Paradigmas de programación paralela

Universidad Nacional de La Plata

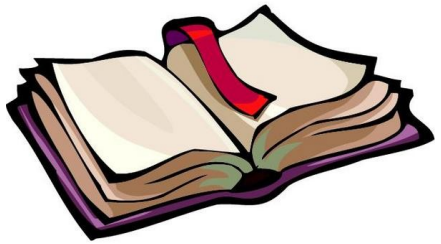
Facultad de Informática

Prof. Dr. Adrian Pousa

Agenda

I. Paradigmas de programación paralela
   i. Paralelismo de datos - SIMD/SPMD
   ii. Divide y vencerás
   iii. Pipelines
   iv. Algoritmos sistólicos
   v. Master-Worker
   vi. Task pools

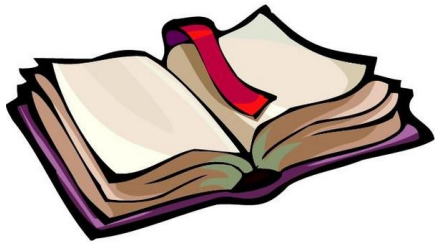
II. Modelos híbridos



Agenda

I. Paradigmas de programación paralela
   i. Paralelismo de datos - SIMD/SPMD
   ii. Divide y vencerás
   iii. Pipelines
   iv. Algoritmos sistólicos
   v. Master-Worker
   vi. Task pools

II. Modelos híbridos



Paradigmas de programación paralela

- Los programas paralelos consisten de una colección de tareas ejecutadas por procesos o hilos sobre múltiples unidades de procesamiento
- Existen varias formas de organizar y estructurar un programa paralelo que pueden ser capturadas por un paradigma o patrón de programación específico

Paradigma/Patrón de programación paralela: clase de algoritmos que resuelven distintos problemas, pero tienen la misma estructura.

- Existen distintas clasificaciones. Los patrones más comunes son:

Paralelismo de datos - SIMD/SPMD

Divide y vencerás

Pipelines

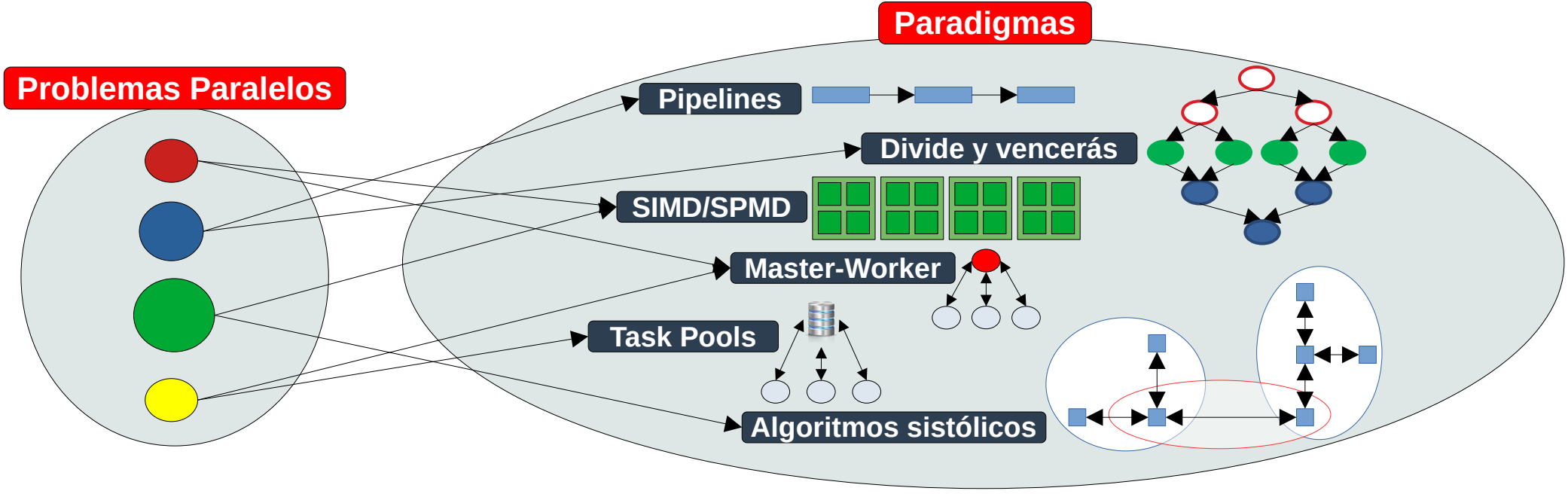
Algoritmos sistólicos

Master-Worker

Task Pools

Paradigmas de programación paralela

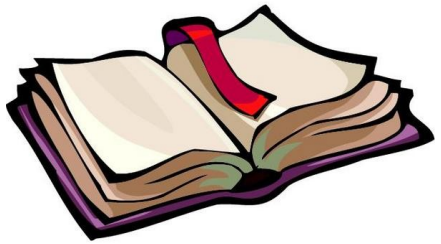
- Para cada paradigma puede escribirse un esqueleto algorítmico que define la estructura de control común.
- En cada paradigma los patrones de comunicación son muy similares y es posible encuadrar distintos tipos de problemas paralelos en uno o mas de estos paradigmas.



Agenda

I. Paradigmas de programación paralela
   i. Paralelismo de datos - SIMD/SPMD
   ii. Divide y vencerás
   iii. Pipelines
   iv. Algoritmos sistólicos
   v. Master-Worker
   vi. Task pools

II. Modelos híbridos



Paradigmas de programación paralela
Paralelismo de datos - SIMD/SPMD

- El **paralelismo de datos** es el paradigma más común y el más simple: las tareas se mapean estática o semiestáticamente a unidades de procesamiento, y luego los procesos o hilos realizan **operaciones similares o iguales** sobre diferentes datos.

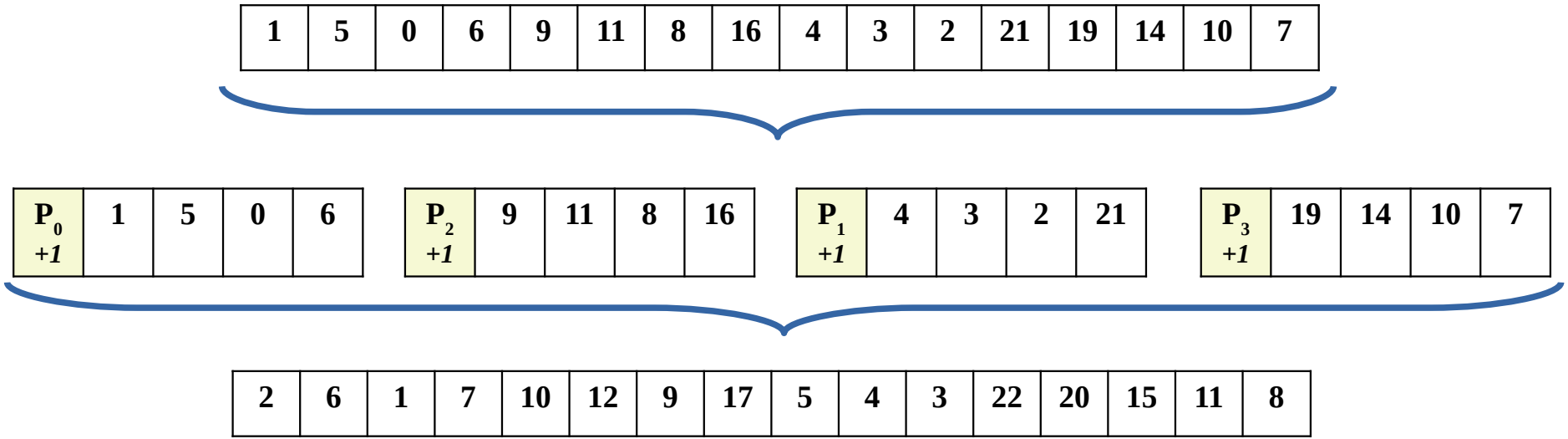
- Los algoritmos que utilizan este paradigma son fáciles de programar y suelen ser altamente escalables.

- El trabajo puede realizarse en fases que consisten en interacciones para sincronizar u obtener datos nuevos para las tareas.

- Se aplica a:
  - Estructuras regulares y estáticas
  - Poca dependencia de datos o ninguna (**Embarrassing Parallel Problem**)
  - Leve dependencia de datos sobre problemas de frontera (Ej: imágenes)
  - Simulación

Paradigmas de programación paralela
Paralelismo de datos - SIMD/SPMD

- **Embarrassing Parallel Problem:** sumar 1 a todos los elementos de un vector.



