# *CORTO NO. 13*

*Marco Ramirez*

## Parte 1: Preguntas de concepto (20 puntos)

1. (5 puntos) Explica con tus propias palabras la diferencia entre descomposición funcional y descomposición de datos. Da un ejemplo para cada una.
    a. **Descomposición funcional**: Consiste en dividir un problema en diferentes tareas o funciones que se pueden ejecutar en paralelo. Cada una de estas tareas se asigna a diferentes procesadores o núcleos. Este enfoque es ideal cuando las distintas partes del problema son independientes entre sí.
        i. **Ejemplo**: En un sistema de criptomonedas, los nodos de la red pueden encargarse de diferentes tareas, como verificar transacciones o minar nuevos bloques. Cada nodo realiza una tarea distinta de manera simultánea.
    b. **Descomposición de datos**: Este enfoque divide grandes cantidades de datos en fragmentos más pequeños que pueden ser procesados en paralelo. Cada procesador trabaja sobre una porción del conjunto de datos, permitiendo la ejecución simultánea de múltiples procesos sobre esos datos.
        i. **Ejemplo**: En sistemas de Big Data como Hadoop o Spark, los datos se dividen en bloques y se distribuyen entre varios nodos para que cada uno procese su parte de forma paralela.
2. (5 puntos) ¿Qué ventajas y desventajas tiene el paralelismo manual en comparación con el paralelismo automático?
    a. **Paralelismo manual**:
        i. **Ventajas**:
            1. Mayor control por parte del programador sobre la asignación de tareas y el uso de recursos.

2. Posibilidad de optimizar la ejecución y distribución de las tareas, lo que puede mejorar el rendimiento en sistemas complejos.

ii. **Desventajas**:

1. Requiere más esfuerzo y conocimientos técnicos por parte del programador.
2. La implementación es más complicada y propensa a errores si no se tiene experiencia con la paralelización.

b. **Paralelismo automático**:

i. **Ventajas**:

1. Simplifica el desarrollo, ya que el compilador o el sistema gestiona automáticamente la paralelización.

ii. Es más fácil de implementar en lenguajes de alto nivel, lo que permite a los programadores centrarse en la lógica del problema.

c. **Desventajas**:

i. Menor control sobre cómo se ejecutan las tareas, lo que puede llevar a ineficiencias en algunos casos.

ii. Puede ser menos eficiente en situaciones que requieren una optimización detallada

3. (5 puntos) En el patrón de diseño 'Fork-Join', ¿qué representa la fase de 'fork' y qué representa la fase de 'join'? Proporciona un ejemplo práctico.
    a. **Fase 'fork'**: En esta fase, el problema principal se divide en múltiples subproblemas más pequeños. Estos subproblemas pueden ejecutarse de manera independiente y en paralelo en diferentes procesadores o hilos.
    b. **Fase 'join'**: Después de que todos los subproblemas se han resuelto, los resultados se combinan en la fase de 'join' para obtener una solución completa al problema original.
    c. **Ejemplo práctico**: Supongamos que queremos sumar una lista muy grande de números. En la fase 'fork', dividimos la lista en varias sublistas y asignamos cada sublista a un procesador. Cada procesador suma los números de su sublista. Luego, en la fase 'join', los resultados parciales se combinan para obtener la suma total.

4. (5 puntos) Describe el funcionamiento del patrón de diseño 'Stencil'. ¿En qué tipo de aplicaciones es más común su uso?
    a. **Funcionamiento**: El patrón Stencil se basa en actualizar elementos de una cuadrícula en paralelo, utilizando los valores de sus vecinos más cercanos (por ejemplo, los elementos arriba, abajo, izquierda y derecha). En cada iteración, el valor de un elemento se calcula en función de los valores de estos vecinos. Este patrón permite que múltiples celdas de la cuadrícula se actualicen simultáneamente, siempre y cuando no dependan de las mismas celdas vecinas.
    b. **Aplicaciones comunes**: Es utilizado principalmente en simulaciones científicas, procesamiento de imágenes y en modelos que requieren una cuadrícula estructurada, como la dinámica de fluidos o simulaciones basadas en métodos numéricos. Un ejemplo clásico es el procesamiento de imágenes, donde cada píxel puede actualizarse en paralelo basándose en los valores de los píxeles circundantes.

