Metodología de la Programación Paralela

Facultad Informática, Universidad de Murcia

Esquemas algorítmicos paralelos:

Paralelismo Relajado
Paralelismo Síncrono

Sesión 2

- Descomposición del trabajo:
  Paralelismo de datos.
  Particionado de datos.
  Algoritmos relajados.
- De paralelismo basado en dependencias de datos:
  Paralelismo síncrono.
  Dependencias en árbol o grafo.
  Pipeline.
- De paralelización de esquemas secuenciales:
  Divide y Vencerás.
  Programación Dinámica.
  Recorridos de árboles: Backtracking y Branch and Bound.
- De múltiples tareas o trabajadores:
  Bolsa de tareas. Granja de procesos. Maestro-Esclavo.

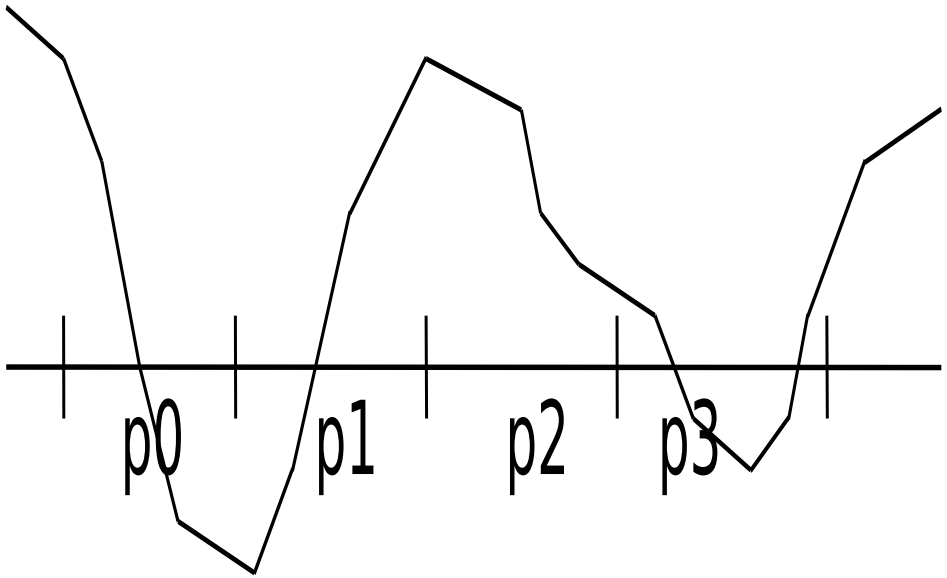
Contenido

Ideas generales

- Cada elemento de proceso trabaja de manera independiente.
- No hay sincronización ni comunicación, salvo las de distribuir datos y recoger resultados.
- Buenas prestaciones en Memoria Compartida y Paso de Mensajes.
- A veces a costa de no utilizar el mejor algoritmo paralelo.
- Fáciles de programar.
- Difícil encontrar algoritmos que se adecuen estrictamente al esquema.

Ejemplo - búsqueda de raíces de una ecuación

Se divide el espacio de búsqueda en *p* subespacios:



- La programación es muy sencilla.
- Puede haber desbalanceo.
  ¿Cómo reducirlo?

¿Otros ejemplos?

Algunos de los vistos se pueden considerar de **paralelismo relajado**:

¿Otros ejemplos?

Algunos de los vistos se pueden considerar de **paralelismo relajado**:

- ¿Ordenación por rango?

¿Otros ejemplos?

Algunos de los vistos se pueden considerar de **paralelismo relajado**:

- ¿Ordenación por rango?
- ¿Suma de matrices?

