

Introducción a Paralelismo

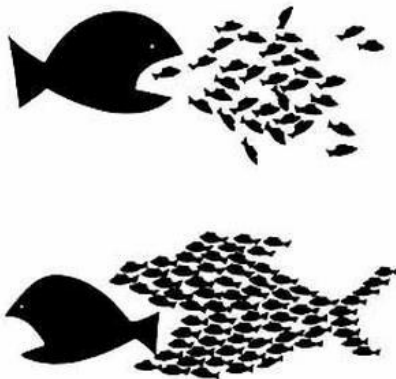
Conceptos básicos

Oscar A. Esquivel-Flores

LCD-UNAM

26 de noviembre de 2021

¿Divide y vencerás?



Paralelismo I

- La velocidad de los procesadores depende del número de circuitos que posee.
- El aumento de la velocidad conduce al calentamiento excesivo de los circuitos con lo que el consumo de energía del procesador también se incrementa.
- Debido a la ineficiencia en el control de energía y el incremento de la densidad de circuitos integrados se ha considerado como alternativa el paralelismo.
- La alternativa es colocar muchos procesadores, relativamente simples, en un solo chip (arquitectura multicore) en lugar de crear un procesador único muy complejo (single-core).

Paralelismo II

- El número de los problemas que requieren más recursos computacionales se incrementa al mismo nivel que el avance de la ciencia y la tecnología.
 - Modelado del clima
 - Dinámica de proteínas
 - Diseño de fármacos
 - Exploración y explotación energética
 - Análisis de datos

Programas Paralelos I

- La mayoría de los programas son diseñados para un sistema single-core. Correr multiples instancias de estos programas no utilizan las ventajas de un sistema multi-core.
- Sin embargo, aunque puedan identificarse acciones en un programa serial que son posibles de ser paralelizadas, una eficiente implementación paralela de un programa serial no se obtiene de una secuencia de paralelizaciones.

Programas Paralelos II

- Podría esperarse que el diseño de programas paralelos identifique tareas seriales comunes y sean paralelizadas eficientemente.
- Sin embargo, con base en lo anterior, si el diseño serial es más complicado entonces se complicaría también reconocer qué paralelizar y por tanto hacerlo de manera eficiente.
- No es suficiente generar algoritmos seriales y paralelizar sus partes, es necesario vislumbrar una *forma de concebir completamente un nuevo algoritmo que explote el poder de múltiples procesadores*.

Programas Paralelos III

- Escribir un programa paralelo se basa en la idea de segmentar o **particionar** el trabajo entre muchos cores.
- Existen dos tipos de diseños básicos: **paralelismo de tareas** y **paralelismo de datos**.
- Repartir el trabajo entre cores requiere coordinar tareas, lo cual involucran la **comunicación** y **balanceo de carga** de trabajo entre los cores.
- Otro tipo de coordinación, es la **sincronización**.

Programas Paralelos IV

- Actualmente la mayoría de programas paralelos utilizan paralelismo explícito, es decir, son extensiones de lenguaje C, C++ que incluyen **instrucciones explícitas** para paralelizar.
- Existen lenguajes de alto nivel que permiten hacer paralelismo, pero tienden a sacrificar el rendimiento por facilitar el desarrollo (*are you shure?*).

Multitareas

- **Multitasking:** El Sistema Operativo ofrece facilidades para ejecutar múltiples programas aparentemente al mismo tiempo (simultáneamente), aún en un sistema mono-core.
 - Esto es posible pues cada proceso se ejecuta en un pequeño intervalo de tiempo (*ms*).
 - Distintos programas son ejecutados en una ventana de tiempo (*time slicing*).
- El S.O. puede cambiar la ejecución de los procesos varias veces, administrar recursos y pausar procesos (blocking).