## Parte 2: Aplicación práctica (30 puntos)

5. (10 puntos) Dado un conjunto de datos de 100 millones de registros, diseña un plan de paralelización utilizando el patrón 'Map-Reduce' para procesar estos datos. Explica cómo dividirías los datos y cómo los resultados finales serían combinados.
    a. Como tenemos un conjunto de 100 millones de registros, lo primero sería dividir este conjunto en fragmentos más pequeños que puedan ser procesados en paralelo. Entonces podemos hacer 10 millones de registros en cada uno, lo que te daría 10 fragmentos, cada fragmento puede ser asignado a un thread o proceso separado.
        i. Ver código de ejemplo **problema_5.c**
6. (10 puntos) Imagina que estás trabajando en una simulación física donde se actualizan los estados de millones de partículas basadas en las posiciones de sus vecinas. ¿Qué patrón de diseño aplicarías y por qué? Explica cómo dividirías el trabajo entre múltiples hilos o procesadores.
    a. El **patrón Stencil** es ideal para problemas donde el estado de cada elemento (en este caso, una partícula) depende de los estados de sus vecinos cercanos. En cada iteración de la simulación, el valor o estado de una partícula se actualiza en función de las partículas que la rodean. Este enfoque es común en simulaciones físicas y de dinámica de fluidos, donde las interacciones locales son fundamentales para determinar el comportamiento general del sistema.
    b. División:
        i. **División de la cuadrícula**:
            1. Primero, dividiría el espacio que contiene las partículas en una cuadrícula, donde cada celda de la cuadrícula contiene un grupo de

partículas. Cada celda se puede procesar de forma paralela, ya que la actualización de las partículas en una celda solo depende de las partículas en las celdas vecinas (arriba, abajo, izquierda, derecha).

**ii. Asignación a hilos o procesadores:**

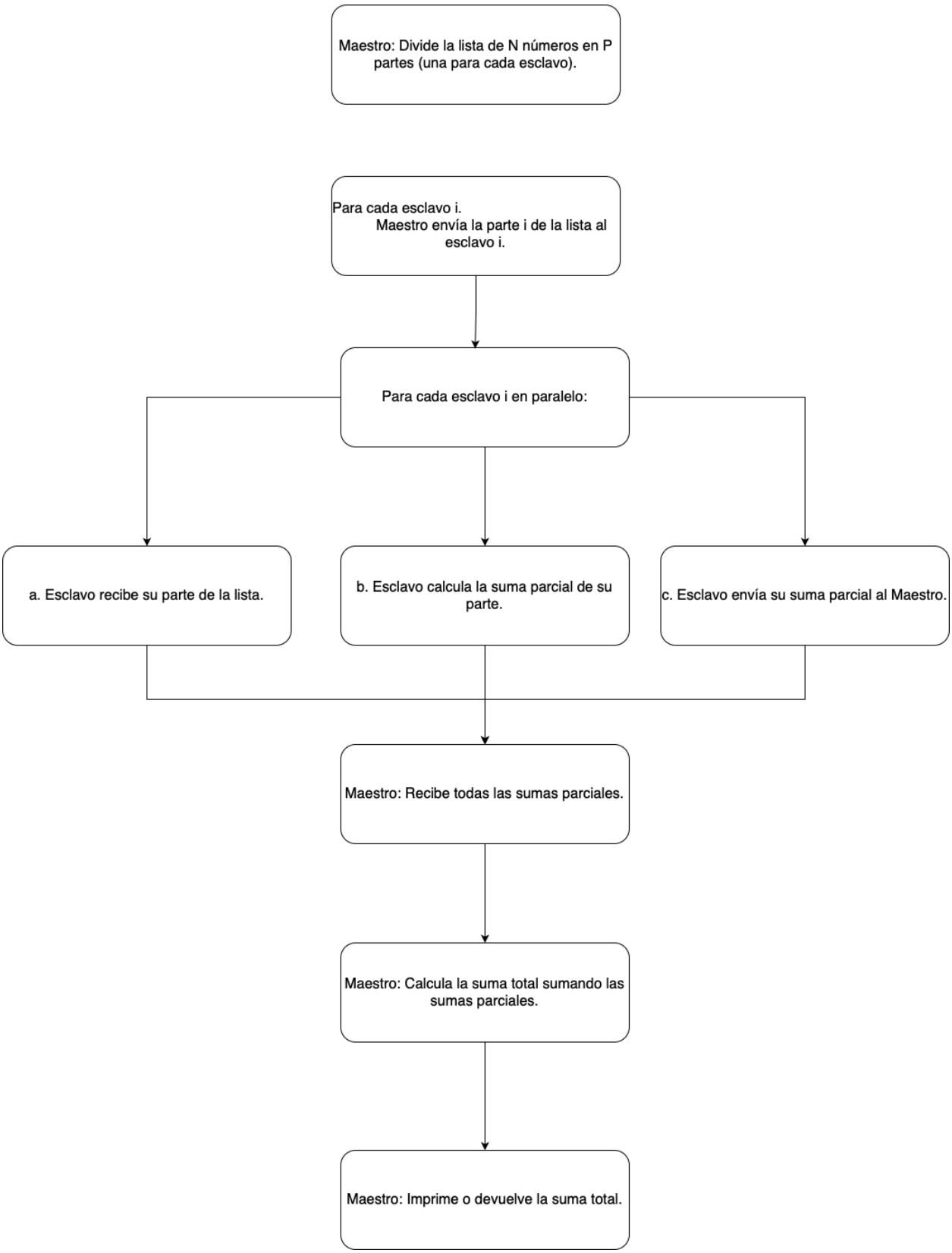
1. Luego, asignaría diferentes subregiones de la cuadrícula a diferentes hilos o procesadores. Cada hilo procesará las partículas dentro de su subregión. Como las actualizaciones de cada partícula solo dependen de sus vecinas cercanas, múltiples hilos pueden procesar distintas regiones de la cuadrícula en paralelo sin problemas de dependencia directa.

