

Tema 2: Modelos de programación paralela

Un resumen rápido...

Modelo de programación

¿Qué es?

- Abstracción del HW para que el programador **exprese** su programa paralelo.

Ejemplos principales:

- Memoria compartida: OpenMP (SPMD), pthread (MIMD), OpenCL (SPMD), ...
- Paso de mensajes: MPI (MIMD)
- Paralelización de datos: Hadoop, Spark

Modelo de ejecución

¿Qué es?

- Abstracción de cómo una arquitectura HW **ejecuta** el código.
- Permite analizar el rendimiento de los programas en los diferentes tipos de arquitectura.
- Ayuda a entender como el código (modelo de programación) se particulariza/implementa en cada tipo de HW.

Ejemplos principales:

- SMP (e.g. procesador multinúcleo)
- NUMA (e.g. servidor multiprocesador)
- MPP (e.g. Cluster)
- SIMT (e.g. warps en una GPU)
- SIMD (e.g. procesadores vectoriales)

Modelo de programación/ejecución

Ojo: no todos los modelos de programación nos permiten ejecutar en todo HW.

Modelo de programación	Modelo de ejecución
Paso de mensajes (MPMD/MIMD)	MPI en un MPP (e.g. HPC cluster)
<pre>if (id == 0) send_msg (P1, b[4..7], c[4..7]); else recv_msg (P0, b[4..7], c[4..7]); for (i=start_iter; i<end_iter; i++) a[i] = b[i] + c[i]; local_sum = 0; for (i=start_iter; i<end_iter; i++) local_sum = local_sum + a[i]; if (id == 0) { recv_msg (P1, &local_sum1); sum = local_sum + local_sum1; Print sum; } else send_msg (P0, local_sum);</pre>	MPI en NUMA (e.g. procesos que se intercomunican mediante una interconexión en placa)
	MPI en SMP (e.g. procesos que se comunican mediante la red de loopback o mediante la memoria compartida)
	GPU (SIMT)

Algunos modelos de programación no tendrían un modelo de ejecución asociado.

Modelo de programación/ejecución

Ojo: no todos los modelos de programación nos permiten ejecutar en todo HW.

Modelo de programación	Modelo de ejecución
OpenCL (SPMD, memoria compartida)	MPP (e.g. HPC cluster)
<pre>// OpenCL kernel. Each work item takes care of one element of c __kernel void vecAdd(__global double *a, __global double *b, __global double *c, const unsigned int n) { //Get our global thread ID int id = get_global_id(0); //Make sure we do not go out of bounds if (id < n) c[id] = a[id] + b[id]; }</pre>	NUMA (e.g. threads)
	SMP (e.g. threads)
	GPU (e.g. threads que forman warps, SIMT)
	**Procesador vectorial (SIMD)

Algunos modelos de programación no tendrían un modelo de ejecución asociado.

Modelo de programación/ejecución

Ojo: no todos los modelos de programación nos permiten ejecutar en todo HW.

Modelo de programación	Modelo de ejecución
MapReduce (SPMD, memoria distribuida)	MPP (e.g. Big Data cluster)
<pre>text_file = sc.textFile("hdfs://...") counts = text_file.flatMap(lambda line: line.split(" ")) \ .map(lambda word: (word, 1)) \ .reduceByKey(lambda a, b: a + b) counts.saveAsTextFile("hdfs://...")</pre>	NUMA (e.g. multiples procesos worker)
	SMP (e.g. multiples procesos worker)
	GPU (e.g. threads que forman warps, SIMT)
	Procesador vectorial (SIMD)

Algunos modelos de programación no tendrían un modelo de ejecución asociado.

Modelo de programación/ejecución

Ojo: no todos los modelos de programación nos permiten ejecutar en todo HW.

Modelo de programación	Modelo de ejecución
Instrucciones vectoriales (SIMD)	MPP
<div>LV V1, R1 (A) LV V2, R1 (B) ADDV V3, V1, V2 SV V3, R1 (C)</div>	NUMA (e.g. inst. vectoriales, strip mining)
	SMP (e.g. inst. vectoriales, strip mining)
	GPU (e.g. threads que forman warps, SIMT)
	Procesador vectorial (SIMD)

Algunos modelos de programación no tendrían un modelo de ejecución asociado.