Paradigmas de programación paralela
Paralelismo de datos - SIMD/SPMD

- Los modelos **SIMD** (Single Instruction Multiple Data) y **SPMD** (Single Program Multiple Data) utilizan un número fijo de procesos o hilos que ejecutan el mismo programa sobre datos diferentes:

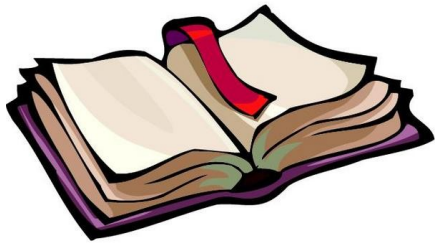
  - **SIMD:** los procesos o hilos ejecutan sincrónicamente **una única instrucción** del mismo programa sobre datos diferentes (GPUs y unidades vectoriales como MMX, SSE y AVX)

  - **SPMD:** los procesos o hilos ejecutan asincrónicamente las **instrucciones del mismo programa** a distintas velocidades sobre datos diferentes (podría ser debido a velocidades diferentes de las unidades de procesamiento o delays en el acceso a los datos)

Agenda

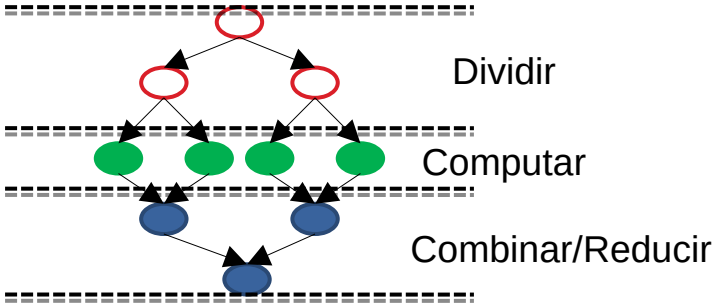
I. Paradigmas de programación paralela
   i. Paralelismo de datos - SIMD/SPMD
   ii. Divide y vencerás
   iii. Pipelines
   iv. Algoritmos sistólicos
   v. Master-Worker
   vi. Task pools

II. Modelos híbridos



Paradigmas de programación paralela
Divide y vencerás

- Se realizan tres etapas:
  - Dividir
  - Computar
  - Combinar/Reducir



- El problema se divide recursivamente en subproblemas.
- Se sigue un estructura tipo árbol, y los procesos o hilos realizan el cómputo en las hojas.
- Cada subproblema se soluciona independientemente y sus resultados se combinan o reducen para producir el resultado final:
  - Se **combinan** cuando de un conjunto de N datos de entrada se obtiene un conjunto de N datos de salida (por ejemplo: Ordenación)
  - Se **reducen** cuando de un conjunto de N datos de entrada se obtiene un único dato de salida (Por ejemplo: Calcular el mínimo en un arreglo)

Paradigmas de programación paralela
Divide y vencerás

- La descomposición recursiva del problema se puede realizar hasta:
  - Que no pueda subdividirse más. No siempre es posible, los recursos son "escasos"
  - Una descomposición intermedia. Que se adapte mejor a los recursos disponibles

¿Es posible lograr el **máximo paralelismo** para una descomposición que arroje en las hojas "mil millones" de cómputos independientes?

¿"mil millones" de procesos o hilos y de unidades de procesamiento para lograr paralalelismo óptimo?

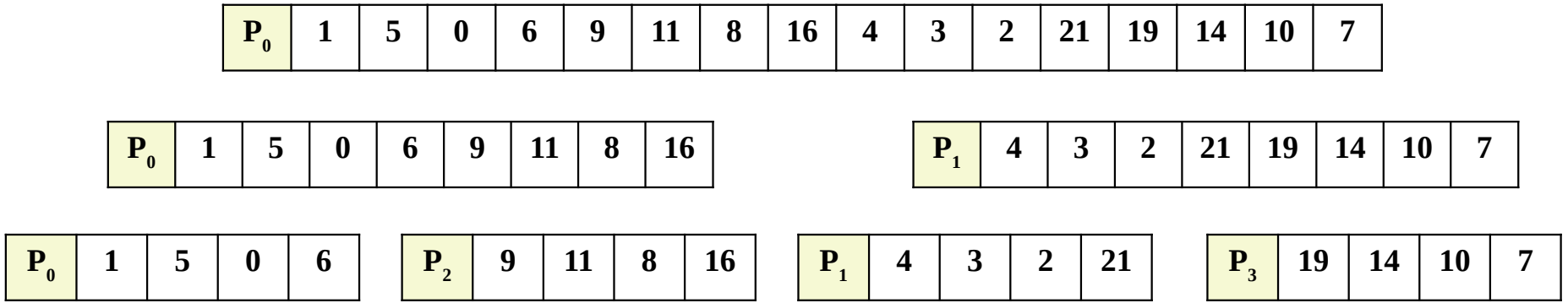
No siempre se tienen los recursos para lograr un paralalelismo óptimo