Paralelismo a nivel instrucción

- **Paralelismo a nivel instrucción (ILP)**: Intenta mejorar el rendimiento del procesador al tener múltiples unidades funcionales (componentes del procesador) trabajando simultáneamente. Existen dos tipo:
 - **pipelining**
 - **Inicialización múltiple** (*multiple issue*): Optimización del Pipelining, conecta en secuencia pedazos de hardware y unidades funcionales. Puede ser:
 - estático (planificación al compilarse),
 - dinámico (planificación al tiempo de ejecución).
- Procesadores que soportan inicialización múltiple dinámica son conocidos como superescalares.
- Los superescalares anticipan (especulan) la instrucción previa que se va ejecutar.

Paralelismo a nivel thread

- **Paralelismo a nivel thread (TLP):** Intenta proveer paralelismo al hacer ejecuciones simultáneas de diferentes threads:
 - *Coarser-grained:* Satura la ejecución simultánea
 - *Finer-grained:* Ejecución de instrucciones individuales

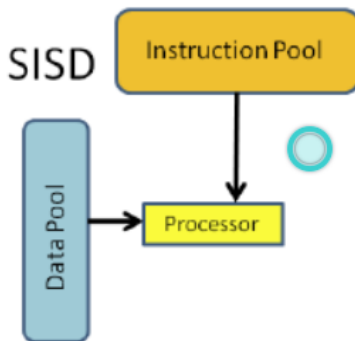
Hardware Paralelo I

- En el paralelismo es común utilizar la **taxonomía de Flynn** para clasificar la arquitectura de computadoras.
- Esta clasificación caracteriza un sistema con base en el cantidad de flujo de instrucciones y la cantidad de flujo de datos que pueden ser manejados simultáneamente.

		Datos	
		Simples	Múltiples
Instrucciones	Simples	SISD	SIMD
	Múltiples	MISD	MIMD

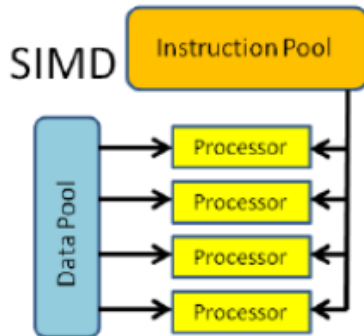
Hardware Paralelo II

- **Una instrucción, un dato (SISD):** Computadora secuencial que no explota el paralelismo, propio de las arquitecturas Von-Neumann. Un único procesador ejecuta un solo flujo de instrucciones para operar datos en una única memoria.



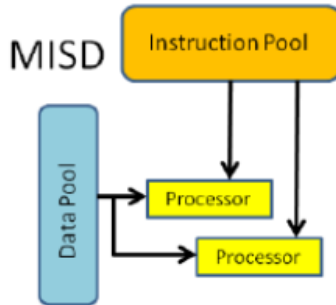
Hardware Paralelo III

- **Una instrucción, múltiples datos (SIMD):** Todas las unidades de procesamiento ejecutan la misma instrucción sincronizadamente sobre distintos datos. Es propia de arquitecturas vectoriales o matriciales.



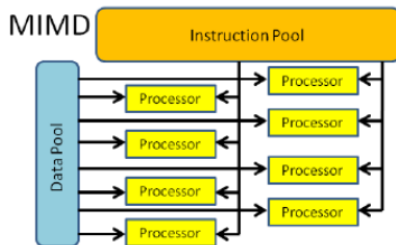
Hardware Paralelo IV

- **Múltiples instrucciones, un dato (MISD):** Es una arquitectura poco común, asociada a sistemas de seguridad o arquitecturas vectoriales segmentadas. Es inusual debido a que los múltiples flujos de instrucciones implican múltiples flujos de datos.



Hardware Paralelo V

- **Múltiples instrucciones, múltiples datos (MIMD):**
Diversos procesadores autónomos que ejecutan simultáneamente instrucciones diferentes sobre datos diferentes. Es una arquitectura propia de los sistemas distribuidos con memoria compartida o distribuida.