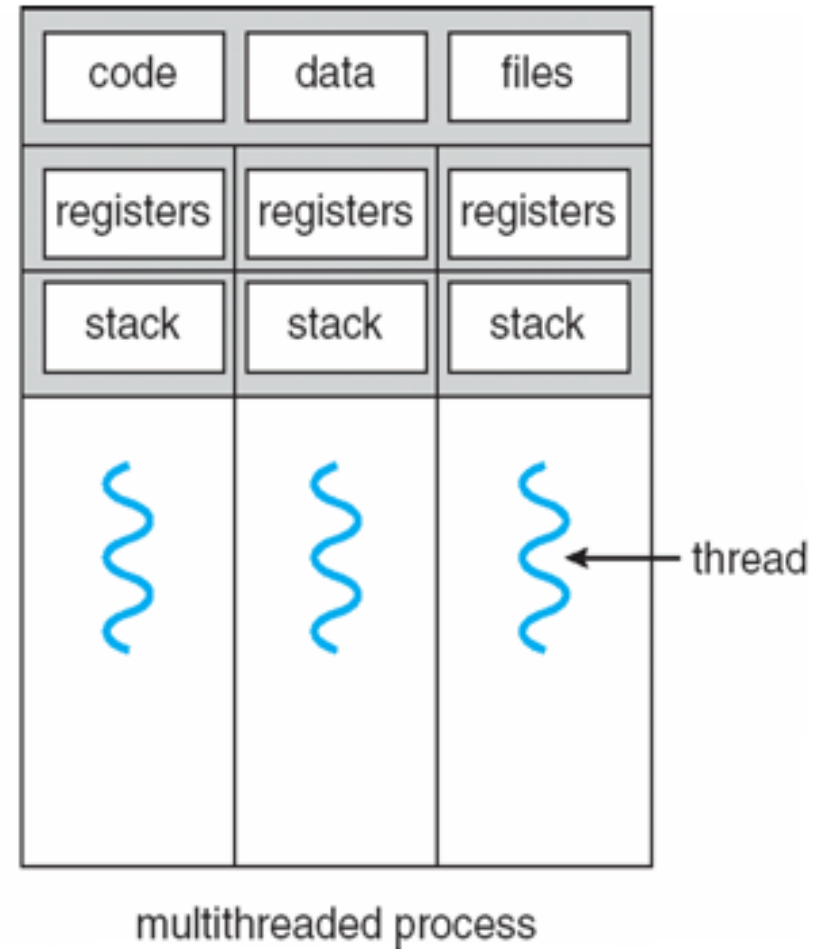
Hilo vs Proceso

Thread (lo esencial para ejecutar)

- Registros, contador de programa, estado de ejecución.
- Pila.

