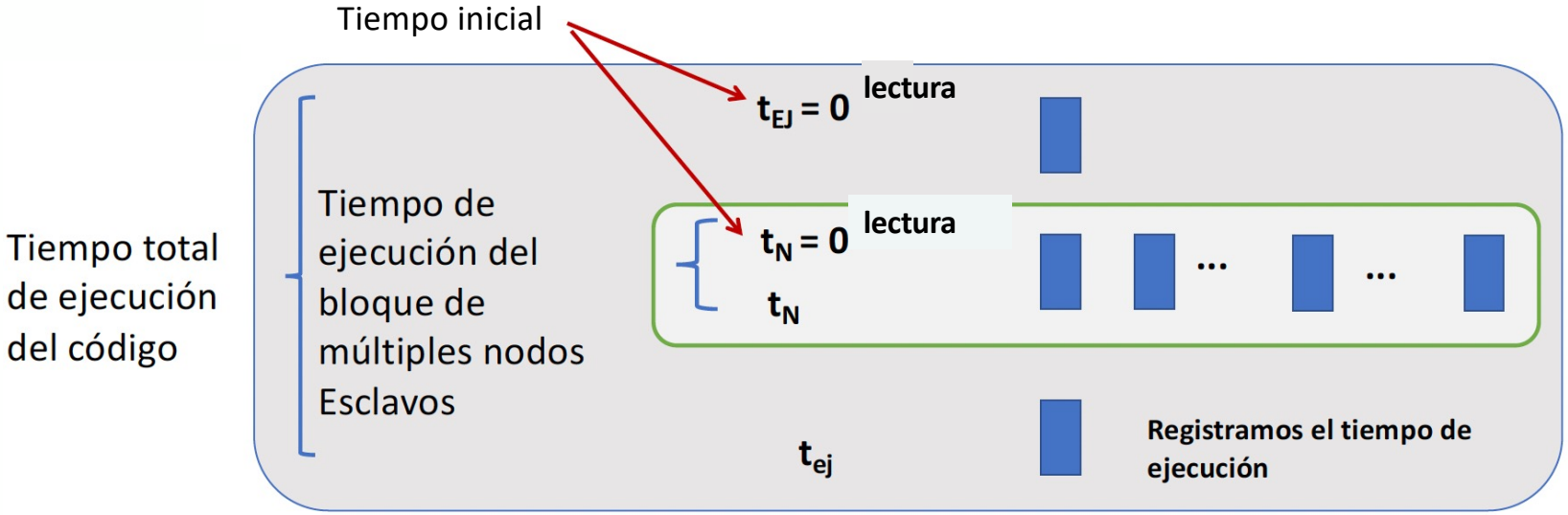


t_1 = 0

t_1 = m unidades de tiempo

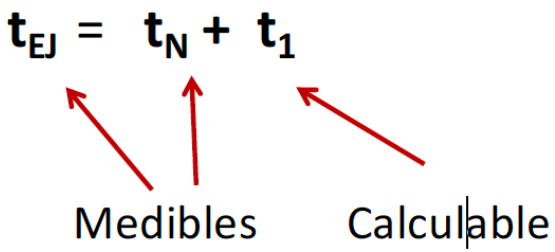
```
########## Paralelo ##########
# El mismo código en paralelo con 2 nodos

no.estandares = function(x,a,b){

 m1 = 0
 m2 = 0

 if(x==1) {
  m1 = Sys.time()
  y = rnorm(55000000,a,b)
  m2 = Sys.time()
 }

 if(x==2) {
  m1 = Sys.time()
  y = rnorm(60000000,a,b)
  m2 = Sys.time()

 }

 d = as.numeric(difftime(m2,m1), units="secs") #
Calcula m1-m2
  z = list()
  z[[1]] = y
  z[[2]] = m1
  z[[3]] = m2
  z[[4]] = d

  return(z)
}

library(parallel)
cl <- makeCluster(2)
l1 = Sys.time()
res = clusterApply(cl, x = 1:2, fun =
no.estandares,2,7)
l2 = Sys.time()
stopCluster(cl)

res_paralelo = c(secuencial=tt,paralelo_total =
difftime(l2,l1), paralelo_1 = res[[1]][[4]],
                 paralelo_2= res[[2]][[4]] )

res_secuencial
res_paralelo
```