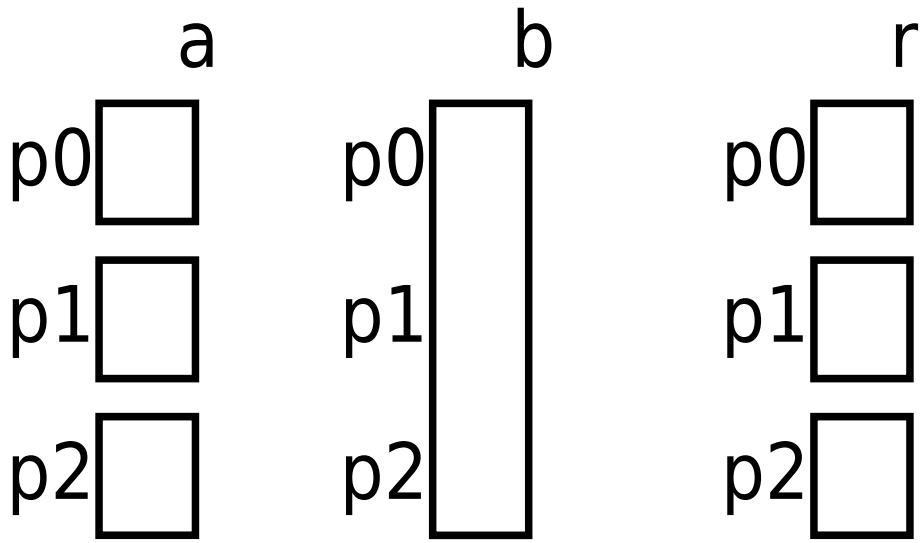
¿Otros ejemplos?

Algunos de los vistos se pueden considerar de **paralelismo relajado**:

- ¿Ordenación por rango?
- ¿Suma de matrices?
- ¿Multiplicación de matrices?

Ejemplo - ordenación por rango

- Memoria Compartida: cada hilo calcula el rango de una parte de los elementos.
- Paso de Mensajes. Primero distribuir en la forma

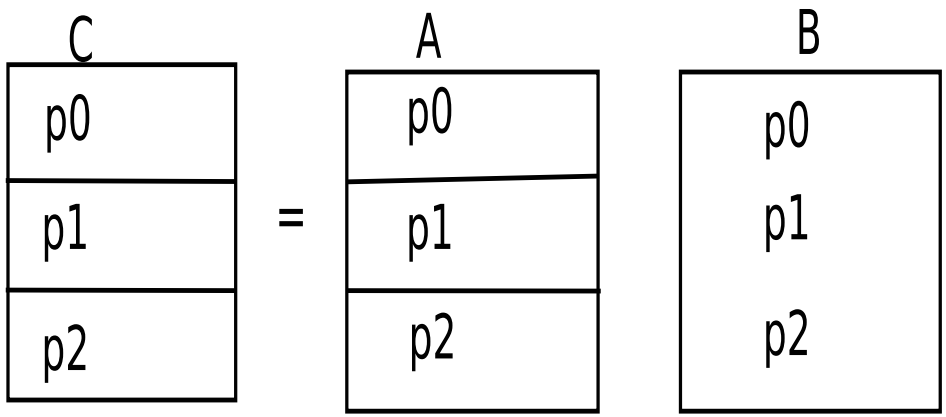


y cada proceso calcula el rango de una parte de los elementos.

- Hay duplicación de datos
- pero se simplifica la programación
- y se obtienen buenas prestaciones.

Ejemplo - multiplicación de matrices

- Memoria Compartida: cada hilo calcula un bloque de filas de la matriz resultado.
- Paso de Mensajes, con distribución:



cada procesador calcula las filas de *C* correspondientes a las filas de *A* que contiene.

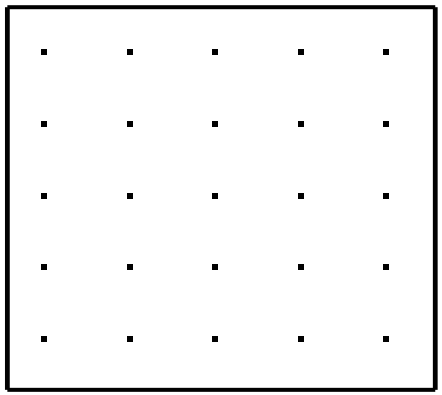
- No es necesaria sincronización ni comunicación
- salvo la distribución y acumulación (¿qué funciones se utilizarían?),
- pero es más costoso el envío inicial al repetirse *B* en cada proceso.