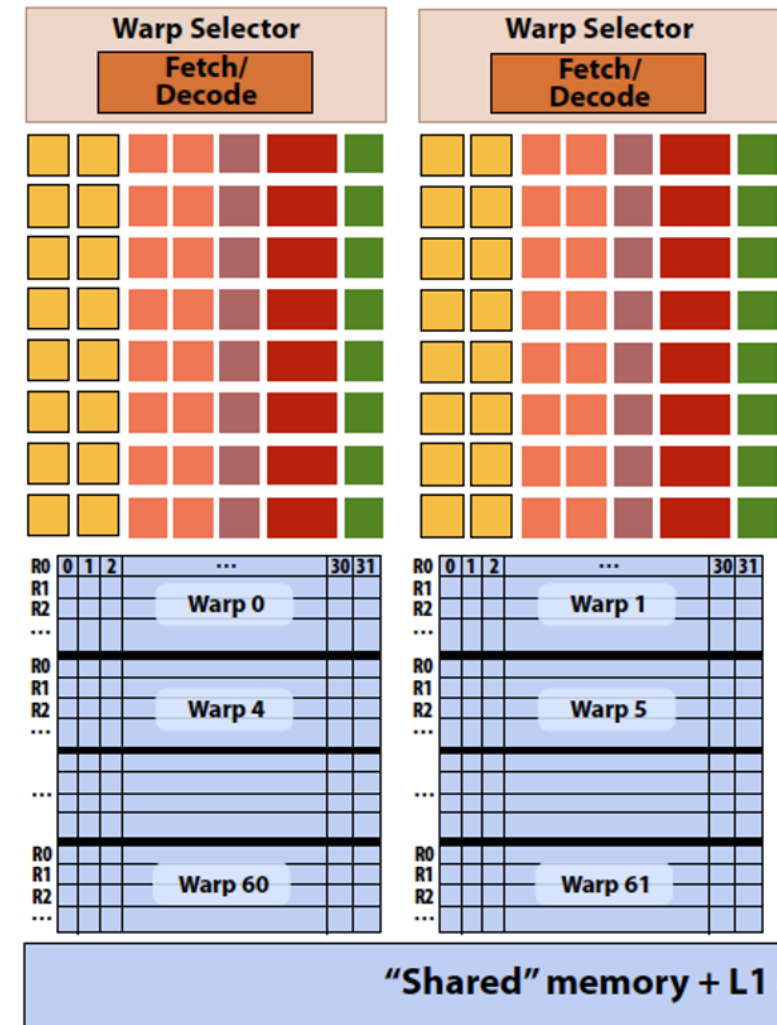
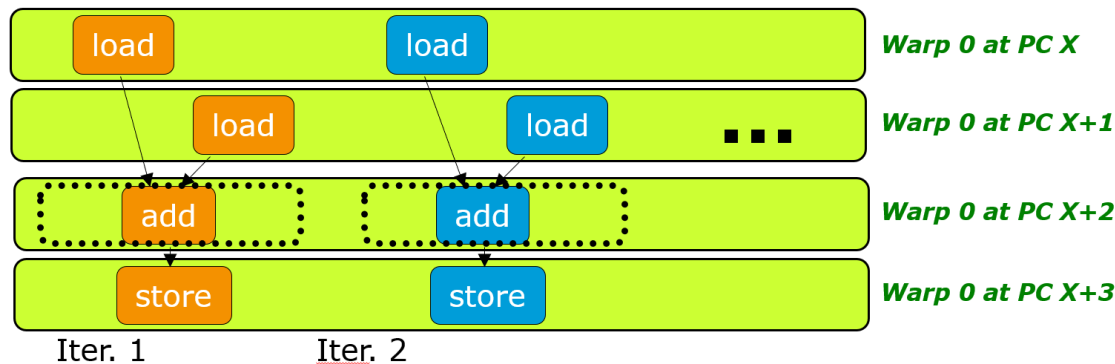
Proceso (administración)

- Espacio de memoria.
- Ficheros abiertos.
- Alarmas, Señales, etc.



SPMD en SIMT

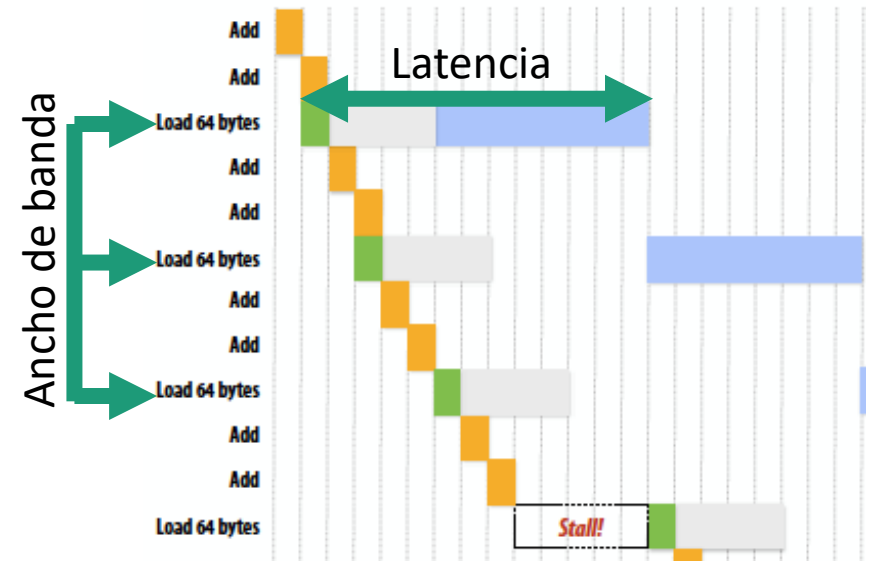
Warp: conjunto de hilos que comparten PC.



Memoria y comunicaciones

La memoria y las comunicaciones van a ser factores limitantes en el paralelismo.

- El ancho de banda nos limita el número de operaciones de memoria o mensajes concurrentes que se pueden lanzar.
- La latencia introduce tiempos adicionales de procesamiento, pero si es un pipeline, suele solo significar un retardo al inicio.



Resumen modelos programación

Aspect	Shared Memory	Message Passing	MapReduce
Communication	Implicit (via loads/stores)	Explicit Messages	Limited and Implicit
Synchronization	Explicit	Implicit (via messages)	Immutable (K, V) Pairs
Hardware Support	Typically Required	None	None
Development Effort	Lower	Higher	Lowest
Tuning Effort	Higher	Lower	Lowest

Tema 2: Paralelización de bucles

Un resumen rápido...

Grafo de dependencias

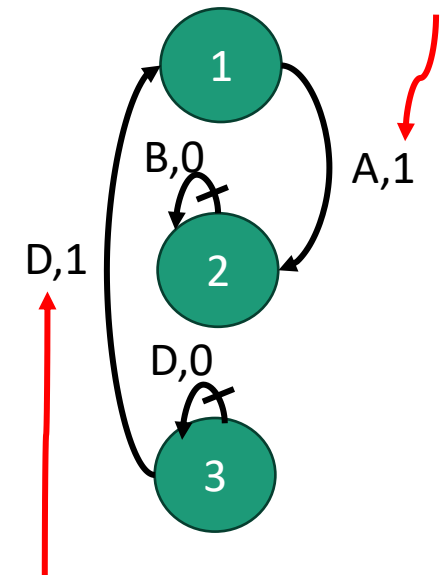
El grafo de dependencias (reducido) resume las dependencias de las instrucciones de un bucle

```
// ejercicio 2.5
do i = 1, 994
  (1) A(i) = D(i) - 10
  (2) B(i+1) = B(i+1) * A(i-1)
  (3) D(i+1) = D(i+1) + D(i+1)
Enddo
```

Dependiendo el origen y el destino:

- RAW: origen es escritura y destino es lectura.
- WAR: origen es lectura y destino es escritura.
- WAW: ambos son escrituras.

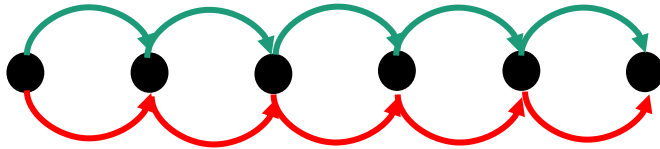
$$[\text{Origen}] - [\text{Destino}] = [i] - [i-1] = 1$$



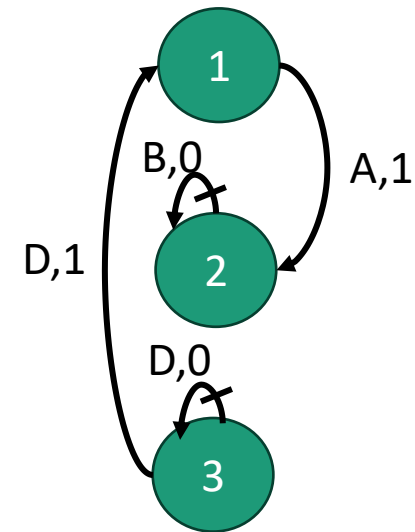
$$[\text{Origen}] - [\text{Destino}] = [i+1] - [i] = 1$$

Espacio de iteraciones

Expresa como las iteraciones dependen unas de otras.



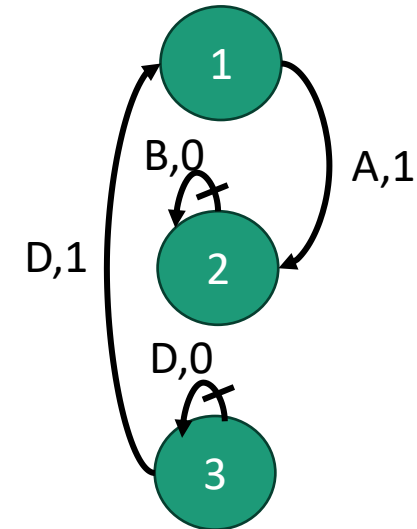
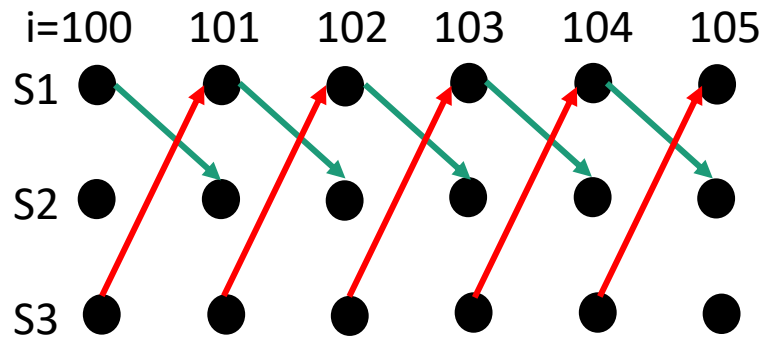
```
// ejercicio 2.5
do i = 1, 994
  (1) A(i) = D(i) - 10
  (2) B(i+1) = B(i+1) * A(i-1)
  (3) D(i+1) = D(i+1) + D(i+1)
Enddo
```