iii. **Sincronización entre hilos**:

1. En cada iteración de la simulación, cada hilo actualizaría las partículas en su subregión. Una vez que todos los hilos han completado sus cálculos, se sincronizan para asegurar que las actualizaciones sean visibles para la siguiente iteración, garantizando que los hilos tengan acceso a los valores actualizados de las partículas vecinas.

7. (10 puntos) Implementa en pseudocódigo un algoritmo que use el patrón 'Maestro/Esclavo' para realizar una tarea simple, como la suma de una lista de números distribuidos en múltiples procesadores.

   a. Ver el diagrama, siguiente página


Maestro: Divide la lista de N números en P partes (una para cada esclavo).
Para cada esclavo i.
Maestro envía la parte i de la lista al esclavo i.
Para cada esclavo i en paralelo:
a. Esclavo recibe su parte de la lista.
b. Esclavo calcula la suma parcial de su parte.
c. Esclavo envía su suma parcial al Maestro.
Maestro: Recibe todas las sumas parciales.
Maestro: Calcula la suma total sumando las sumas parciales.
Maestro: Imprime o devuelve la suma total.

## Parte 3: Proyecto Corto (50 puntos)

Implementa una pequeña simulación en el lenguaje de tu preferencia (Python, C, C++, etc.) utilizando el patrón Fork-Join. La simulación debe calcular la suma de los elementos de una lista dividida en sublistas, donde cada sublista es procesada en paralelo, y luego los resultados parciales son combinados.

Requisitos del proyecto:

- El código debe ejecutarse en paralelo utilizando al menos 4 hilos o procesadores.
- Documenta cómo lograste dividir las tareas y cómo manejas la combinación de los resultados.
- Entrega el código junto con un reporte en PDF que explique la arquitectura utilizada y los resultados obtenidos.

**Reporte**

***División de Tareas (Fork)***

Para implementar la simulación de suma de los elementos de una lista usando el patrón Fork-Join, primero dividí la lista de 100 elementos en 4 sublistas, una por cada hilo. Cada hilo tiene la responsabilidad de procesar una parte de la lista, es decir, de calcular la suma de los elementos asignados a su sublista.

***Proceso de division***

- La lista de entrada se divide en bloques de igual tamaño, lo que asegura que cada hilo trabaje con una porción equivalente de datos.
- Para hacer esto, calculé el tamaño de cada sublista usando la fórmula: tamaño_sublista = N / NUM_HILOS. Esto asigna a cada hilo un bloque de la lista original, asegurando una distribución equitativa de la carga de trabajo.
- Si el número total de elementos (N) no es divisible exactamente entre el número de hilos (NUM_HILOS), el último hilo puede recibir una porción ligeramente diferente, ajustando el tamaño del bloque en su iteración.

***Combinación de Resultados (Join)***

Después de que cada hilo ha procesado su parte de la lista y calculado la suma parcial, el siguiente paso es combinar estos resultados parciales para obtener la suma total de la lista original.

***Proceso de combinación:***

Una vez que cada hilo ha completado su tarea, utiliza pthread_join para garantizar que el hilo principal (Maestro) espere a que todos los hilos terminen su trabajo.

Los resultados parciales se almacenan en una variable llamada suma_parcial para cada hilo. Después de que cada hilo ha terminado, el hilo principal recoge esas sumas parciales y las combina sumándolas en una variable global suma_total, que contiene la suma total de todos los elementos de la lista.

### *Entorno de Desarrollo*

Para esta implementación, utilicé un MacBook Pro con M1, que tiene 10 núcleos de procesamiento y 16GB de RAM, lo que proporciona un entorno eficiente para realizar este tipo de simulaciones paralelas. Utilicé VSCode como entorno de desarrollo integrado (IDE) para escribir el código en C. El compilador utilizado fue Apple clang version 16.0.0, el cual admite la paralelización mediante el uso de la biblioteca pthread en POSIX.

### *Resultados*