Sistemas MIMD I

Son sistemas que ejecutan instrucciones simultáneas múltiples que operan sobre flujos de datos múltiples

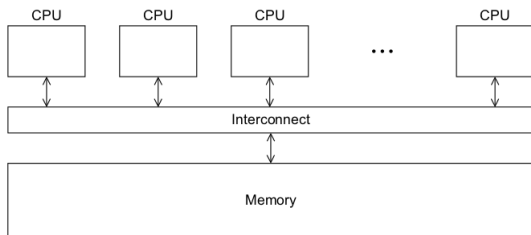
- Consisten de una **colección de unidades de procesamiento independiente** (cores) que tienen sus propias unidades de control (UC) y aritmética-lógica (ALU).
- Estos sistemas son **asincrónicos**, cada unidad de procesamiento opera a su ritmo.
- En varios de los sistemas MIMD no se cuenta con un **reloj global** por lo que no hay relación entre los procesadores

Sistemas MIMD II

- Se requiere un sistema de sincronización para **coordinar la ejecución** de instrucciones en distintos procesadores
- Hay dos arquitecturas principales que soportan este tipo de sistemas MIMD:
 - Sistemas de memoria compartida (**SMC**)
 - Sistemas de memoria distribuida (**SMD**)

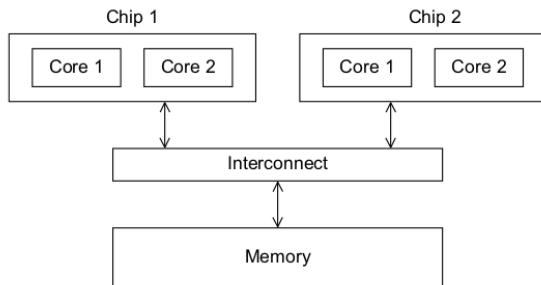
Sistemas memoria compartida I

- Una colección de procesadores autónomos está conectada a un **sistema de memoria** mediante una red de interconexión
- Los procesadores usualmente se comunican **implícitamente** al acceder a **estructuras de datos compartidas**



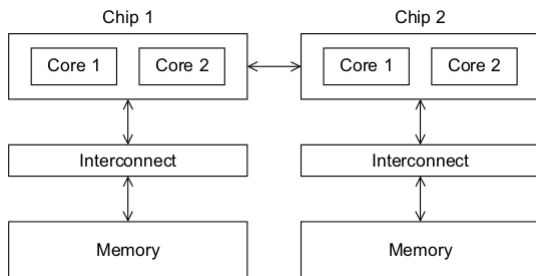
Sistemas memoria compartida II

Un sistema multicore con **acceso de memoria uniforme** conecta todos los procesadores directamente a la memoria principal



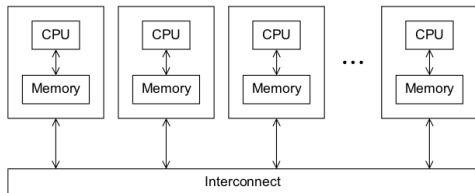
Sistemas memoria compartida III

Sistema multicore con acceso de **memoria no uniforme** tiene una conexión directa a cada bloque de la memoria principal, los procesadores pueden acceder a otro bloque mediante hardware.



Sistemas memoria distribuida

- Cada procesador está asociado con una unidad de memoria privada
- Cada par procesador-memoria se comunica a través de una red de comunicación con los demás **nodos**.
- La comunicación es **explícita** por medio de:
 - Paso de mensajes
 - Uso de funciones especiales que proveen acceso a la memoria de otro procesador



Software Paralelo I

En los **sistemas de memoria compartida**:

- La comunicación es explícita
- Los procesos/hilos creados se comunican por medio de **variables compartidas**
- Los hilos pueden ser **hilos dinámicos** o **hilos estáticos**
- Un problema común debido a la ejecución asincrónica de los procesadores es el **no determinismo**

Software Paralelo II

- Otros problemas persistentes son las **condiciones de carrera** dentro de **secciones críticas**
- Por lo que es necesario crear mecanismos de **sincronización** para asegurar la **exclusión mutua**
- Estos mecanismos pueden ser **locks**, **mutex**, **semáforos**, **monitores**
- Una alternativa a los textbf mutex son busy-waiting