Espacio de iteraciones

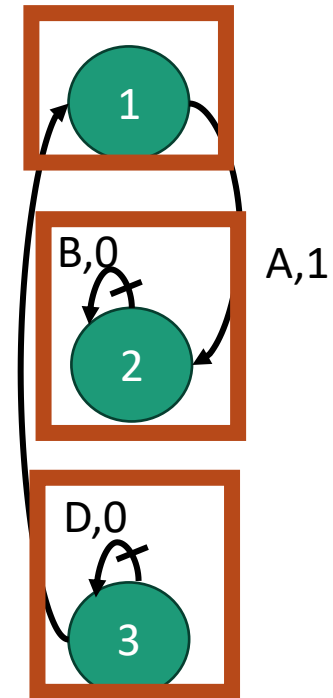
Normalmente nos interesa más el grafo completo de dependencias, que es un desarrollo del espacio de iteraciones para cada sentencia del programa.

Ojo: a veces abusamos del lenguaje y llamamos a esto también espacio de dependencias.



Paralelización automática (A-K)

1. Dividimos en **componentes fuertemente conexas**.
2. Planificar el DAG resultante (C3->C1->C2)
3. Determinar el tipo de cluster:
 - Si alguna dependencia intraccluster es de distancia >0, es serie.
 - Si todas las dependencias intraccluster son de distancia 0 o no hay dependencias, es paralelo.
4. Escribir el código



```
doall i=1,994
(3)D(i+1)=D(i+1)+D(i+1)
barrier
doall i=1,994
(1)A(i)=D(i)-10
barrier
doall i=1,994
(2)B(i+1)=B(i+1)*A(i-1)
```


Evitando barreras

Dadas dos componentes fuertemente conexas C1 y C2 que se ejecutan una detrás de la otra:

- Si **C1 no depende de C2**, la barrera **no es necesaria (C1 y C3 se pueden ejecutar concurrentemente)**.
- Si **todas las dependencias entre C1 y C3 son de distancia 0**, hay dos opciones:
 - Mantener la barrera.
 - Fusionar los bucles.
- Si hay una **dependencia** entre C1 y C3 y **distancia > 0**, **mantenemos la barrera (no se pueden fusionar los bucles)**.
 - Habría que probar a hacer *peeling*.

Peeling

Consiste en alinear las dependencias para hacerlas de distancia 0

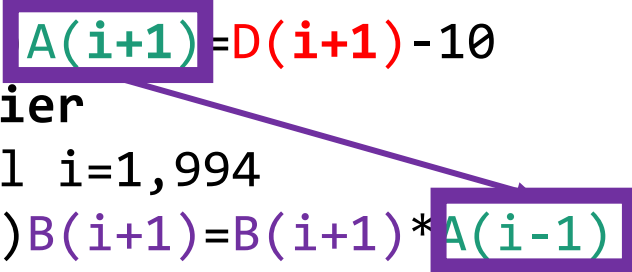
```
doall i=1,994
  (3) D(i+1)=D(i+1)+D(i+1)
barrier
doall i=1,994
  (1) A(i)=D(i)-10
barrier
doall i=1,994
  (2) B(i+1)=B(i+1)*A(i-1)
```

```
doall i=1,994
  (3) D(i+1)=D(i+1)+D(i+1)
barrier
doall i=0,993
  (1) A(i+1)=D(i+1)-10
barrier
doall i=1,994
  (2) B(i+1)=B(i+1)*A(i-1)
```

Peeling

Consiste en alinear las dependencias para hacerlas de distancia 0

```
doall i=1,994
  (3)  $D(i+1) = D(i+1) + D(i+1)$ 
barrier
doall i=0,993
  (1)  $A(i+1) = D(i+1) - 10$ 
barrier
doall i=1,994
  (2)  $B(i+1) = B(i+1) * A(i-1)$ 
```



```
doall i=1,994
  (3)  $D(i+1) = D(i+1) + D(i+1)$ 
barrier
doall i=0,993
  (1)  $A(i+1) = D(i+1) - 10$ 
barrier
doall i=-1,992
  (2)  $B(i+3) = B(i+3) * A(i+1)$ 
```

Peeling

Consiste en alinear las dependencias para hacerlas de distancia 0.

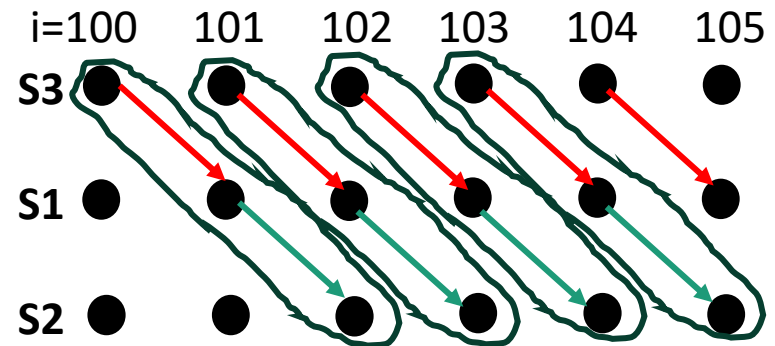
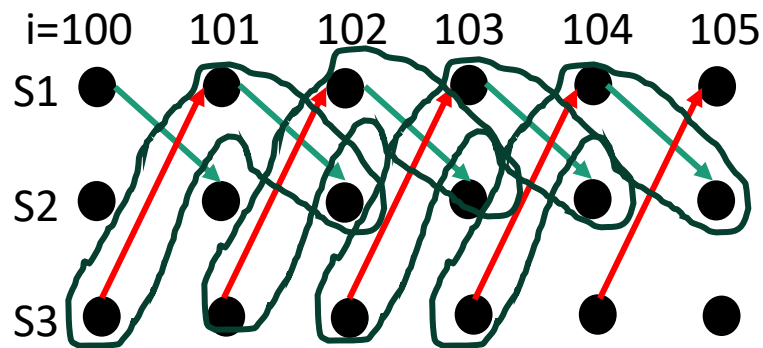
```
doall i=1,994
  (3)D(i+1)=D(i+1)+D(i+1)
barrier
doall i=0,993
  (1)A(i+1)=D(i+1)-10
barrier
doall i=-1,992
  (2)B(i+3)=B(i+3)*A(i+1)
```

```
A(1)=..., B(2)=..., B(3)=...
doall i=1,992
  (3)D(i+1)=D(i+1)+D(i+1)
  (1)A(i+1)=D(i+1)-10
  (2)B(i+3)=B(i+3)*A(i+1)
D(994)=..., D(995)=..., A(994)=...,
```

Peeling

A veces este se puede ver desde el grafo de dependencias completo. Aunque esto se suele ver mucho mejor ordenando según el algoritmo A-K primero...

```
A(1)=..., B(2)=..., B(3)=...  
doall i=1,992  
  (3) D(i+1)=D(i+1)+D(i+1)  
  (1) A(i+1)=D(i+1)-10  
  (2) B(i+3)=B(i+3)*A(i+1)  
D(994)=..., D(995)=..., A(994)=...
```



Planificadores

Planificador en paralelización de bucles: manera de repartir las iteraciones de un bucle paralelo.

- Estático: los batchs de tareas se reparten antes de ejecutar el bucle.
 - Óptimo cuando las tareas tardan lo mismo. La planificación se podría hacer en muchos casos en compilación.
 - Consecutivo: 0 0 0 0 1 1 1 1 2 2 2 2 3 3 3 3
 - Entrelazado: 0 1 2 3 0 1 2 3 0 1 2 3 0 1 2 3
- Dinámico: las tareas se reparten en ejecución. Según cada procesador queda libre, pide un nuevo batch de tareas.
 - Agrega un cierto overhead de ejecutar el planificador.