

# Computación Paralela y Distribuida

2022-II

---

José Fiestas

17/08/22

Universidad de Ingeniería y Tecnología  
jfiestas@utec.edu.pe

# Unidad 1: Fundamentos de paralelismo y arquitecturas paralelas

Al finalizar la unidad, los alumnos conocen:

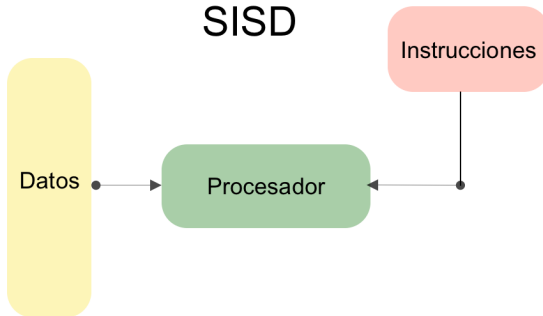
1. Transición del procesamiento secuencial al paralelo
2. Taxonomía de Flynn
3. Arquitecturas paralelas: memoria compartida vs memoria distribuida.
4. Paradigmas del paralelismo
5. Metas del Paralelismo: velocidad y precisión
6. DAG (Directed Acyclic Graphs)

## 2. Taxonomia de Flynn

---

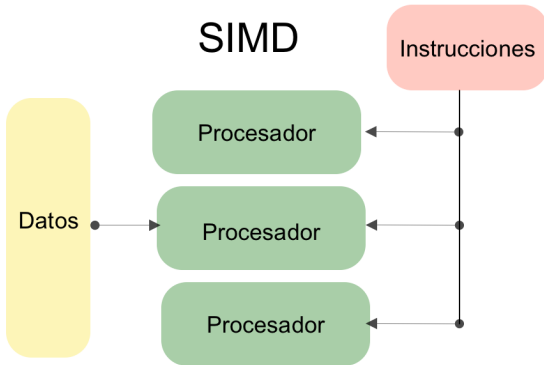
# Single Instruction Single Data

Procesadores secuenciales ejecutan una instrucción sobre una unidad de memoria (dirección). Arquitectura Von Neumann es SISD. E.g. microcontroladores



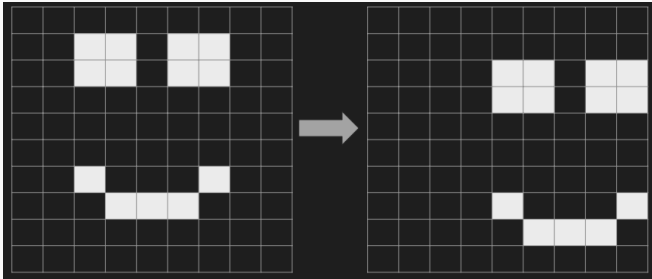
# Single Instruction Multiple Data

Una única instrucción (tarea) se ejecuta en uno o más núcleos en múltiples **data streams**, en forma simultánea. E.g. GPUs



# Single Instruction Multiple Data

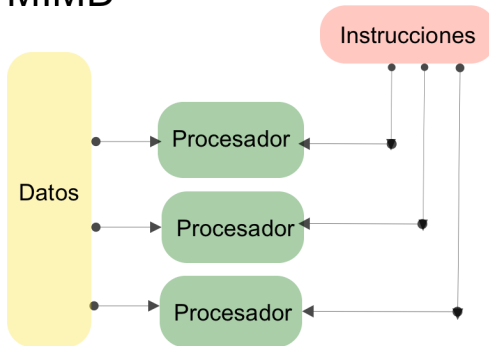
E.g. moviendo una imagen pixel a pixel. Instrucción:  $x+2, y-1$



# Multiple Instruction Multiple Data

Múltiples tareas en múltiples procesos se ejecutan en distintos **data streams** en forma simultánea . E.g. arquitecturas de memoria compartida o distribuída.

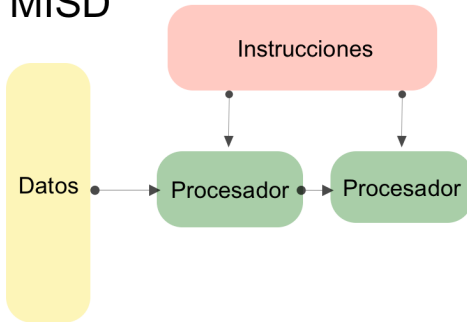
## MIMD



# Multiple Instruction Single Data

Multiples procesos ejecutan distintas instrucciones en la misma data. E.g. detección de errores en tiempo real (sistemas de navegacion aerea)

MISD





### **3. Arquitecturas paralelas: memoria compartida vs memoria distribuida.**

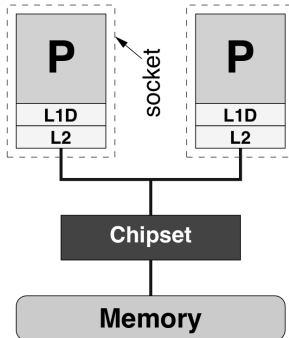
---

# Computadoras de memoria compartida

Es un sistema en el que los CPUs comparten un mismo espacio físico (dirección). Dependiendo del acceso a memoria, pueden ser:

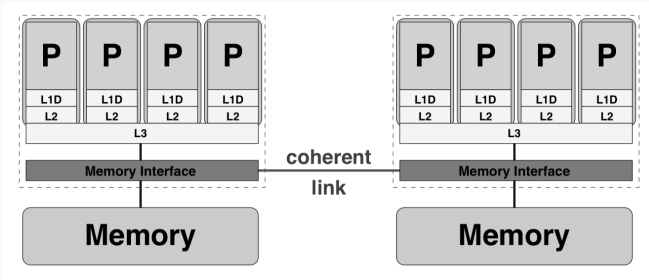
## **UMA:** Uniform

Memory Access, es un modelo 'plano' de acceso a memoria (latencia = ancho de banda para todos los procesadores). También llamado Simmetric Multiprocessing (SMP).  
E.g. procesadores multicore.



# Computadoras de memoria compartida

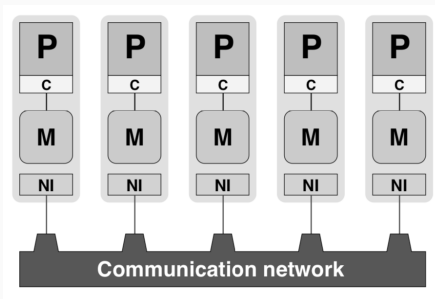
**ccNUMA:** Cache Coherent Non-uniform Memory Access, la memoria esta físicamente distribuida, pero una red lógica permite que funcione como memoria en un mismo espacio físico (dirección)



En ambos casos la **coherencia de Caché** permite mantener consistencia entre las cachés y la memoria en caso se realicen modificaciones por algún CPU

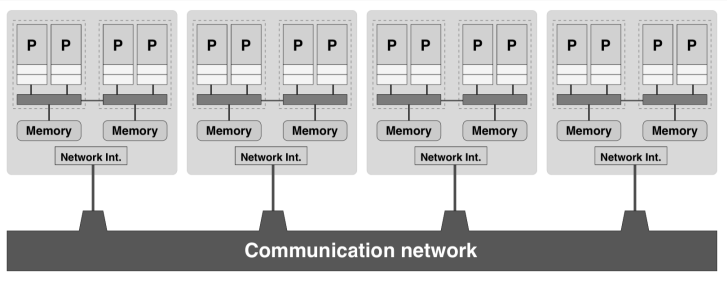
# Computadoras de memoria distribuída

En este sistema, cada procesador está conectado a una memoria local, y ningún otro tiene acceso a esta memoria. Los procesadores se comunican a través de una red enviando a, y recibiendo mensajes de los otros procesadores.



# Sistemas híbridos

En la práctica, los sistemas no son puramente distribuidos o compartidos, sino una combinación de ambos (híbridos). I.e., nodos de memoria compartida conectados a través de una red, que añaden una complejidad mayor al sistema de comunicación entre procesadores. E.g. CPU-GPU clusters



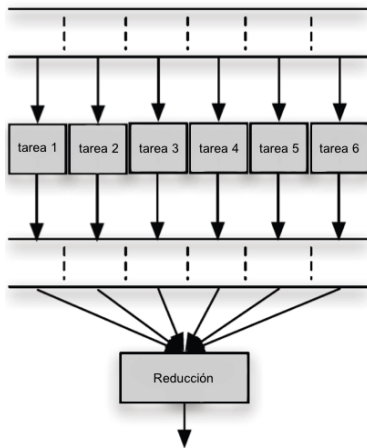
## **4. Métodos de paralelismo (paradigmas)**

---

# Paralelismo de la data (del resultado)

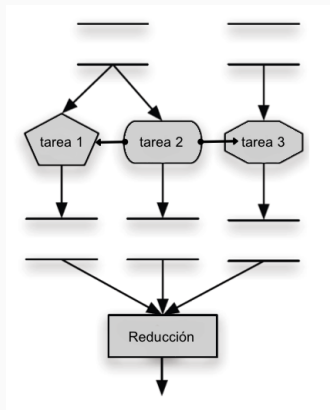
La estructura del resultado define el programa. Cada proceso solo es responsable de una tarea.

La información entre los procesos no se envía, sino se lee de la memoria compartida. Procesos y comunicación es implícita (OMP). Se construye alrededor del resultado final, procesando los elementos del resultado simultáneamente



# Paralelismo de la tarea (especialista)

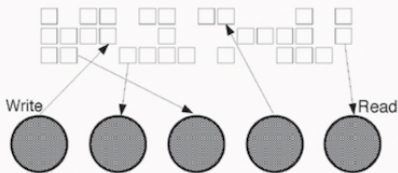
Comunicación (envío de mensajes) entre procesos. Utilizado en programación en paralelo (MPI) y orientada a objetos (C,C++,Fortran) No utiliza memoria compartida, sino espacio de memoria particionado en p nodos. Paralelismo es explícito. Se paraleliza la tarea (especialidad) y se agrupa a los especialistas conectados en paralelo en una red lógica





# Paralelismo de la agenda (híbrido)

Caso intermedio, que mantiene la distinción entre un grupo de datos (objetos) y un grupo de procesos. Se construye enfocado en la agenda de tareas, y se asigna procesos paralelos para cumplirla.



**Figure 2.3**

Distributed data structures: Concurrent processes *and* data objects figure as autonomous parts of the program structure. Processes communicate by reading and writing shared data objects

## **5. Metas del paralelismo**

---

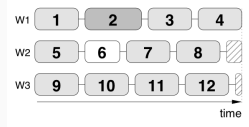
El problema de diseño de un algoritmo paralelo se refiere a

- Dado un algoritmo, buscar la arquitectura adecuada en paralelo
- Dada una arquitectura, buscar el algoritmo adecuado en paralelo. Es el caso clásico, que requiere :
  - Identificar **tareas** (tasks) con o por **procesos** (threads)
  - Diseñar un plan de ejecución (**scheduling**), de acuerdo a las dependencias y el I/O
  - Identificar los puntos de **comunicación** entre procesos y del I/O

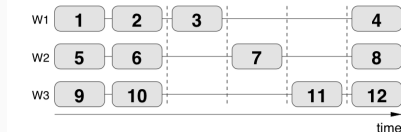
# Problemas en el diseño en paralelo

Los principales problemas que aparecen son:

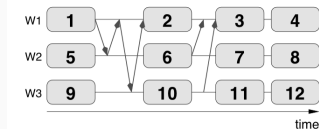
## - Desbalance de cargas



## - Cuellos de botella



## - Comunicación



¿Cómo programar en paralelo?

- 1 - Escoger el paradigma (método) más cercano al problema
- 2 - Desarrollar código de acuerdo al paradigma elegido
- 3 - Si el código no es eficiente, retornar al paso 1

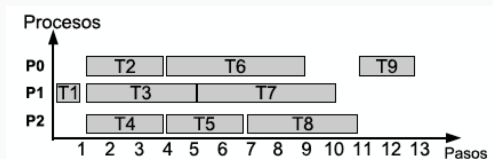
A tener en cuenta es el grado de **sincronización** y **comunicación** entre procesadores.

Un mal diseño de comunicación implica mayor costo computacional, mientras que una mala sincronización origina tiempos muertos (inactividad de procesadores)

# Objetivos




**Primer objetivo** es *minimizar el tiempo total de ejecución*. I.e.

- **Tiempo de cálculo** propiamente dicho, se optimiza asignando tareas a procesadores distintos para maximizar la concurrencia
- **Tiempo de comunicación**, asignando tareas independientes en cada proceso
- **Tiempo de ocio**, evitar tiempos de inactividad.



**Segundo objetivo** es mantener (o mejorar) la *precisión del algoritmo* secuencial correspondiente.

Este no es un problema trivial

-  David B. Kirk and Wen-mei W. Hwu *Programming Massively Parallel Processors: A Hands-on Approach*. 2nd. Morgan Kaufmann, 2013. isbn: 978-0-12-415992-1.
-  Norm Matloff. *Programming on Parallel Machines*. University of California, Davis, 2014.
-  Peter S. Pacheco. *An Introduction to Parallel Programming*. 1st. Morgan Kaufmann, 2011. isbn: 978-0-12-374260- 5.
-  Michael J. Quinn. *Parallel Programming in C with MPI and OpenMP*. 1st. McGraw-Hill Education Group, 2003. isbn: 0071232656.
-  Jason Sanders and Edward Kandrot. *CUDA by Example: An Introduction to General-Purpose GPU Program- ming*. 1st. Addison-Wesley Professional, 2010. isbn: 0131387685, 9780131387683.