Software Paralelo III

En los **sistemas de memoria compartida**:

- Los procesadores acceden a su **memoria privada**
- Es común la creación de **procesos** en lugar de hilos
- La comunicación es **explícita** por medio de **paso de mensajes** entre procesos
- El paso de mensajes se realiza por medio de APIs que proveen dos operaciones básicas: **send** y **receive**

Software Paralelo IV

En los **sistemas de memoria compartida**:

- Los procesos serán numerados, por medio de este **identificador** se hace el envío y recepción del mensaje.
- El modelo de ejecución es **SPMD**: un solo programa - múltiples datos
- Una copia del programa se ejecuta en cada procesador donde se realizan diferentes acciones
- Comúnmente los mensajes son enviados a diferentes **bloques** de memoria en distintos procesadores
- Otras funciones utilizadas son **broadcast**, **reduction**

Software Paralelo V

- Al desarrollar aplicaciones paralelas se debe pensar en la ejecución de **procesoshilos** que llevan acabo **distintas tareas**
- El software paralelo disponible actualmente, va desde **APIs y estándares** hasta paquetes que proveen **mecanismos de paralelización**:
 - **POSIX (Portable Operating System Inteface for X)**: Intefaz Portátil de Sistema Operativo tipo UNIX
 - **Pthreads**: Biblioteca para el manejo de hilos en POSIX
 - **OpenMP**: API para la programación de multiporceso
 - **MPI (Message passing interface)**: Interfaz de paso de mensajes
 - **CUDA**: Compute Unified Device Architecture. Plataforma de programación paralela para la programación de dispositivos GPUs

Software Paralelo VI

- Otras alternativas son:
 - Matlab, R
 - Python
 - Julia
 - Paquetes, bibliotecas de OpenMP, MPI para Matlab, R, Python, Julia
 - **Numba**: Compilador JIT que traduce código Python para acelerar ejecución y programar GPUs.
 - **CUDA.jl**: Envoltorio de CUDA para Julia

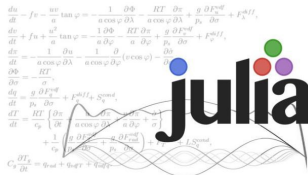
Software Paralelo VII

- Numba:
 - Ofrece macros para acelerar funciones en Python
 - Crea threads simplificados
 - Desarrolla operaciones vectorizadas (SIMD)
 - Traduce código para programar GPUs



Software Paralelo VIII

- Julia:
 - Crea código compilado
 - Ofrece funciones matemáticas altamente optimizadas
 - Contiene macros para acelerar la ejecución de funciones
 - Crea threads simplificados
 - Desarrolla operaciones vectorizadas (SIMD)
 - Traduce código para programar GPUs
 - Dotado de paquetes para implementar diversos tipos de paralelismo



The image shows the Julia logo, which consists of the word "julia" in a stylized, lowercase font. Above the letters are four colored circles: green, red, blue, and purple. The logo is overlaid on a background of complex mathematical formulas, including partial derivatives and trigonometric functions, such as $\frac{du}{dt} = f(u) - \frac{uv}{a} \tan \varphi$ and $\frac{d\varphi}{dt} = \frac{g}{a} \frac{\partial F^{eq}}{\partial \sigma} + f^{eq} I_1$.

Diseño de un programa paralelo I

- La idea de dividir un programa secuencial en procesos/threads, equilibrar la carga de trabajo, sincronizarlos y minimizar la comunicación desafortunadamente **no** es un diseño paralelo
- **Ian Foster** provee una secuencia de pasos para diseñar un programa paralelo¹:
 - **Particionar** (*Partitioning*)
 - **Comunicar** (*Communication*)
 - **Agregar** (*Aggregation*)
 - **Asignar** (*Mapping*)

¹Foster Ian, "Designing and building parallel programs" 

Diseño de un programa paralelo II

- **Particionar:** Divide los cálculos a realizar y los datos que se utilizarán en el cálculo en pequeñas tareas. Identificar las tareas que pueden ejecutarse en paralelo es el aspecto relevante en este punto.
- **Comunicar:** Determina la comunicación entre las tareas identificadas en el punto anterior y que será necesario establecer.

