



UNIVERSIDAD DEL BÍO-BÍO  
FACULTAD DE CIENCIAS EMPRESARIALES

# Patrones de Programación Paralela

## Computación Heterogénea


Profesor: Dr. Joel Fuentes - [jfuentes@ubiobio.cl](mailto:jfuentes@ubiobio.cl)

Ayudantes:

- Daniel López - [daniel.lopez1701@alumnos.ubiobio.cl](mailto:daniel.lopez1701@alumnos.ubiobio.cl)
- Sebastián González - [sebastian.gonzalez1801@alumnos.ubiobio.cl](mailto:sebastian.gonzalez1801@alumnos.ubiobio.cl)

Página web del curso: <http://www.face.ubiobio.cl/~jfuentes/classes/ch>

# Patrones de diseño

- 
1. Patrones de Control Paralelo
  2. Patrones de Administración de Datos
  3. Otros Patrones

# Patrones de diseño paralelo

- Un patrón de diseño es una combinación de tareas recurrentes que resuelve un problema específico en el diseño de algoritmos paralelos.
- Patrones entregan un “vocabulario” para el diseño de algoritmos.
- Puede ser útil comparar patrones paralelos con patrones seriales.
- Patrones son universales, pueden ser usados en cualquier sistema de programación paralela.

# Patrones de Control Paralelo

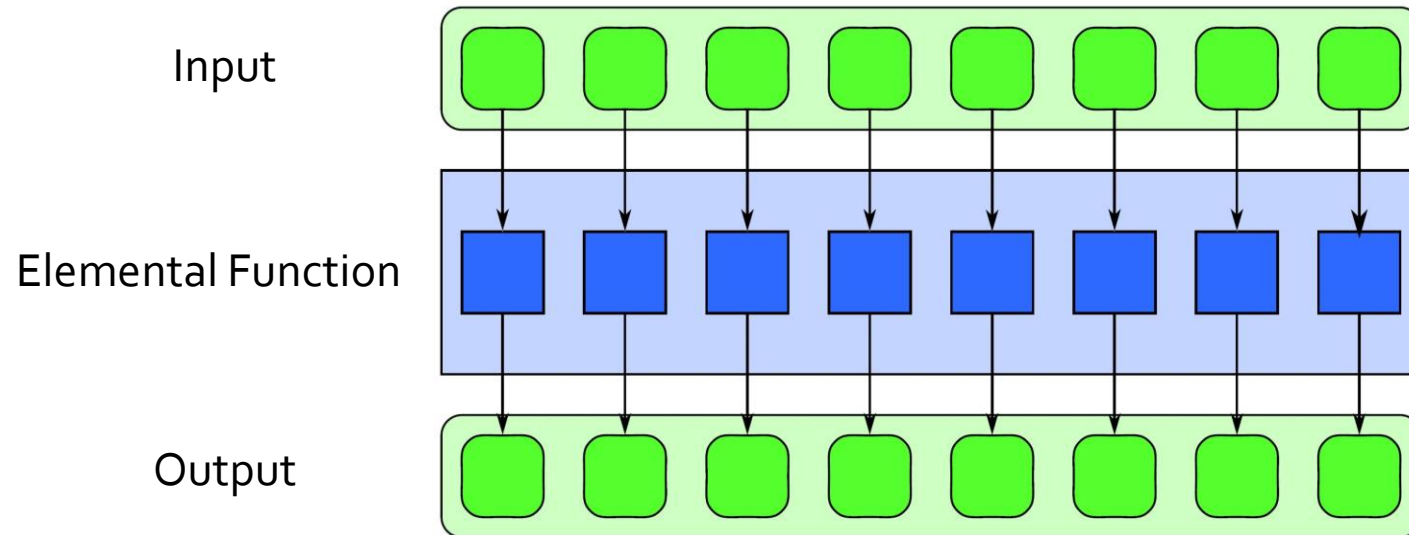
- Se extienden desde patrones de control serial
- Cada patrón de control paralelo está relacionado al menos a un patrón de control serial, pero con especificaciones más flexibles.
- Patrones de control paralelo: **fork-join, map, stencil, reducción, scan, recurrencia**

# Patrones de Control Paralelo: Fork-join

- **Fork-join:** permite controlar el flujo a múltiples flujos paralelos, para luego unirlos.
- Muchos lenguajes de programación implementan este patrón mediante **spawn** y **sync**
  - El árbol de llamada es un árbol de llamadas en paralelo y funciones que son ejecutadas en flujo paralelo (spawned)