Paradigmas de programación paralela
Divide y vencerás

- En este paradigma la idea es crear procesos o hilos dinámicamente en función de los recursos.
- Si la recursión es muy grande podría incrementar el costo de la comunicación.
- Es muy común cuando se tienen estructuras híbridas (Ej: MPI-OpenMP)
- Se aplica en algoritmos por naturaleza recursivos:
  - Ordenación
  - Búsquedas
  - Map-Reduce

Paradigmas de programación paralela
Divide y vencerás

- Suponer un sistema con cuatro unidades de procesamiento utilizadas para obtener el mínimo valor de un vector.
- **Etapa dividir:** Suponer que inicialmente un proceso P_0 tiene todos los datos. P_0 descompone los datos del problema en dos partes, una parte se la queda y la otra la "comunica" al proceso P_1. Esto se vuelve a repetir recursivamente hasta un punto en que se puedan aprovechar los recursos de la mejor manera.



Paradigmas de programación paralela
Divide y vencerás

- **Etapa computar:** cada proceso calcula el mínimo de la porción de vector que le corresponde.

P_0	1	5	0	6

P_2	9	11	8	16

P_1	4	3	2	21

P_3	19	14	10	7

- **Etapa combinar/reducir:** el proceso P_2 "comunica" el mínimo valor de su porción al proceso P_0. Lo mismo hace el proceso P_3 al proceso P_1.
  - Los procesos P_0 y P_1 se encargan de “combinar/reducir" los resultados recibidos con los que ya tienen.
  - Por último el proceso P_1 "comunica" su resultado al proceso P_0 el cual se encarga de “combinar/reducir" el resultado final.

P_0	0	8

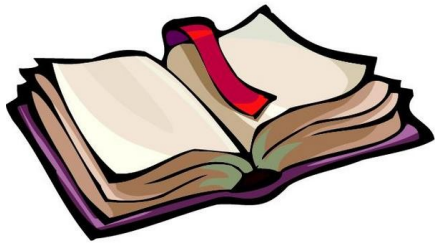
P_0	0	2

P_1	2	7

Agenda

I. Paradigmas de programación paralela
   i. Paralelismo de datos - SIMD/SPMD
   ii. Divide y vencerás
   iii. Pipelines
   iv. Algoritmos sistólicos
   v. Master-Worker
   vi. Task pools

II. Modelos híbridos



Paradigmas de programación paralela
Pipelines

- Suponer que el cómputo completo involucra el cálculo sobre muchos conjuntos de datos y puede ser visto en términos de un flujo de datos a través de varias etapas.
- Similar a una linea de ensamblaje/montaje donde en cada etapa se realiza una acción diferente sobre los datos.
- La aplicación se subdivide en subproblemas y cada subproblema se debe completar para comenzar el siguiente.
- El pipeline puede ser:

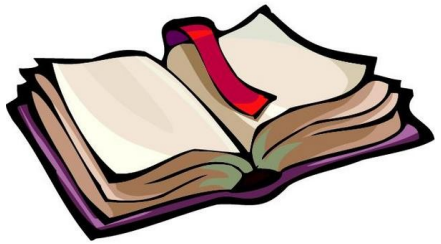
Paradigmas de programación paralela
Pipelines

- Este paradigma es común en la descomposición funcional donde se realizan operaciones diferentes en cada unidad de procesamiento.
- El máximo paralelismo se alcanza cuando el pipeline se llena, mientras tanto puede haber muchas unidades de procesamiento ociosas.
- El **grado de concurrencia** está **limitado** por la cantidad de etapas: un gran número de etapas permite mayor concurrencia pero incrementa el costo de la comunicación. El volumen de trabajo en una etapa debería ser suficientemente grande comparado al tiempo de comunicación.
- Se aplica a:
  - Paralelismo a nivel de instrucción (ILP)
  - Procesamiento gráficos (Pipeline gráfico)
  - Procesamiento de señales