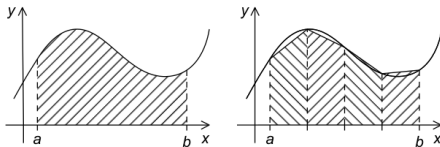
Diseño de un programa paralelo III

- **Agregar:** Combina en tareas mas grandes las subtareas y comunicaciones definidas en el punto anterior. Esto es, si existen secuencias de subtareas específicas se conjuntan en una subtaska más grande.
- **Asignar:** Asigna la subtareas compuestas a procesos/threads. Esto debe hacerse de tal forma que la comunicación sea mínima y la carga de trabajo sea equivalente.

Diseño de un programa paralelo IV

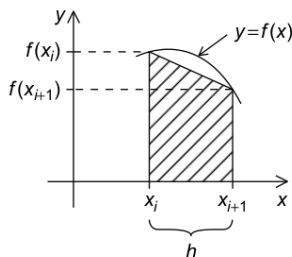
Un ejemplo de introducción sencillo e ilustrativo es el cálculo del área bajo la curva descrita por la función $y = f(x)$ acotada en un intervalo $[a, b]$ por medio de **la regla del trapecio**

- La idea básica consiste en dividir el intervalo $[a, b]$ en n subintervalos y calcular la área de cada segmento A_i para para aproximar el área total A



Diseño de un programa paralelo V

- Aproximar el área de cada subintervalo por medio de un trapecio
- Al tomar dos puntos x_i y x_{i+1} la longitud del subintervalo es $h = x_{i+1} - x_i$
- La longitud de los segmentos verticales del trapecio será $f(x_i)$ y $f(x_{i+1})$
- El área del trapecio será $A_i = \frac{h}{2}[f(x_i) + f(x_{i+1})]$



Diseño de un programa paralelo VI

- El tamaño de cada subintervalo es $h = \frac{b-a}{n}$
- Sean $x_0 = a$ y $x_n = b$, la longitud del subintervalo se tiene:
- con
$$x_0 = a, x_1 = a + h, x_2 = a + 2h, \dots, x_{n-1} = a + (n-1)h, x_n = b,$$
- La longitud de los segmentos verticales del trapecio estará dada por $f(x_i)$ y $f(x_{i+1})$
- La suma de las áreas de los trapecios será:
$$A = h[f(x_0)/2 + f(x_1) + f(x_2) + \dots + f(x_{n-1}) + f(x_n)/2]$$

Algoritmo Paralelo I

Recordemos los pasos del diseño paralelo: Particionar, Identificar, Agregar y Asignar.

- Particionar: Identificamos dos tareas principales:
 - Encontrar el área de los trapecios individuales
 - Calcular la suma de esas áreas
- Identificar: Por medio del canal de comunicación se reunirá cada una de las areas individuales para calcular la suma global
- ¿Podremos agregar las tareas y asignarlas a distintas unidades de procesamiento?

Algoritmo Paralelo II

Recordemos los pasos del diseño paralelo: Particionar, Identificar, Agregar y Asignar.

- Podemos intuir que mientras más subáreas se calculen mejor será la aproximación de A
- Debemos usar varios trapecios
- Se calcularán mas áreas de trapecios que unidades de procesamiento a utilizar.
- Se conjuntarán las areas de los trapecios en grupos.
- Se dividirá el intervalo $[a, b]$ en `comm_sz`
- Un proceso, digamos el proceso 0, sumará las sumas parciales

Algoritmo Paralelo III

Recordemos los pasos del diseño paralelo: Particionar, Identificar, Agregar y Asignar.

- ① a, b, n
- ② $h = (b - a) / n$
- ③ `local n = n / comm sz`
- ④ `local a = a + my rank * local n * h`
- ⑤ `local b = local a + local n * h`
- ⑥ `local integral = Trapecio(local a, local b, local n, h)`