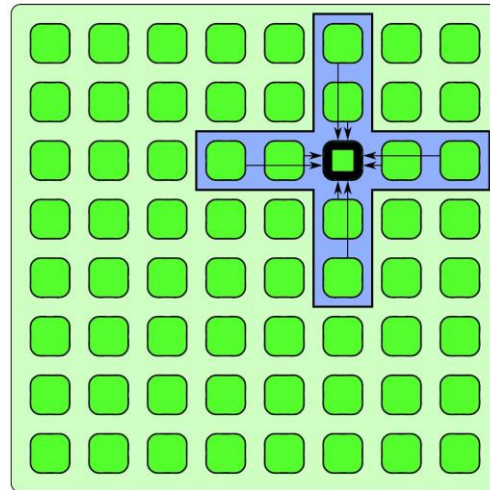
# Patrones de Control Paralelo: Map

- **Map:** ejecuta un función sobre todos los elementos de una colección
- Replica una iteración serial donde cada iteración es independiente de otras. El número de iteraciones es conocido, y el cómputo sólo depende de la iteración y datos desde la colección.
- La función replicada es referida como “función elemental”



# Patrones de Control Paralelo: Stencil

- **Stencil:** Corresponde a una generalización de Map. Función elemental que accesa un conjunto de “vecinos”
- Normalmente combinada con iteración
- Condiciones de borde deben ser manejadas cuidadosamente.



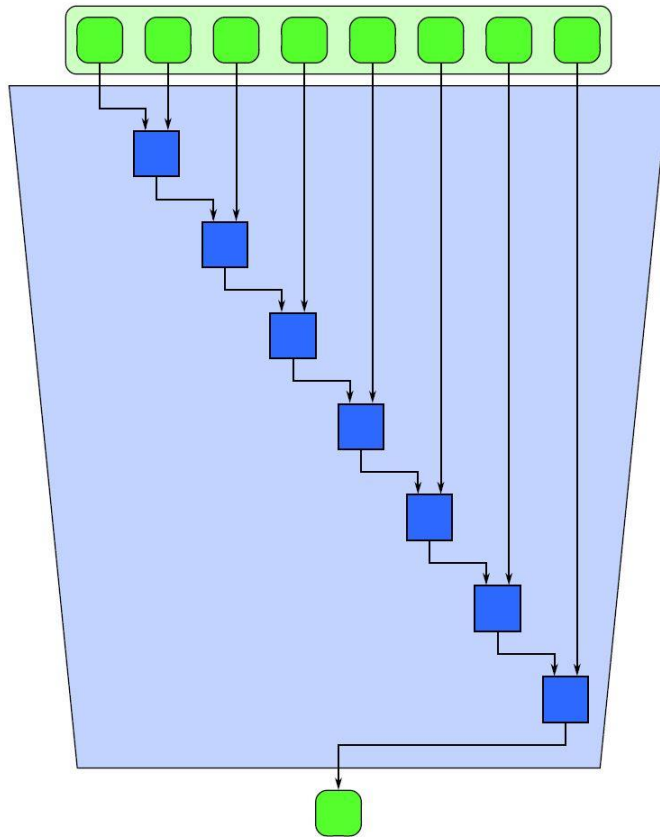
# Patrones de Control Paralelo: Reducción

- **Reducción:** Combina cada elemento en una colección usando una “función de combinación”
- Diferentes órdenes de la reducción son posibles.
- Ejemplos de funciones de combinación: add, mul, max, min, AND, OR, y XOR

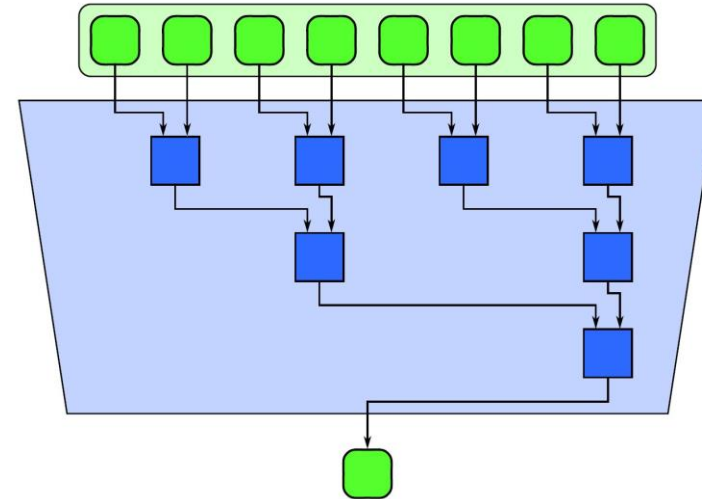


# Patrones de Control Paralelo: Reducción

Reducción Serial



Reducción Paralela

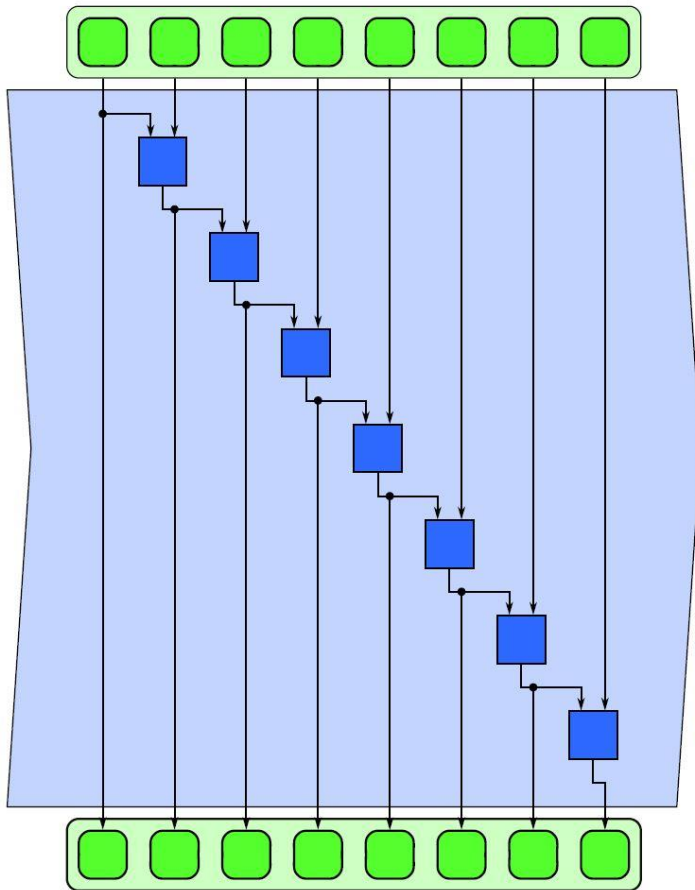


# Patrones de Control Paralelo: Scan

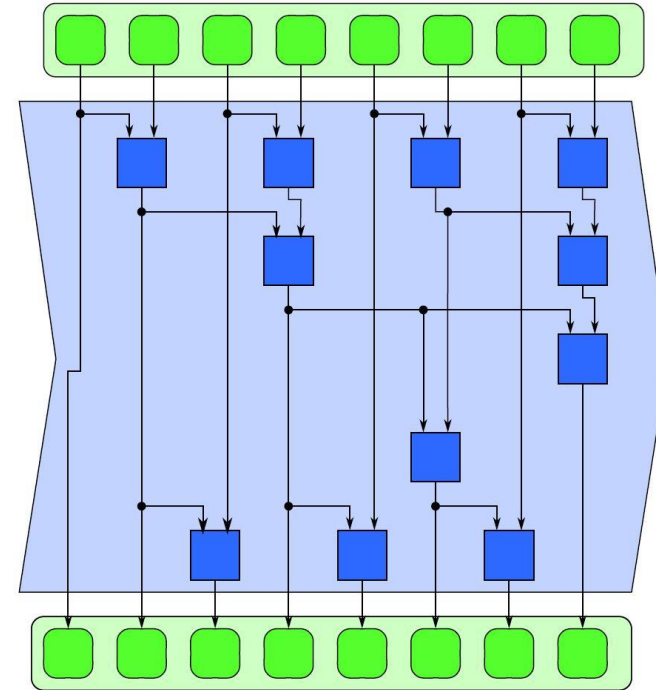
- **Scan:** computa las reducciones parciales de una colección
- Por cada output en una colección, una reducción del input hasta ese punto es ejecutada.
- Si la función usada es asociativa, el scan puede ser paralelizado
- Paralelizar el scan no es trivial, ya que pueden existir dependencias a iteraciones anteriores en el loop.
- Un scan paralelo requerirá más operaciones que la versión serial.

# Patrones de Control Paralelo: Scan

Serial Scan



Parallel Scan



# Patrones de Control Paralelo: Recurrencia

- **Recurrencia:** Versión más compleja que el Map, donde las iteraciones del loop puede depender de otras
- Similar a Map, pero elementos pueden usar outputs de elementos adyacentes como inputs
- Para que una recurrencia sea ejecutable, debe haber un orden serial de la recurrencia de elementos tal que puedan ser computados usando outputs generados anteriormente.

# Patrones de diseño

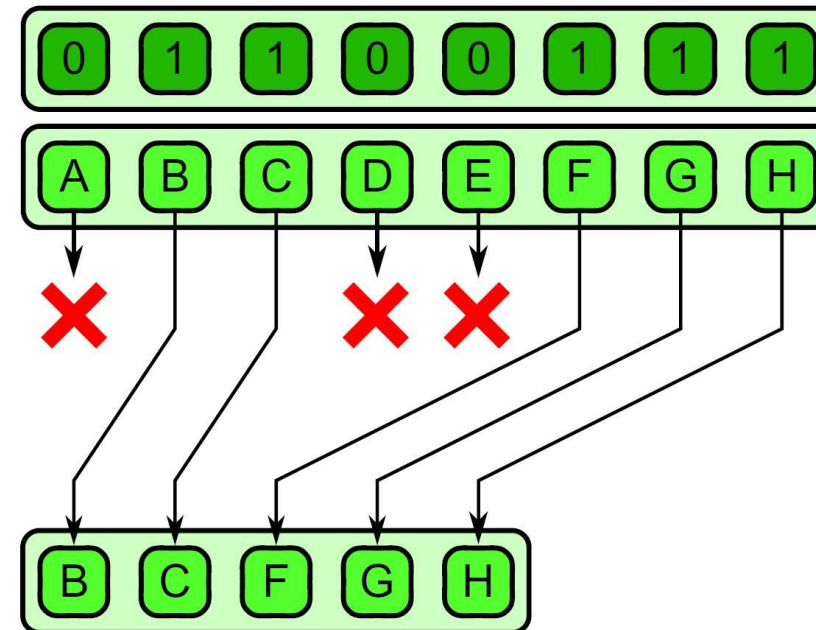
1. Patrones de Control Paralelo
- ➡ 2. Patrones de Administración de Datos
3. Otros Patrones

# Patrones de Datos Paralelos

- Para evitar problemas como data-race, es importante saber donde están los datos y si éstos son compartidos por multiples procesos o hilos.
- Algunos patrones de administración de datos ayudan con la localidad de datos
  - Datos disponibles cuando los procesos lo necesiten
  - Evitar cache misses
- Patrones de administración de datos paralela: **pack**, **pipeline**, **descomposición geométrica**, **gather** y **scatter**

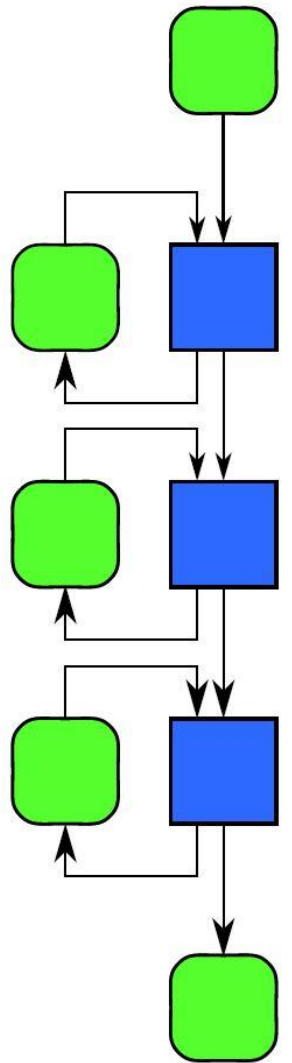
# Pack

- **Pack** es usado para eliminar espacio no utilizado en una colección.
- Elementos marcados como *false* son descartados/eliminados, y los elementos restantes ubicados en una secuencia contigua en el mismo orden.
- Útil cuando se usa Map
- **Unpack** es el patrón inverso y es usado para ubicar elementos de Vuelta en sus posiciones originales



# Pipeline

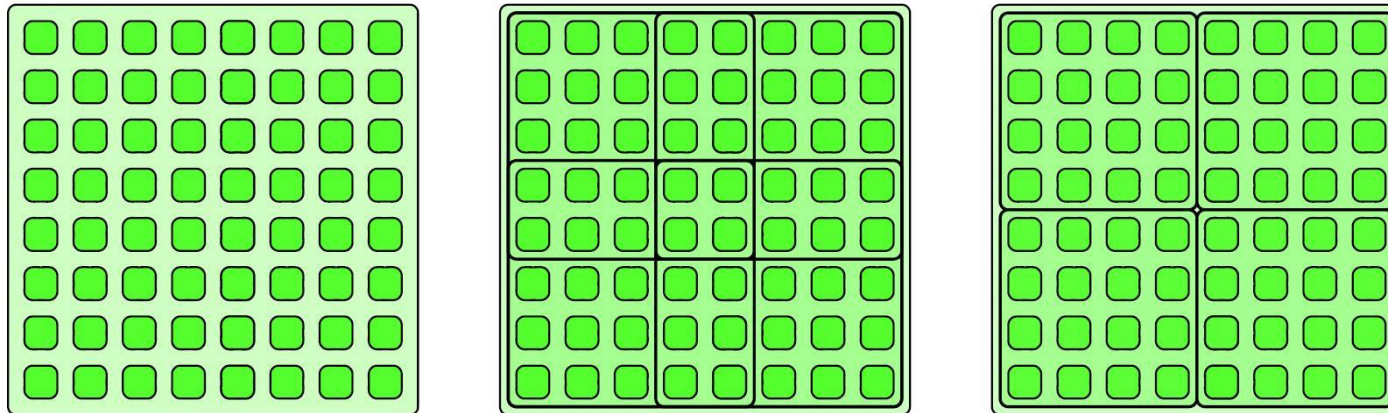
- **Pipeline** conecta tareas en una forma productor-consumidor
- Un pipeline lineal es la idea básica del patrón, sin embargo variaciones como en un grafo DAG también es posible.
- Pipelines son útiles cuando son utilizados con otros patrones que obtengan mayor paralelismo.





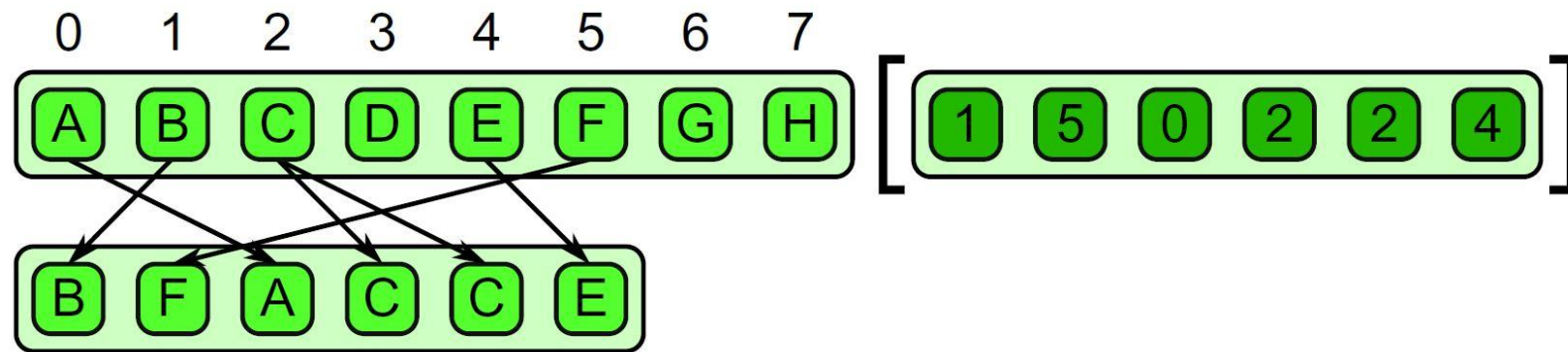
# Descomposición Geométrica

- **Descomposición Geométrica** – organiza datos en subcolecciones.
- Descomposición con superposición y sin superposición son posibles.
- Este patrón no necesariamente mueve datos, sólo nos entrega otra vista de este.



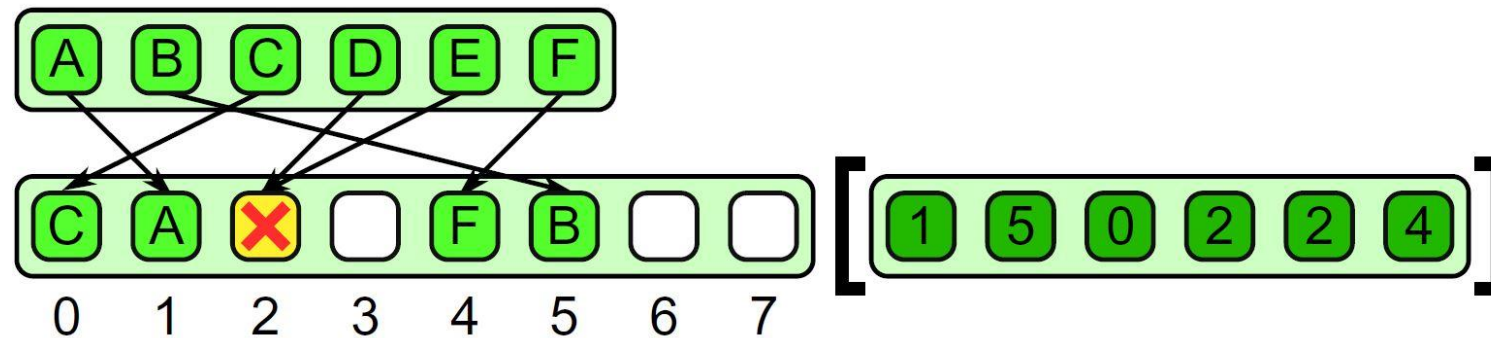
# Gather

- **Gather** lee una colección de datos dado una colección de índices.
- Puede imaginarse como una combinación de map y lecturas seriales aleatorias.
- La colección resultante comparte el mismo tipo de la colección de entrada, pero el mismo tamaño que la colección de índices.




# Scatter

- **Scatter** es el inverso de gather
- Un conjunto de datos y otro de índices es requerido. Cada elemento de la entrada es escrito como resultado en el índice entregado para esa posición.
- Condiciones de data-race puede ocurrir cuando dos elementos son escritos a la misma ubicación.



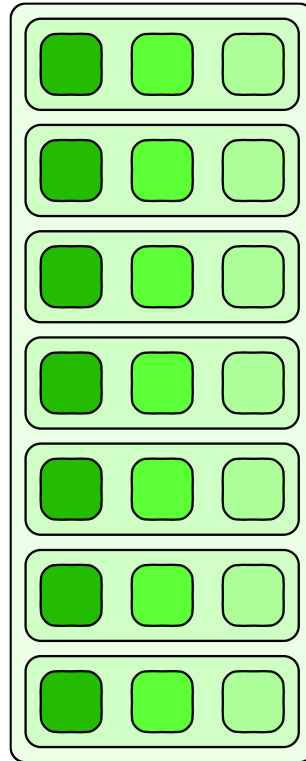
# Patrones de diseño

1. Patrones de Control Paralelo
2. Patrones de Administración de Datos
-  3. Otros Patrones

## Otros Patrones Paralelos: AoS vs SoA

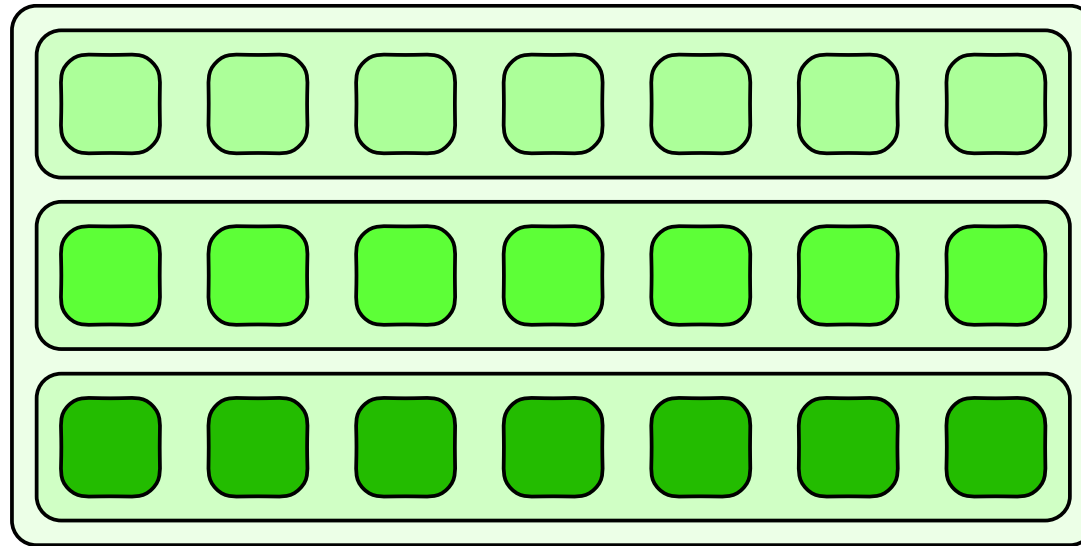
- **Array of Structures (AoS)**

- Puede entregar mejor utilización de la caché si los datos son accedidos aleatoriamente.



## Otros Patrones Paralelos: AoS vs SoA

- **Structures of Arrays (SoA)**
  - Típicamente mejor para vectorización



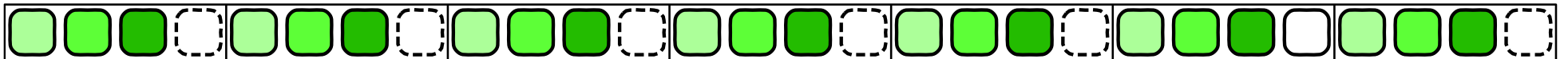
# Otros Patrones Paralelos: AoS vs SoA

- Organización en memoria:

Array of Structures (AoS), padding at end



Array of Structures (AoS), padding after each structure



Structure of Arrays (SoA), padding at end



Structure of Arrays (SoA), padding after each component



# Otros Patrones Paralelos: AoS vs SoA

## AoS Code

```
struct node {  
    float x, y, z;  
};  
struct node NODES[1024];  
  
float dist[1024];  
for(i=0;i<1024;i+=16){  
    float x[16],y[16],z[16],d[16];  
    x[:] = NODES[i:16].x;  
    y[:] = NODES[i:16].y;  
    z[:] = NODES[i:16].z;  
    d[:] = sqrtf(x[:]*x[:] + y[:]*y[:] + z[:]*z[:]);  
    dist[i:16] = d[:];  
}
```

## SoA Code

```
struct node1 {  
    float x[1024], y[1024], z[1024];  
}  
struct node1 NODES1;  
  
float dist[1024];  
for(i=0;i<1024;i+=16){  
    float x[16],y[16],z[16],d[16];  
    x[:] = NODES1.x[i:16];  
    y[:] = NODES1.y[i:16];  
    z[:] = NODES1.z[i:16];  
    d[:] = sqrtf(x[:]*x[:] + y[:]*y[:] + z[:]*z[:]);  
    dist[i:16] = d[:];  
}
```



# Otros Patrones Paralelos: AoS vs SoA

## AoS Code

```
struct node {  
    float x, y, z;  
};  
struct node NODES[1024];  
  
float dist[1024];  
for(i=0;i<1024;i+=16){  
    float x[16],y[16],z[16],d[16];  
    x[:] = NODES[i:16].x;  
    y[:] = NODES[i:16].y;  
    z[:] = NODES[i:16].z;  
    d[:] = sqrtf(x[:]*x[:] + y[:]*y[:] + z[:]*z[:]);  
    dist[i:16] = d[:];  
}
```

- Tipo de organización más lógica
- Extremadamente difícil de acceder por gathers y scatters
- No es muy útil para vectorización

## Otros Patrones Paralelos: AoS vs SoA

- Arreglos separados para cada campo de la estructura.
- Mantiene acceso a memoria contiguos cuando la vectorización.

### SoA Code

```
struct node1 {  
    float x[1024], y[1024], z[1024];  
}  
struct node1 NODES1;  
  
float dist[1024];  
for(i=0;i<1024;i+=16){  
    float x[16],y[16],z[16],d[16];  
    x[:] = NODES1.x[i:16];  
    y[:] = NODES1.y[i:16];  
    z[:] = NODES1.z[i:16];  
    d[:] = sqrtf(x[:] * x[:] + y[:] * y[:] + z[:] * z[]);  
    dist[i:16] = d[:];  
}
```

# Resumen

- Patrones de control paralelo
  - **fork-join, map, stencil, reducción, scan, recurrencia**
- Patrones de administración de datos
  - **pack, pipeline, descomposición geométrica, gather y scatter**
- Otros patrones

# Ejemplos usando patrones

- **Merge sort con reducciones**
- Ordenar un arreglo de enteros mediante map y reducción
- Idea: Mapear cada elemento en un vector de un solo elemento y luego aplicar la operación merge entre vectores
  - $<>$  es la operación merge :  $[1,3,5,7] <> [2,6,15] = [1,2,3,5,6,7,15]$
  - $[]$  es la lista vacía

# Ejemplos usando patrones

Entrada: [14,3,4,8,7,52,1]

Mapeado a [[14],[3],[4],[8],[7],[52],[1]]

Reducción desde la derecha:

$$\begin{aligned} & [14] <> ([3] <> ([4] <> ([8] <> ([7] <> ([52] <> [1]))))) \\ & = [14] <> ([3] <> ([4] <> ([8] <> ([7] <> [1,52])))) \\ & = [14] <> ([3] <> ([4] <> ([8] <> [1,7,52]))) \\ & = [14] <> ([3] <> ([4] <> [1,7,8,52])) \\ & = [14] <> ([3] <> [1,4,7,8,52]) \\ & = [14] <> [1,3,4,7,8,52] \\ & = [1,3,4,7,8,14,52] \end{aligned}$$

Se realizan  $O(n)$  operaciones merge, pero cada una toma  $O(n) = O(n^2)$

# Ejemplos usando patrones

Entrada: [14,3,4,8,7,52,1]

Mapeado a [[14],[3],[4],[8],[7],[52],[1]]

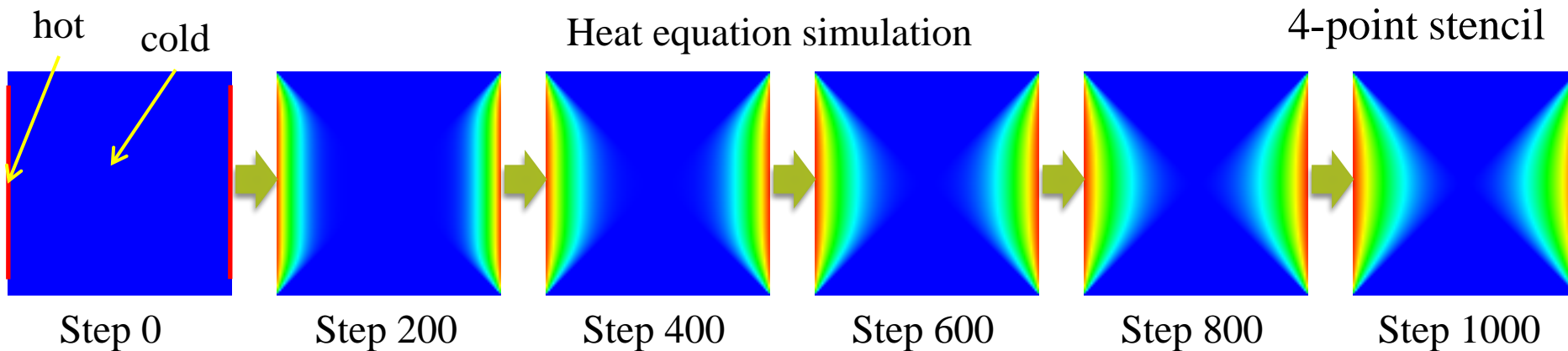
Reducción como árbol:

$$((( [14] <> [3] ) <> ([4] <> [8] )) <> (( [7] <> [52] ) <> [1] ))$$
$$= ([3,14] <> [4,8]) <> ([7,52] <> [1])$$
$$= [3,4,8,14] <> [1,7,52]$$
$$= [1,3,4,7,8,14,52]$$

Se realizan  $O(\log n)$  operaciones merge, pero cada una toma  $O(n)$  =  $O(n \log n)$

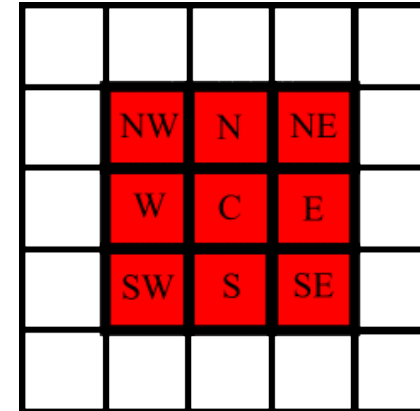
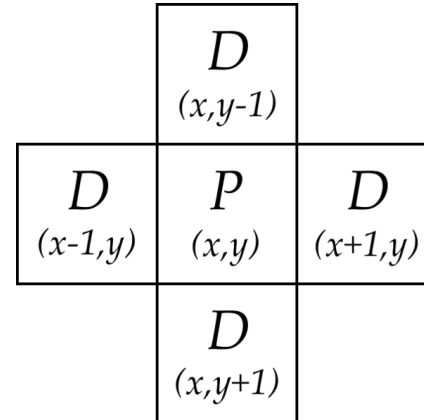
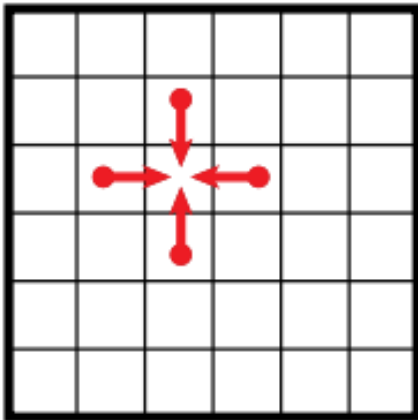
# Ejemplos usando patrones

- Simulaciones de propagación de calor con **stencil**
- Calcular la propagación de calor en una placa de metal usando la ecuación de Laplace e iteración de Jacobi



# Ejemplos usando patrones

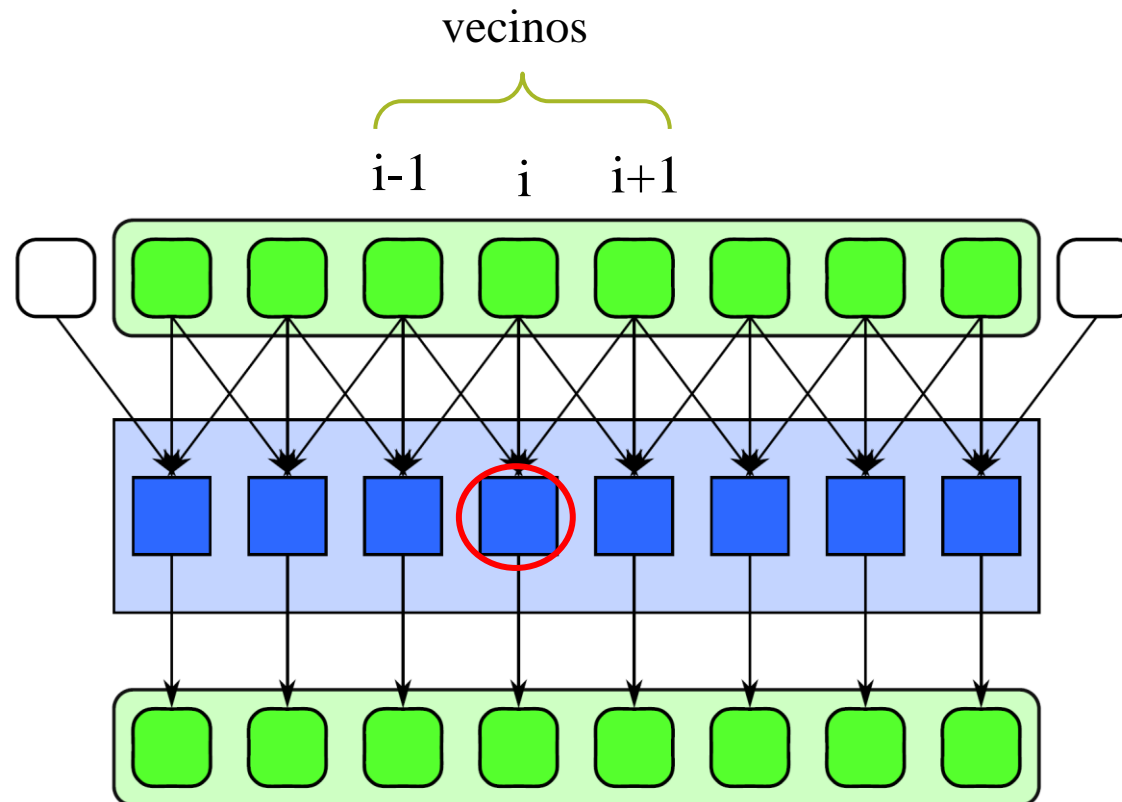
- Simulaciones de propagación de calor con **stencil**
- Calcular la propagación de calor en una placa de metal usando la ecuación de Laplace e iteración de Jacobi
- La operación consiste en calcular el promedio para todas las celdas de la superficie e iterar hasta converger.





# Ejemplos usando patrones

- Simulaciones de propagación de calor con **stencil**



# Referencias

- Parallel Computing Center. University of Oregon <http://ipcc.cs.uoregon.edu/index.html>