Tiempo secuencial y Tiempo Paralelo



Caso paralelo. Medimos por separado los siguientes tiempos

- t_{EJ} = tiempo de ejecución total del código: Maestro + Esclavos
- t_N = **t**iempo de ejecución de la parte que se ejecuta con N nodos Esclavos
- Esta medida de tiempo la hace el Maestro antes de enviar


$t_{EJ} = t_N + t_1$
Medibles
Calculable

Resultados

Veamos como se ejecuta eo código en r

res_secuencial

Time differences in secs

secuencial parte_1 parte_2

5.948483 2.719291 3.229029

> res_paralelo

Time differences in secs

secuencial paralelo_total paralelo_1 paralelo_2

5.948483 5.985737 2.826460 3.072287

El tiempo de ejecución paralela es aproximadamente la mitad del tiempo de ejecución secuencial.

Conclusión, hemos logrado nuestro objetivo

¿Todo el código es paralelizable?

La respuesta es **NO**

Todo código tiene unas secciones paralelizables y otras que no lo son. Regresemos al cálculo de las normas. Analicemos el código secuencial.

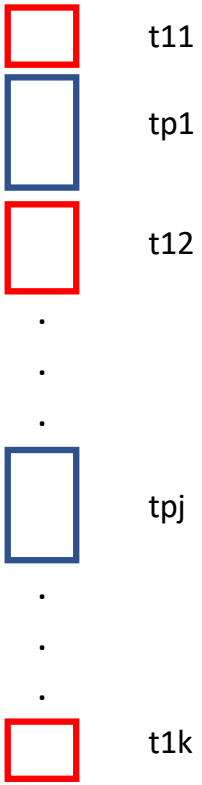
```
# rnorm(z,a,b) produce z valores aleatorios normales(a,b)

######## Secuencial #######
# Parámetros para la normal
a = 2
b = 7

t1 = Sys.time() # tiempo inicial del sistema
y1 = rnorm(55000000,a,b)
t11 = Sys.time()  # tiempo final del sistema para la primera norma

y2 = rnorm(60000000,a,b)
t12 = Sys.time() # tiempo final del sistema para la segunda norma

t2 = Sys.time() # tiempo final del sistema
tt = t2-t1

res_secuencial = c(secuencial=tt,parte_1=t11-t1,parte_2=t12-t11) #

res_secuencial
```

Sección **NO** paralelizable

Sección paralelizable

Sección **NO** paralelizable

Tiempos de Ejecución de un Código Paralelo



El tiempo deejecución de un código paralelo se compone de etapas secuenciales (no paralelizables) y etapas paralelas

El tiempo total de ejecuciónE es la suma de los tiempos de ejecución de todas las etapas ecuenciales (T1j) + los tiempos de ejecución de todas las etapas potencialmente paralelas Tni

$$Tni = \Sigma_j T1j + \Sigma_i Tni$$

Entonces, es tarea del programador debe registrar cada uno de esos tiempo para analizar el rendimiento de su código.

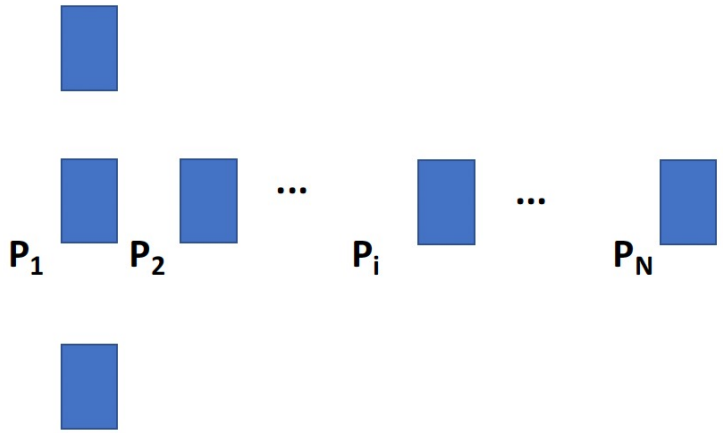
Avancemos con las definiciones.

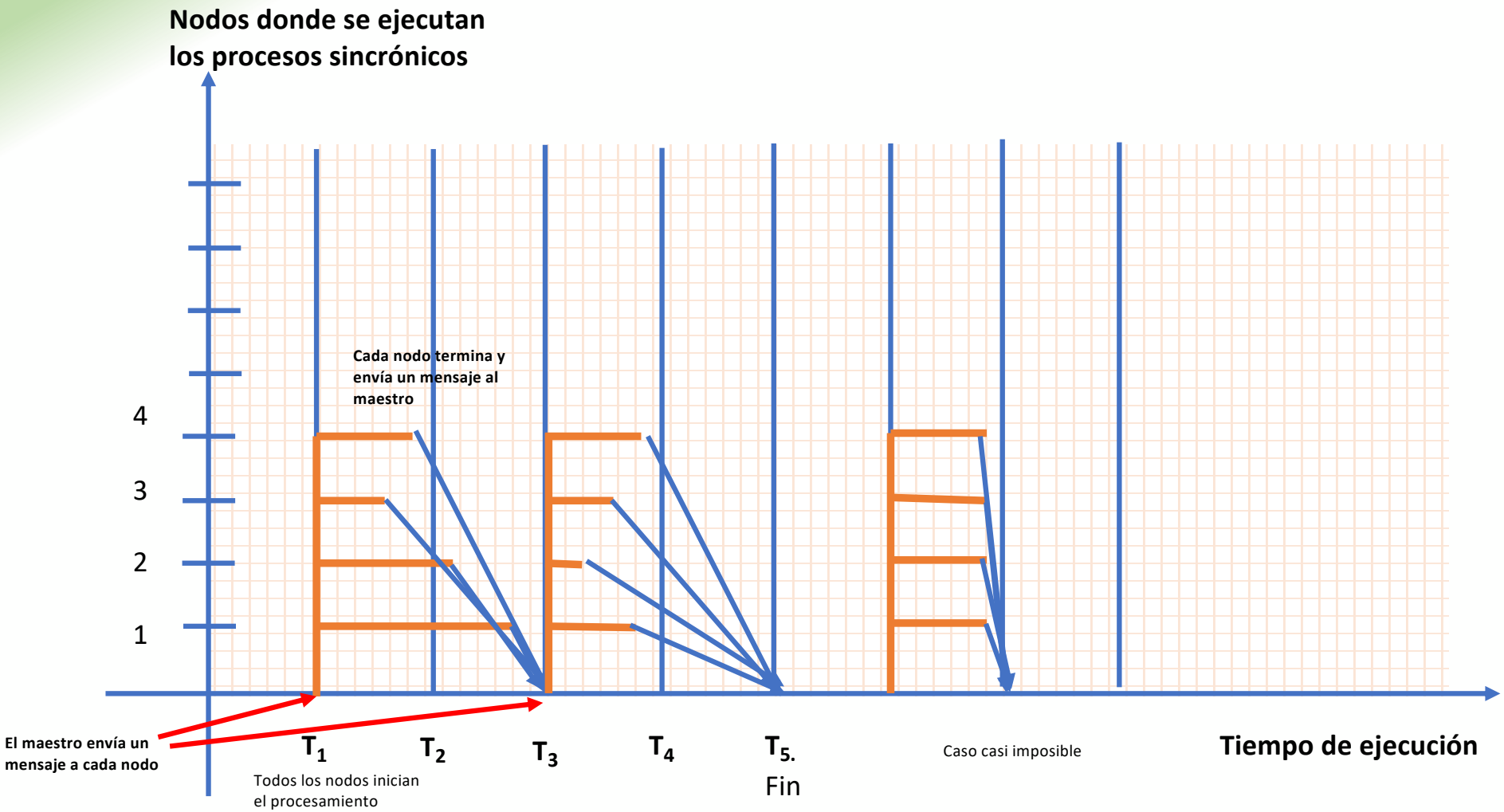
Paradigma Maestro-Esclavo

Es posible medir el tiempo de ejecución de los procesos que ejecutan los esclavos ubicados en nodos distintos: $\mathbf{t_{Pi}}$

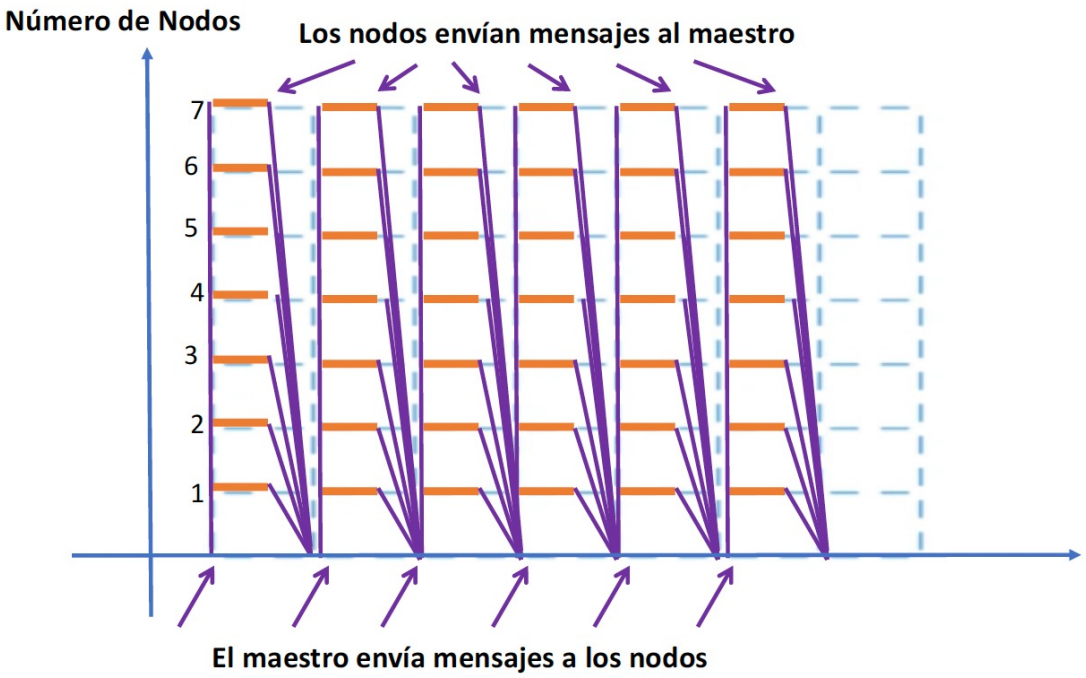
Los contadores de tiempo se ubican deltro del código que ejecuta cada esclavo

Cada t_{Pi} tarda K unidades de tiempo en ejecutar su parte del código



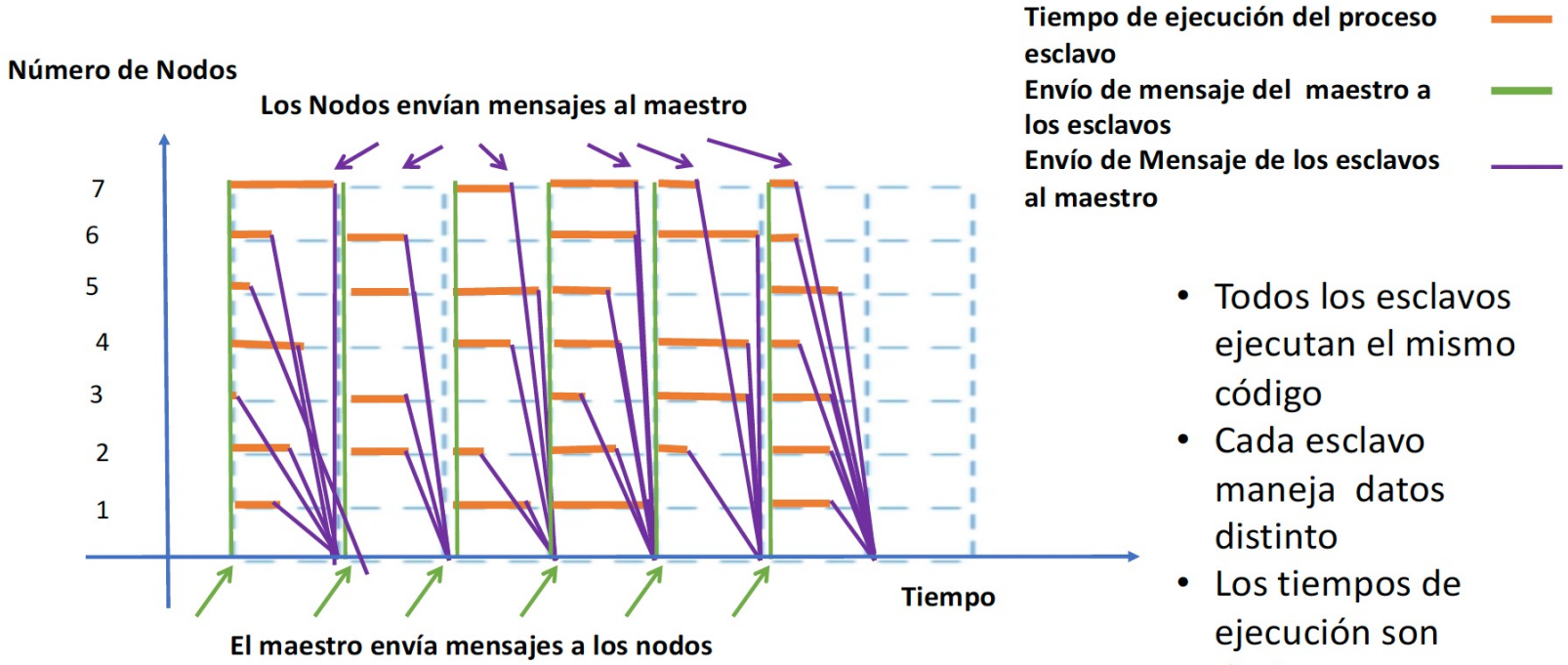

Nodos donde se ejecutan
los procesos sincrónicos
Cada nodo termina y
envía un mensaje al
maestro
4
3
2
1
El maestro envía un
mensaje a cada nodo
T1
T2
T3
T4
T5.
Fin
Todos los nodos inician
el procesamiento
Caso casi imposible
Tiempo de ejecución

Instantes de Cálculo y Comunicación en el Paradigma Maestro-Esclavo: Caso Ideal



- **Todos los esclavos ejecutan el mismo código**
- **Todos los nodos manejan la misma cantidad de datos**
- **Los tiempos de ejecución son idénticos**

Instantes de Cálculo y Comunicación en el Paradigma Maestro-Esclavo: Caso Real



- Todos los esclavos ejecutan el mismo código
- Cada esclavo maneja datos distinto
- Los tiempos de ejecución son distintos

Limitantes del Paralelismo

Evaluación del Rendimiento de

Códigos Paralelos

Conceptos y medidas

Rendimiento de un Código Paralelo

Herramientas para evaluar la calidad de nuestro código paralelo si contamos con un sistema que posee n nodos. Las fórmulas cuantitativas de rendimiento de códigos sirven para responder las siguientes preguntas:

- ¿Cuántas veces más rápido es el código paralelo respecto al secuencial?
- ¿Con cuántos procesadores lograré alcanzar la máxima aceleración “factible” de ese código?
- ¿Estoy subutilizando procesadores?

Aceleración (speedup)

Aceleración ($\mathbf{A_n}$): la aceleración experimentada por un código secuencial al reprogramarse como código paralelo para **n** procesadores.

$$A_n = \text{Tiempo secuencial/ Tiempo Paralelo} = T_1 / T_n$$

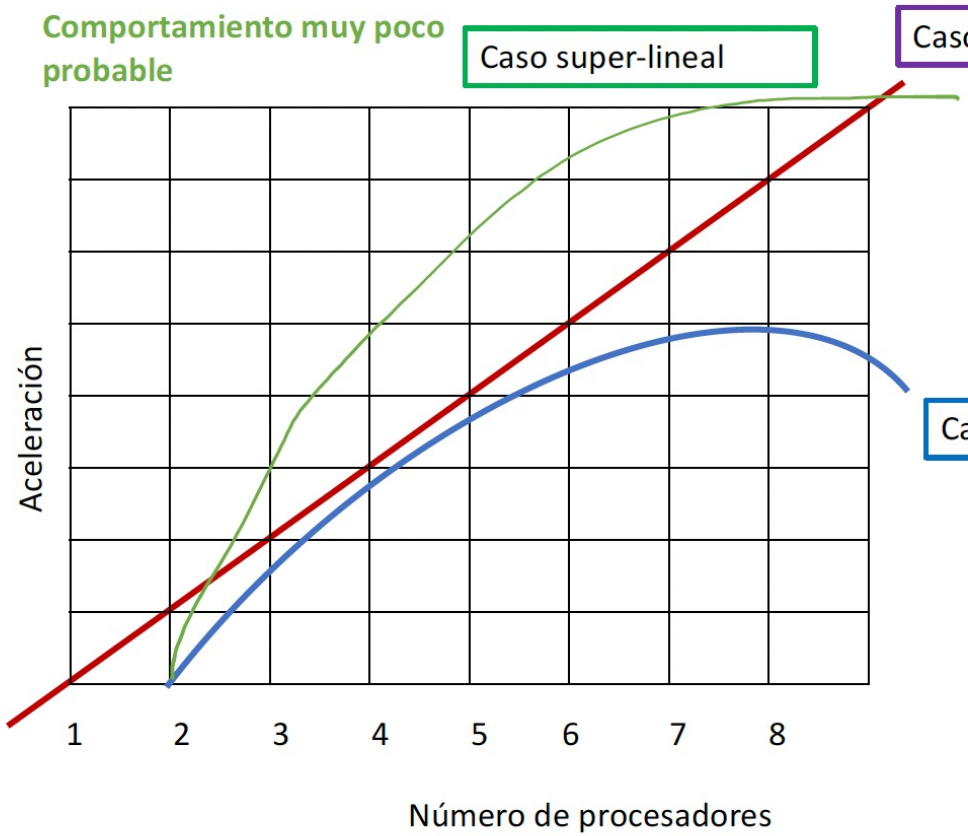
Si registramos los tiempos adecuados podemos saber cuantas veces más rápido se ejecuta el código paralelo respecto al código secuencial

Como $A_n = T_1 / T_n$

entonces $T_n = T_1 * A_n$

Haremos el siguiente experimento: estudiaremos como se comporta la curva de la aceleración a medida que aumentamos el número de procesadores

Aceleración (speedup)



El **caso lineal** es el comportamiento deseado pero poco probable.

- Decimos que el algoritmo escala linealmente con el aumento de los procesadores

El **caso infra-lineal** es el comportamiento esperado.

- Decimos que el algoritmo escala bien hasta 7 procesadores
- A partir de 7 procesadores ya no vale la pena seguir aumentando los procesadores

Eficiencia

Definición de Eficiencia (En):

Medida del trabajo útil de un conjunto de **n** procesadores al ejecutar un código paralelo.

$$E(n) = \text{Aceleración} / \text{número de procesadores} = A/n$$

El tiempo CPU, memorias y comunicación con la máquina, en cualquier sistema paralelo *es costoso.*

El tiempo de CPU se distribuye entre los usuarios trabajando en concurrencia con cuando nuestro código está en ejecución.

Entonces es importante no tener **procesadores ociosos.**

Eficiencia

$1/n \leq E(n) \leq 1$ ⇒

- $E(n) \rightarrow 1$ los p procesadores están ocupados
- $E(n) \rightarrow 0$ el programa se ejecuta en 1 solo procesador

Para optimizar el uso de las máquinas paralelas debemos

- Utilizar los procesadores al “máximo posible” (al menos por encima del 50% de utilización)
- Detectar en que momento la eficiencia comienza a bajar.

Ese será el punto de inflexión que nos indica que no vale la pena utilizar más procesadores

Eficiencia

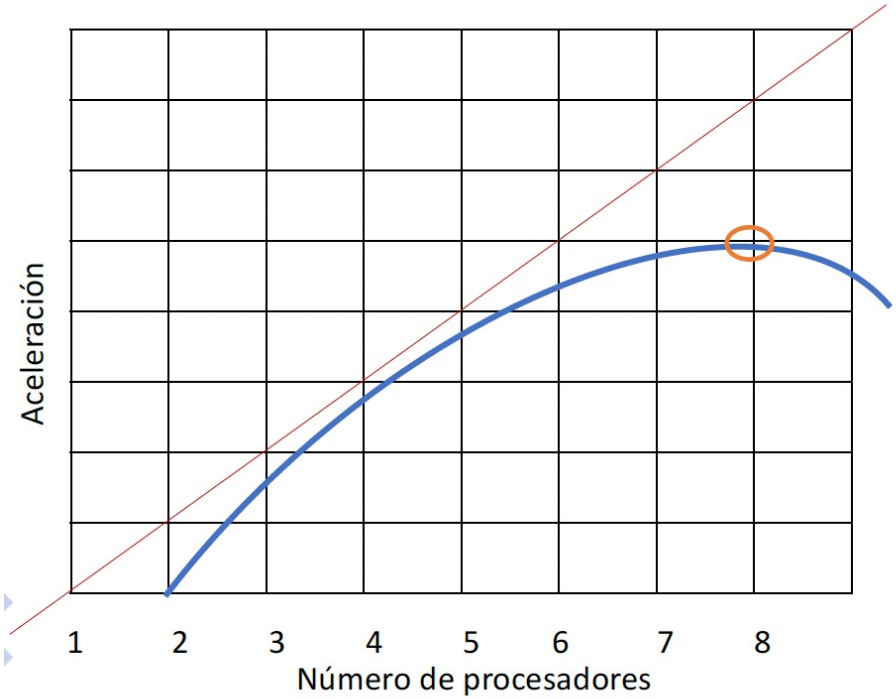
¿Cómo calculamos este punto de inflexión?