Agenda

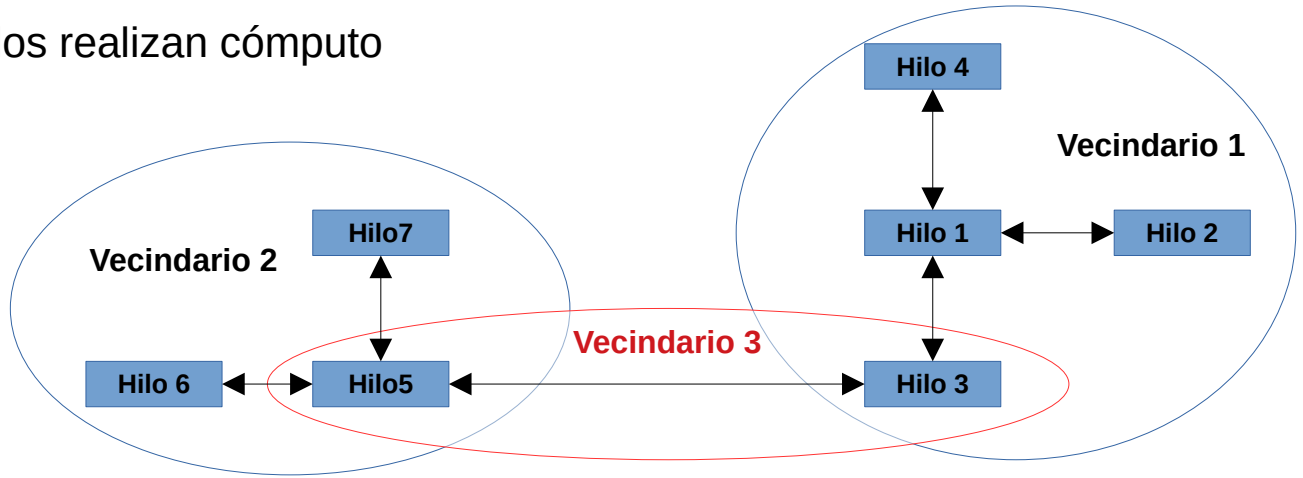
I. Paradigmas de programación paralela
   i. Paralelismo de datos - SIMD/SPMD
   ii. Divide y vencerás
   iii. Pipelines
   iv. Algoritmos sistólicos
   v. Master-Worker
   vi. Task pools

II. Modelos híbridos



Paradigmas de programación paralela
Algoritmos sistólicos

- Los algoritmos sistólicos son un caso particular del paradigma pipelines.
- Los procesos o hilos cooperan sincrónicamente con un conjunto regular de procesos o hilos vecinos más cercanos.
- Existen dos etapas que se repiten hasta que en conjunto todos resuelven el problema:
  - Todos los procesos o hilos comunican
  - Todos los procesos o hilos realizan cómputo



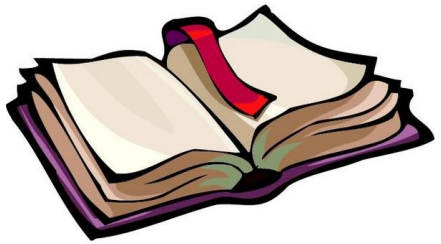
Paradigmas de programación paralela
Algoritmos sistólicos

- Este paradigma es útil para muchas aplicaciones iterativas de paralelismo de datos donde los datos tienen cierta dependencia.
- Se utiliza cuando los datos se encuentran divididos entre distintos procesos o hilos y cada uno es responsable de actualizar una parte particular. El valor de un nuevo dato depende de los valores de los datos de vecinos cercanos.
- Se aplica:
  - Aplicado por primera vez por Kung y Leiserson para multiplicar matrices
  - Procesamiento de imágenes
  - Ecuaciones diferenciales
  - Simulaciones

Agenda

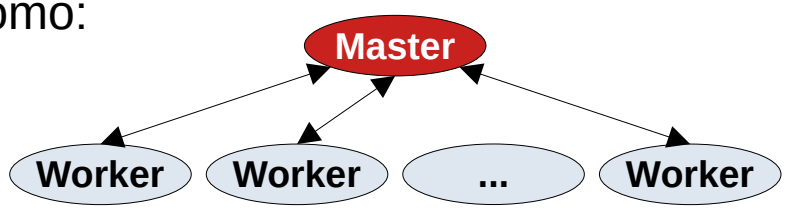
I. Paradigmas de programación paralela
   i. Paralelismo de datos - SIMD/SPMD
   ii. Divide y vencerás
   iii. Pipelines
   iv. Algoritmos sistólicos
   v. Master-Worker
   vi. Task pools

II. Modelos híbridos



Paradigmas de programación paralela
Master-Worker

- El paradigma **Master-Worker** también es conocido como:
  - Task-farming
  - Master-slave