Contenido

Ideas generales

- Se realizan iteraciones sucesivas:
  - Cada elemento de proceso realiza el mismo trabajo sobre una porción distinta de los datos.
  - Datos de una iteración se utilizan en la siguiente.
  - Tras cada iteración hay sincronización (local o global).
- Las prestaciones están afectadas por la sincronización:
  - En Memoria Compartida buenas prestaciones.
  - En Paso de Mensajes bajan prestaciones por el coste de las comunicaciones.

Ejemplo - iteración de Jacobi



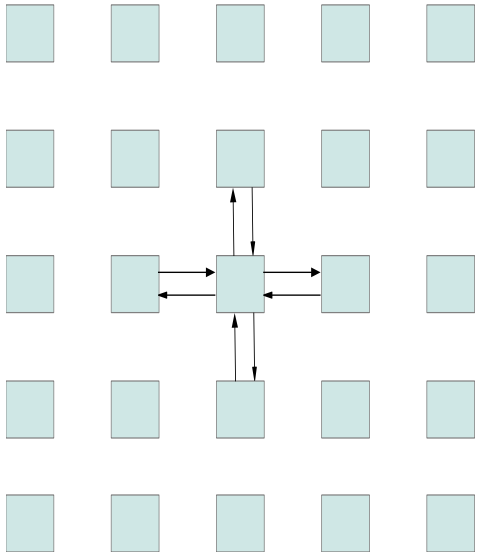
valores fijos
en la frontera

- Se calcula la ecuación de diferencias:

$$V(i,j) = \frac{V(i-1,j) + V(i+1,j) + V(i,j-1) + V(i,j+1)}{4}$$

- Converge gradualmente a una solución cada vez más precisa.
- Para obtener una solución más precisa aumentar el número de puntos.
- Una iteración tras otra secuencialmente, pero dentro de cada iteración paralelismo.
- Entre iteraciones se necesita comunicación local.
- Para comprobar condición de fin comunicación global.

Ejemplo - iteración de Jacobi, descomposición de los datos



Se pueden agrupar elementos de formas distintas:

- Por bloques de filas.
- Cíclico por filas.
- Por bloques.
- Cíclico por bloques.
- ...

¿Cuál consideramos mejor?

Ejemplo - iteración de Jacobi, Memoria Compartida

```
for(i=0;i<numiter;i++) {
    #pragma omp parallel for private(j,k)
    for(j=0;j<n;j++)
        for(k=0;k<n;k++)
            b[j,k]=(a[j-1,k]+a[j+1,k]+a[j,k-1]+a[j,k+1])/4;
    a=b;
}
```

- Sincronización al acabar el `pragma`.
- Asigna filas completas a cada hilo, ¿topología lógica?
- ¿Y si el número de iteraciones no es fijo?

Ejemplo - iteración de Jacobi, Memoria Compartida - Paralelismo explícito

Crear un proceso por cada hilo:

```
    #pragma omp parallel for private(hilo)
    for(hilo=0;hilo<p;hilo++)
        iteracion(a,n,hilo,p);
```

```
#pragma omp parallel private(hilo)
{
    hilo=omp_get_thread_num();
    iteracion(a,n,hilo,p);
}
```

```
iteracion(a,n,hilo,p):
for(i=0;i<numiter;i++) {
    for(j=hilo*n/p;j<(hilo+1)*n/p;j++)
        for(k=0;k<n;k++)
            b[j,k]=(a[j-1,k]+a[j+1,k]+a[j,k-1]+a[j,k+1])/4;
    #pragma omp barrier  //necesaria?
    a=b;
    #pragma omp barrier
}
```

- La barrera implica sincronización
- y con Paso de Mensajes comunicación.
- ¿Coste de la barrera?
- ¿Cómo son las variables?

Ejemplo - iteración de Jacobi, condición de fin

- El número de iteraciones puede no ser fijo,
- puede depender de la norma $\|a_i - a_{i-1}\|$.
- Necesario guardar la información de la iteración anterior
- y calcular la norma tras cada iteración,
- lo que tiene un coste computacional (¿cuánto en comparación con el coste computacional de cada iteración?)
- y es necesario acumular y difundir el resultado,
- lo que implica sincronización y/o comunicación
- similar al trabajo realizado en una barrera global.

Ejemplo - iteración de Jacobi, Paso de Mensajes

Similar a Memoria Compartida con paralelismo explícito:

```
En cada Pi, i=0,...,p-1
distribucion  //de P0 al resto por bloques de filas
iteracion(a,n,i,p);
acumulacion  //de todos en P0 bloques de filas
```

```
iteracion(a,n,nodo,p):
for(i=0;i<numiter;i++) {
    for(j=1;j<=n/p;j++)
        for(k=0;k<n;k++)
            b[j,k]=(a[j-1,k]+a[j+1,k]+a[j,k-1]+a[j,k+1])/4;
    intercambio de filas
    //se envia fila a[1,:] hacia arriba y a[n/p,:] abajo
    //se reciben las filas en a[0,:] y a[n/p+1,:]
}
```

- Cada proceso trabaja con un bloque de $\frac{n}{p}$ filas,
- pero recibe dos más,
- y usa esas posiciones para almacenar la última fila del proceso anterior y la primera del proceso siguiente.
- La sincronización entre iteraciones no es con barrera global.

Ejemplo - iteración de Jacobi, Paso de Mensajes, costes

- De P_0 a cada proceso envío de $\frac{n}{p}+2$ filas: $(p-1)\left(t_s+\left(\frac{n}{p}+2\right)nt_w\right)$
- En cada una de las *iter* iteraciones:
  - Las n^2 actualizaciones distribuidas entre los p procesos: $4\frac{n^2}{p}$
  - En cada proceso central envío y recepción de dos filas: $4\left(t_s+nt_w\right)$

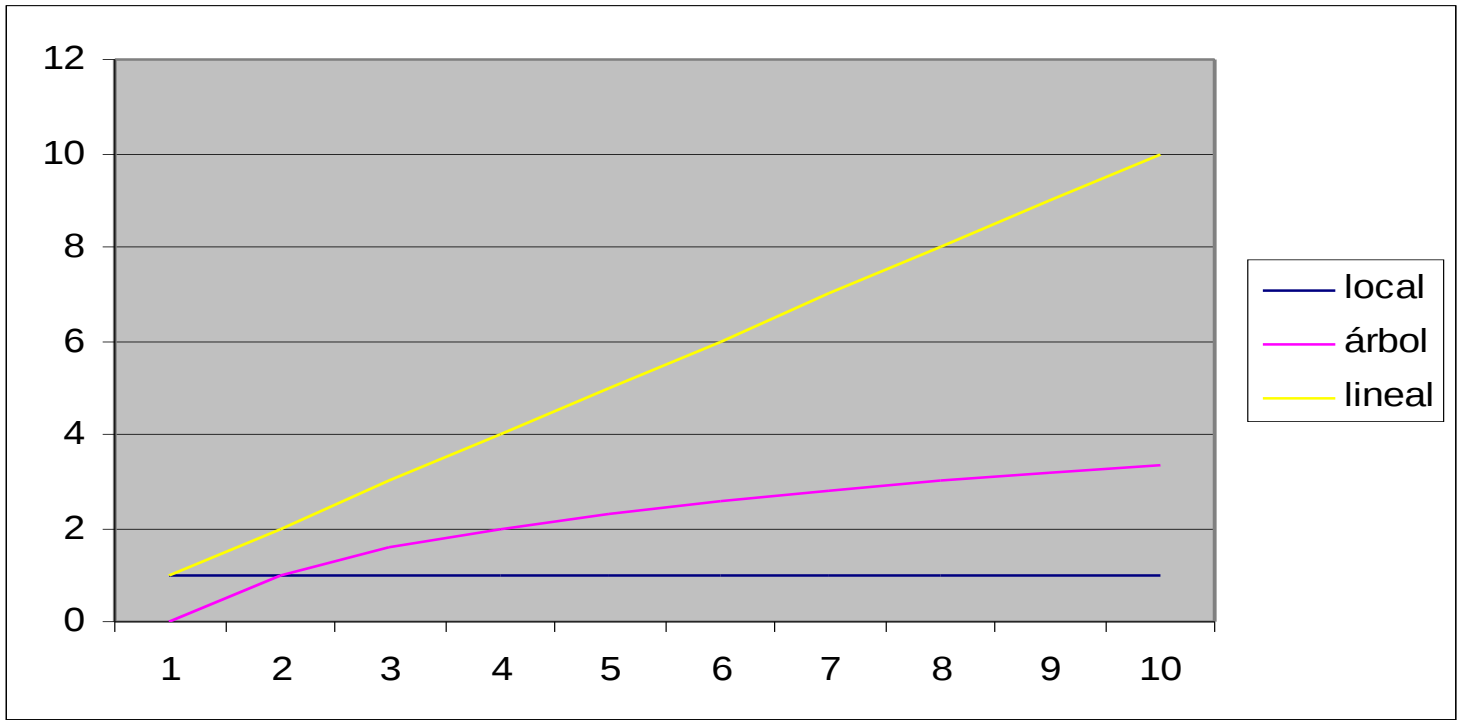
$$iter\left(4\frac{n^2}{p}+4t_s+4nt_w\right)$$

- Acumulación de los bloques en proceso P_0: $(p-1)\left(t_s+\frac{n}{p}nt_w\right)$

Mismo coste de computación que de comunicaciones, $\frac{n^2}{p}$, pero la computación se hace varias iteraciones, y en cada iteración el coste de comunicación es menor que el de computación.
Pero si hay que calcular la condución de fin, es necesaria sincronización global.

Ejemplo - sincronización global

- En la iteración de Jacobi hay puntos de sincronización (entre iteraciones y para evaluar el criterio de convergencia).
- En algunos casos la sincronización debe ser global (criterio de convergencia) y en otros puede ser local (comunicación de filas entre iteraciones).
- La global se puede hacer:
  - Con acceso a variable global.
  - En árbol.
- Tienen costes distintos:



Ejemplo - sincronización con barrera lineal

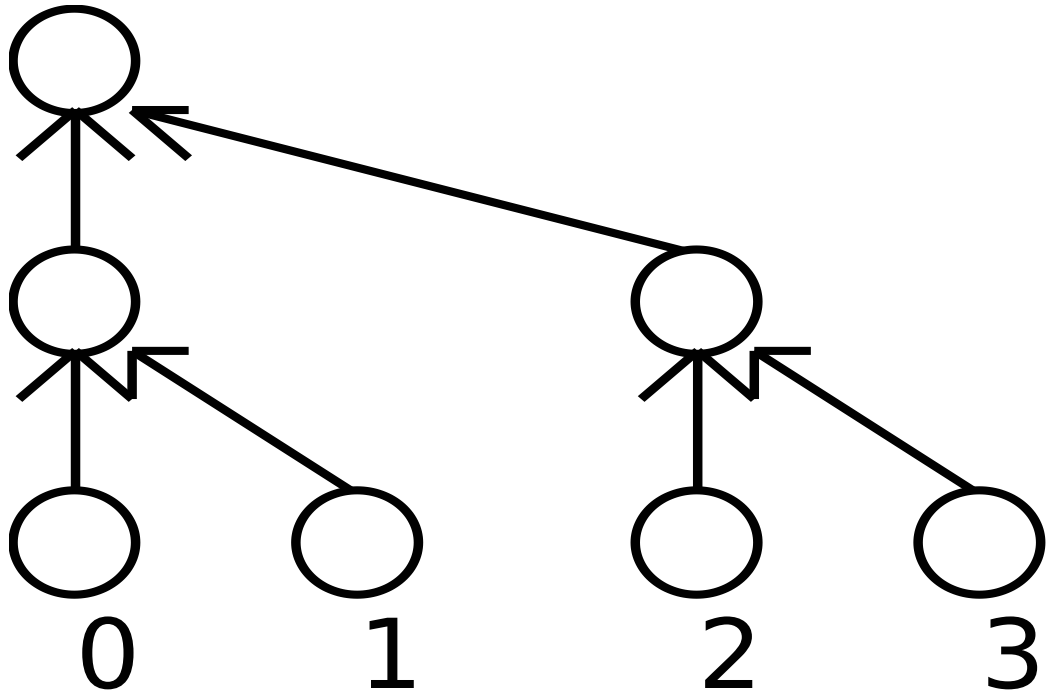
- Normalmente las barreras se proporcionan con el sistema.
- Se pueden implementar de distintas maneras.
- Barrera lineal por conteo de variable:

```
P(llegada);
cont++;
if(cont<n)
    V(llegada);
else
    V(salida);
P(salida);
cont--;
if(cont>0)
    V(salida);
else
    V(llegada);
```

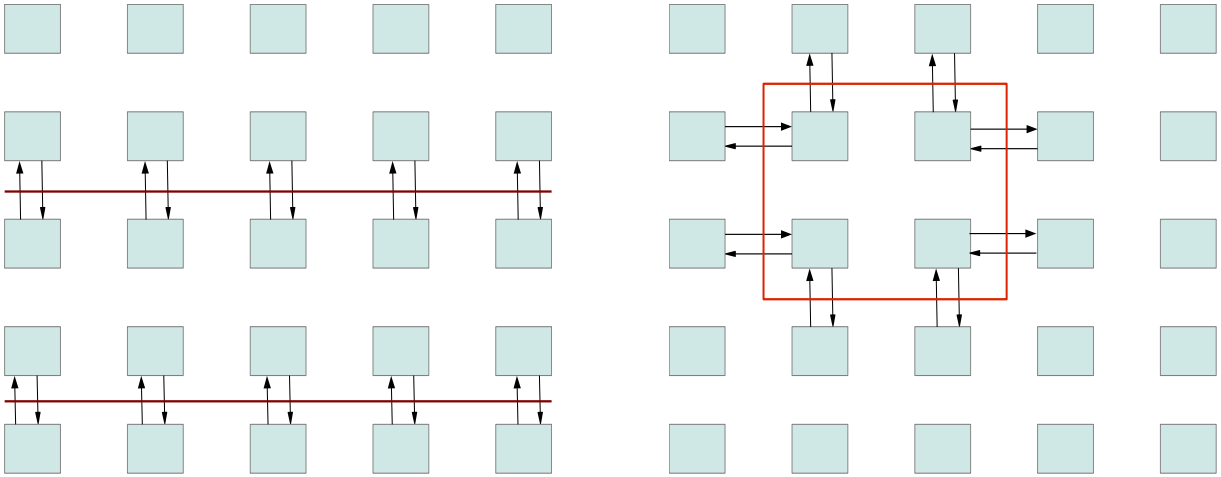
- Coste lineal en el número de elementos de proceso, por acceso en secuencial a la evaluación y actualización del contador.
- Con paso de mensajes un proceso distinguido (Maestro) se encarga de gestionar la variable: se sincroniza con el resto para recibir peticiones de actualización y comunicarles su valor $\Rightarrow$ comunicaciones y aumento de coste computacional.

Ejemplo - sincronización en árbol

- La contención se puede reducir descentralizando.
- Sincronización por subgrupos hasta centralizar.
- Ejemplo de suma de números, donde se acumulan los resultados parciales para obtener el total.
- Se puede utilizar función `Reduction`.
- Coste depende de implementación, pero puede ser $\theta(\log p)$:



Ejemplo - iteración de Jacobi, efecto **Volumen-Superficie**



- Distribución de mayor dimensión produce menos frontera para la misma superficie (o menos superficie para el mismo volumen), lo que puede generar menos comunicaciones.
- Distribución por bloques de $\frac{n}{p}$ filas:

$$4\left(t_s + nt_w\right)$$

- Distribución por bloques cuadrados de tamaño $\frac{n}{\sqrt{p}} \times \frac{n}{\sqrt{p}}$:

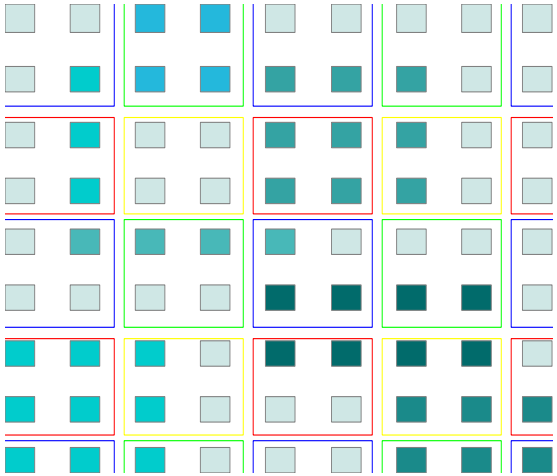
$$8\left(t_s + \frac{n}{\sqrt{p}}t_w\right)$$

- La distribución *checkerboard* es más escalable.

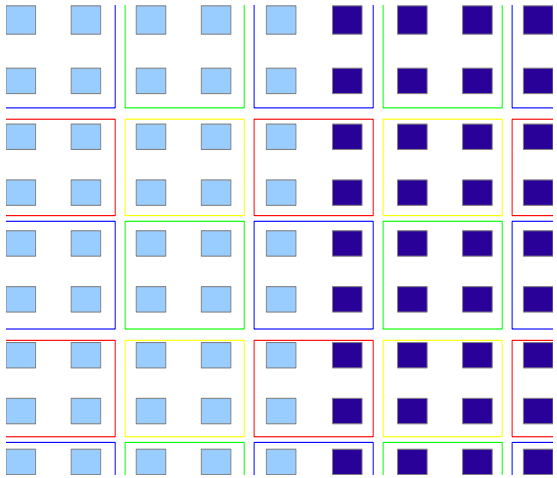
Distribuciones cíclicas

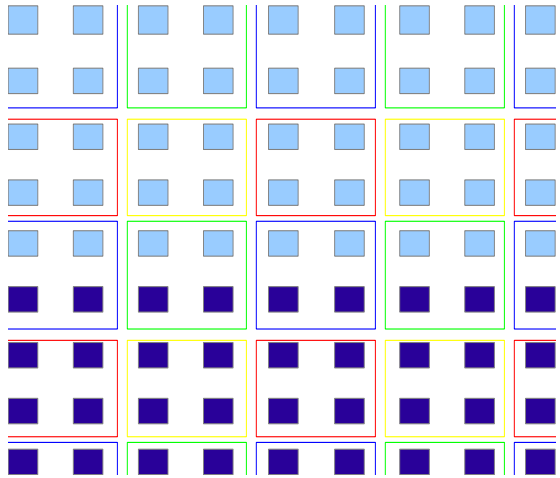
Pueden ser preferibles por volúmenes de computación distintos en

- distintas posiciones:



- iteraciones distintas:





Contenido

- Se pueden consultar las secciones correspondientes y sus ejemplos del capítulo 6 del libro de Introducción a la Programación Paralela y en el Concurso de Programación Paralela.
  - Algoritmo de Cannon de multiplicación de matrices (libro IPP).
  - Método iterativo para resolución de sistemas de ecuaciones (ejercicio en libro de IPP).
  - Juego de la vida (ejemplo en libro IPP).
  - En el CPP, en los que hay que hacer varias iteraciones y para realizar una iteración se necesitan los resultados de iteraciones anteriores. Varias variantes de la iteración de Jacobi y el juego de la vida.
- Y pensar cómo podrían implementarse los ejemplos vistos en esta sesión.