- Respuesta: Graficando la curva de aceleración del código paralelo con n procesadores. Esa curva debe ser exponencial creciente. El punto donde la curva deja de crecer me dice cual el el número óptimo de procesadores a utilizar
- Si graficamos el comportamiento de la velocidad vs el número de procesadores, podremos ver cuando la velocidad deja de decrecer

Escalabilidad:

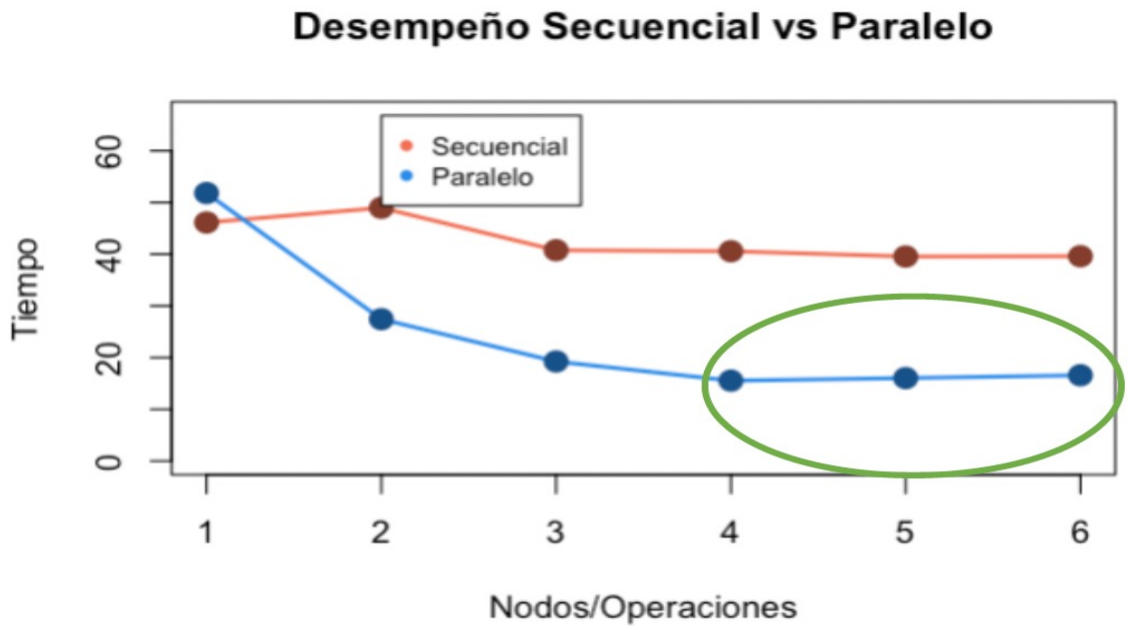
- Se dice que un código es escalable para un determinado rango de procesadores [1... n], si la eficiencia E(n) del sistema se mantiene constante y en todo momento por encima de un factor 0.5.

Eficiencia



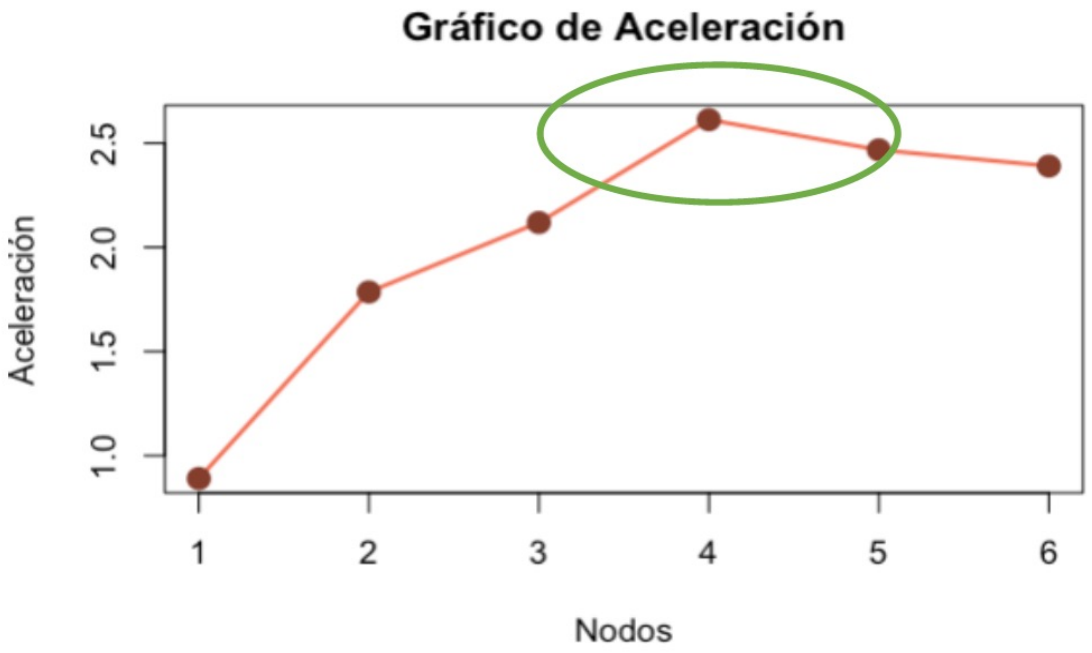
Este es el punto de inflexión a partir del cual baja la eficiencia de los procesadores

Curvas de comportamiento de los Tiempos



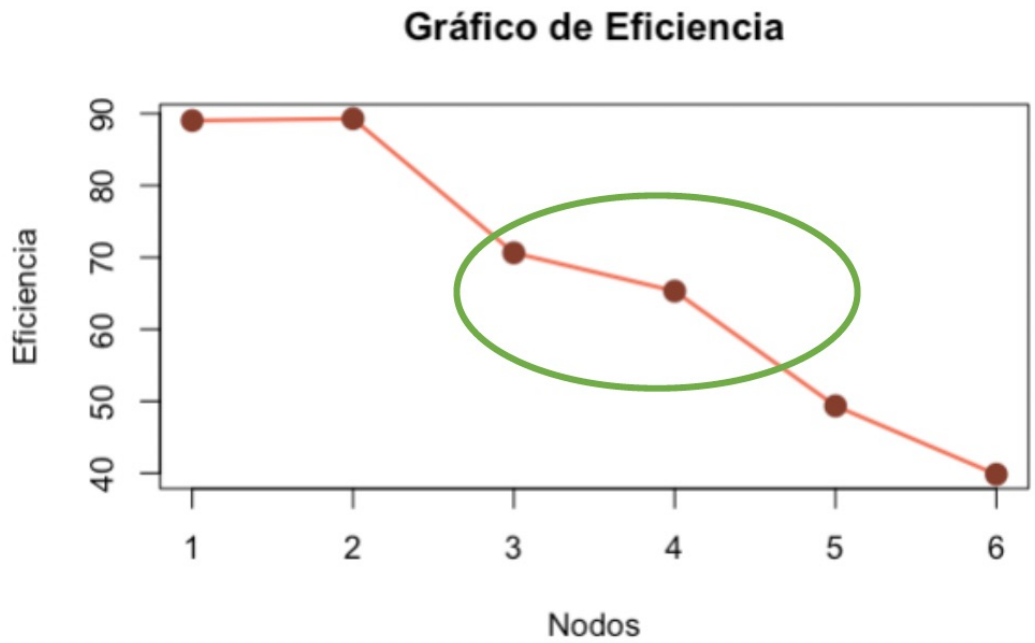
- A medida que agregamos más procesadores el tiempo de ejecución se reduce, hasta que se estabiliza

Aceleración



A partir del 4to nodo la curva de la aceleración deja de crecer, desciende

Eficiencia



Hasta el 4 nodo la eficiencia esta por encima del 50%, luego decae de manera importante

Conclusión

- Ley de Amdahl establece justamente que todo código paralelizable tiene un límite a partir del cual, aunque usted cuente con infinitos procesadores:
  - No podrá reducir más el tiempo de procesamiento
  - No acelerará mas el procesamiento del código
  - Estará subutilizando la capacidad de los procesadores de la maquina

El tiempo de máquina es costoso, sobre todo si utilizamos un supercomputador, entonces el análisis de rendimiento del código desarrollado, sobre esa máquina en particular, es muy importante

Seguimos …….