- En este paradigma existen dos tipos de procesos o hilos:

Master	Uno o varios Workers
• Controla la ejecución del programa • Ejecuta la función principal de un programa paralelo • Puede encargarse de crear varios o todos los workers • Se encarga de descomponer el problema en tareas • Distribuye las tareas a los diferentes workers de forma estática o por demanda. Eventualmente, puede tener rol de Worker y asignarse tarea. • Recolecta los resultados parciales para procesar el resultado final	• Realizan trabajo • Pueden ser creados estática o dinámicamente por el Master • Solicitan o reciben tareas del Master (Por demanda) • Eventualmente podrían generar más tareas (aplicaciones con datos generados dinámicamente) • Envían los resultados al Master

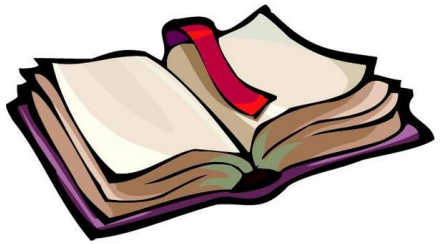
Paradigmas de programación paralela
Master-Worker

- Este paradigma se utiliza cuando el diseño está dominado por la necesidad de **balancear la carga dinámicamente**. La carga queda balanceada automáticamente.
- Los algoritmos que utilizan este paradigma tiene una buena escalabilidad cuando las tareas a realizar superan ampliamente el número de workers y el costo de cada tarea no es variable.
- Es sensible al overhead por comunicación o sincronización (**cuellos de botella**).
- Se aplica a:
  - Carga de trabajo asociadas a procesos o hilos que son variables e impredecibles. Realizar una distribución de carga estática puede llevar a desbalances de carga que afectan el rendimiento
  - Grandes volúmenes de datos centralizados

Agenda

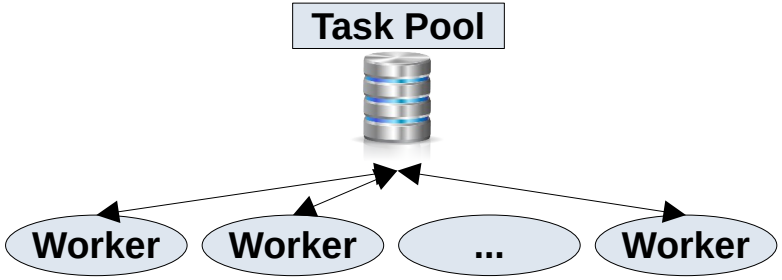
I. Paradigmas de programación paralela
   i. Paralelismo de datos - SIMD/SPMD
   ii. Divide y vencerás
   iii. Pipelines
   iv. Algoritmos sistólicos
   v. Master-Worker
   vi. Task pools

II. Modelos híbridos



Paradigmas de programación paralela
Task pools

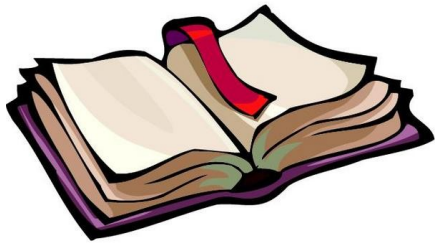
- **Task pools** o **Work pools** es similar al paradigma Master-Worker.
- Los workers podrían prescindir de un hilo Master y acceder directamente a un repositorio o estructura de datos (**pool - bag of tasks**) donde se almacena el trabajo a realizar.
- Eventualmente los workers podrían generar más trabajo que depositarían en el pool.
- Se deben considerar los accesos concurrentes al pool para evitar condiciones de carrera.



Agenda

I. Paradigmas de programación paralela
   i. Paralelismo de datos - SIMD/SPMD
   ii. Divide y vencerás
   iii. Pipelines
   iv. Algoritmos sistólicos
   v. Master-Worker
   vi. Task pools

II. Modelos híbridos



Paradigmas de programación paralela
Modelos híbridos

- En algunos casos, se puede aplicar más de un paradigma a un problema específico, lo que resulta en un modelo de algoritmo híbrido.
- Un modelo híbrido puede estar compuesto por múltiples paradigmas aplicados jerárquicamente o múltiples paradigmas aplicados secuencialmente a diferentes fases de un algoritmo paralelo.
- Por ejemplo, los datos pueden fluir por un pipeline y dentro de cada nodo del pipeline puede haber paralelismo de datos: