Sistemas Distribuidos y Paralelos

Fernando R. Rannou
Departamento de Ingeniería Informática
Universidad de Santiago de Chile

September 6, 2023

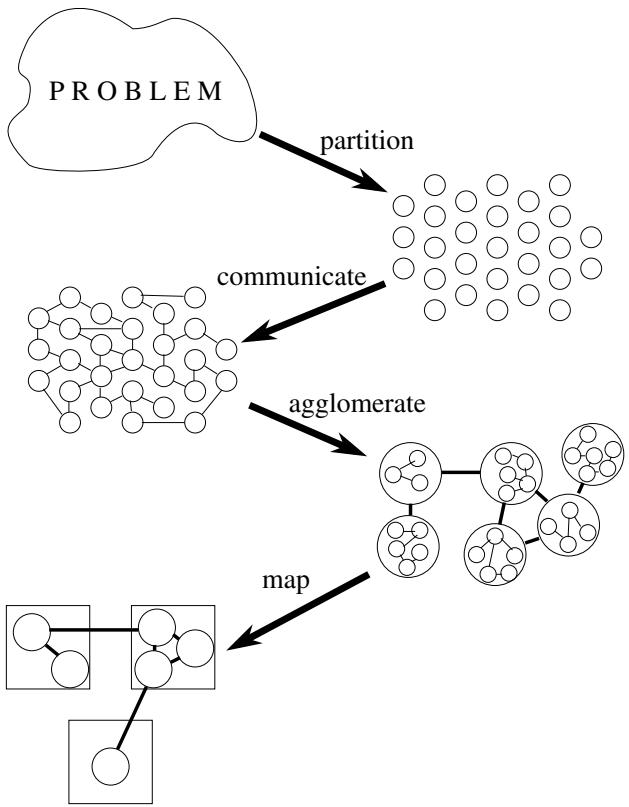
Diseño de algoritmos paralelos

- Diseñar algoritmos paralelos requiere considerar aspectos tales como:
  1. Tamaño esperado del problema
  2. Plataforma computacional: topología, red de conexión, etc
  3. Software disponible para implementación, etc.
- En programación secuencial estos apsectos no siempre son relevantes
- Diseñar y programar algoritmos paralelos eficientes requiere práctica; no existe receta única
- Sin embargo, hay algunos elementos que un diseñador debiera considerar; Ian Foster llama a estos aspectos *Diseño Metodológico*.

Diseño metodológico

Ian Foster propone las siguientes etapas:

1. Particionamiento
2. Comunicación
3. Aglomeración
4. Mapeo



Diseño metodológico

1. **Particionamiento**
   - La computación y/o el dato es divido en tareas más pequeñas.
   - La idea es explorar oportunidades de paralelización.
   - Aspectos prácticos como número de procesadores no son considerados en esta etapa.
2. **Comunicación**
   - Se determina la comunicación necesaria para coordinar la ejecución de las múltiples tareas.
3. **Aglomeración**
   - La partición y la comunicación resultantes son evaluados respecto de los requerimientos de rendimiento y costos de implementación.
   - Es posible re-agrupar (aglomerar) tareas en tareas más grandes para mejorar el rendimiento global y reducir costos

Diseño metodológico

4. **Mapeo**
   - Las tareas y los datos son asignados a los procesadores de tal forma de maximizar la utilización de los procesadores y minimizar los costos de comunicación
   - El mapeo puede ser estático o dinámico

Particionamiento

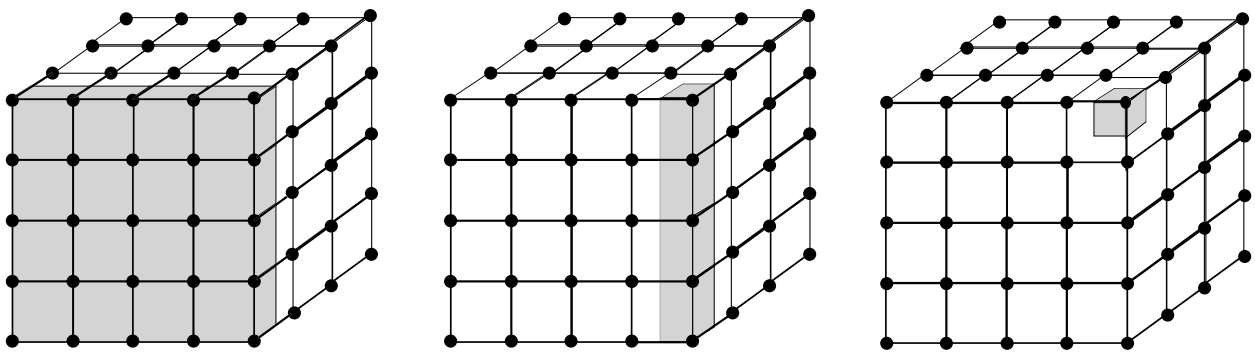
- El problema se descompone en computaciones más simples, por ejemplo:
  - particionando la computación misma (sobre los mismos datos)
  - particionando los datos
  - particionando computación y datos (preferido)
- Idealmente, queremos un gran número de tareas muy pequeñas
- La idea es explorar las oportunidades de paralelización de mi aplicación
- El resultado es una descomposición de *granularidad fina* (fined-grained)

Descomposición del dominio

- Consiste dividir los “datos”, ojala en trozos de igual tamaño
- Luego dividimos la “computación” de tal forma de adaptarse a la división de los datos
- Particionamos los datos que más frecuentemente son accesados por la computación
- Dependiendo de la aplicación, podriamos tener
  - Descomposición de estructuras de datos
  - Descomposición del dominio espacial (parecido al de datos)
  - Descomposición del dominio temporal

Descomposición de estructura de datos

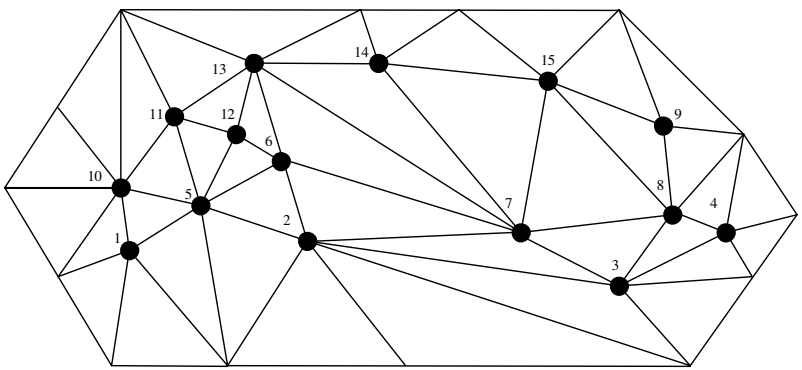
- En aplicaciones de computación científica típicamente particionamos las matrices u otros arreglos n-dimensionales
- Ejemplo: supongamos que estamos realizando un procesamiento de imágenes tridimensionales



- En esta etapa elegimos la descomposición más agresiva, es decir la última

Descomposición dominio espacial

- A veces conviene estudiar el particionamiento a nivel de la aplicación misma y no de las estructuras de datos
- Ejemplo: solución de una ecuación diferencial mediante elementos finitos

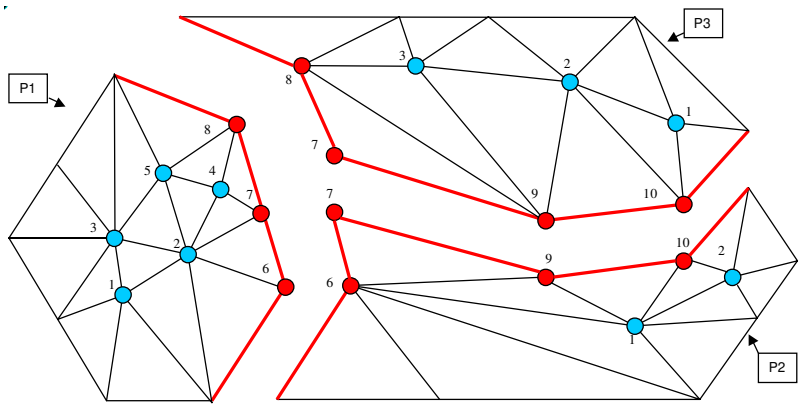


	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	x				x					x					
2		x	x		x	x	x								
3		x	x	x			x	x							
4			x	x				x	x						
5	x	x			x	x				x	x	x			
6		x			x	x	x					x	x		
7		x	x			x	x	x					x	x	x
8			x	x			x	x	x						x
9				x				x	x						x
10	x				x					x	x				
11					x					x	x	x	x		
12					x	x					x	x	x		
13						x	x				x	x	x	x	
14							x						x	x	x
15							x	x	x					x	x

- Sin embargo, este particionamiento inducirá un particionamiento en las matrices resultantes

Descomposición dominio espacial (cont)

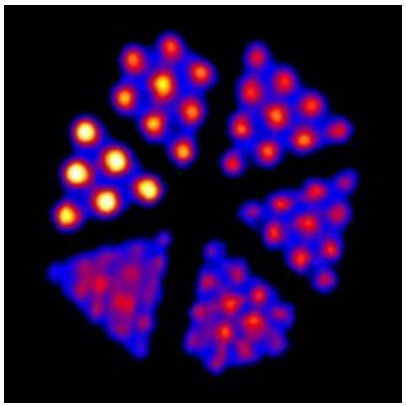
- Particionamiento sin traslape



	P1					P2		P3			logical boundary				
	1	2	3	4	5	1	2	1	2	3	6	7	8	9	10
1	x	x	x												
2	x	x	x	x	x						x	x			
3	x	x	x		x										
4		x		x	x							x	x		
5		x	x	x	x								x		
1						x	x				x			x	x
2						x	x								x
1								x	x						x
2								x	x	x				x	x
3									x	x			x	x	
6		x				x					x	x		x	
7		x		x							x	x	x	x	
8				x	x					x		x	x	x	
9						x			x	x	x	x	x	x	x
10						x	x	x	x					x	x

Descomposition temporal

- Considere el siguiente ejemplo: **Simulación de un experimento de medicina nuclear**
- Suponga deseamos "scannear" un cilindro con radioactivad en diferentes compartimientos durante 1 hora



- Una partición espacial, por ejemplo por compartimiento, sería errónea.
- Una partición temporal es correcta

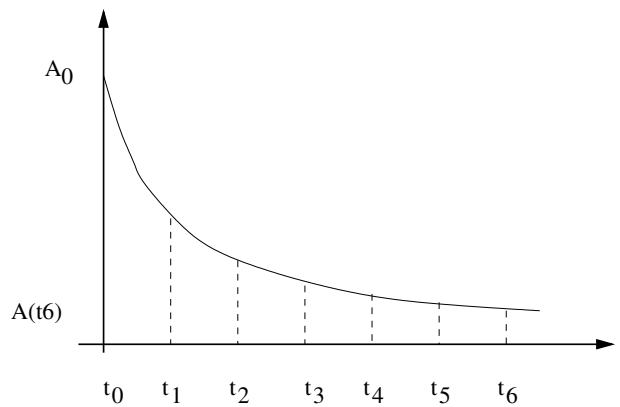
Descomposition temporal (cont)

- La radioactividad en los compartimientos decae según la ley:

$$A(t) = A_0 \exp\{-\lambda(t - t_0)\}$$

  donde A_0 es la actividad inicial en tiempo t_0, y λ es una constante inversamente proporcional a la vida media del isótopo

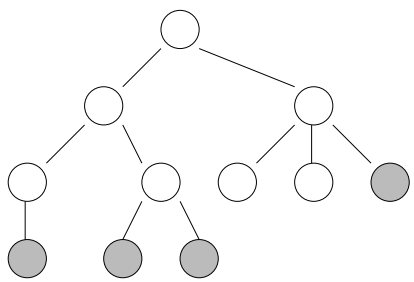
- Luego, una descomposición temporal en n etapas produce los siguientes intervalos de tiempo



$$t_i = i \times \frac{(t - t_0)}{n}$$

Descomposición funcional

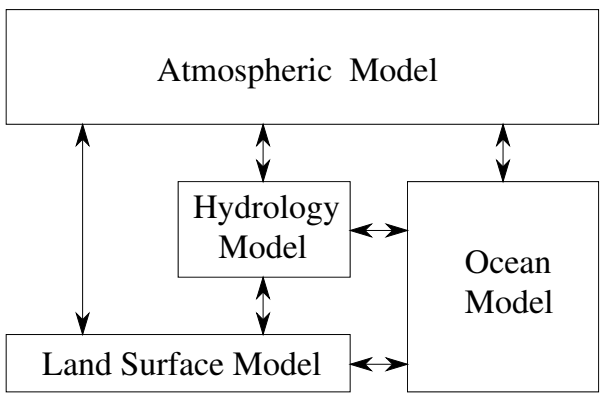
- Una descomposición funcional divide el problema en módulos funcionales conceptualmente distintos
- El esfuerzo de paralelización se centra en la computación y no en los datos
- **Ejemplo 1**: Búsqueda de soluciones en un árbol
  - Inicialmente se crea un tarea para el nodo raíz
  - Un tarea evalúa su nodo y si no es solución crea una o varias tareas (sub-árboles)
  - Además, se crea un canal de comunicación para cada nueva tarea, para el retorno de soluciones



```
funtion FindSolution(T) {
    if (IsSolution(T)) {
        value = evaluate(T);
        return(value);
    }
    foreach child T(i) of T
        FindSolution(T(i));
}
```

Descomposición funcional (cont)

- **Ejemplo 2**: Modelo computacional del clima
- Cada tarea representa un "código" completamente distinto
- Una vez dividido el problema funcionalmente, se analizan los requerimientos de datos y las inter-relaciones con los otros módulos
- Si las tareas son disjuntas, entonces no hay comunicación



Comunicación

- Una aplicación paralela generalmente implica un cierto nivel de comunicación entre sus tareas:
  1. **Altamente acopladas**: alto nivel de comunicación; Por ejemplo, existe comunicación cada 10-100 instrucciones
  2. **Medianamente acopladas**: Ejemplo: se envía o recibe mensajes cada 1000-10000 instrucciones
  3. **Débilmente acopladas**: las tareas casi no se comunican; algunos llaman a este tipo de tareas *Embarasozamente paralelas* (embarransingly parallel).
- Independiente del acoplamiento, la partición implicará un cierto requerimiento de flujo de información entre las tareas
- En esta etapa se especifica dicho flujo

Tareas y canales

- Podemos formalizar la necesidad de comunicación entre dos tareas como la existencia de un *canal* unidireccional de comunicación
- La tarea que envía la información se denomina productor o fuente
- La tarea que recibe la información, consumidor o destino
- Dicho canal puede o no puede implementarse en la práctica
- Un canal real implica un costo de comunicación y por lo tanto una demora en la computación
- Evitamos introducir canales y operaciones de comunicación innecesarios
- Intentamos explotar operaciones de comunicación concurrentes

Taxonomía de comunicación

Podemos caracterizar la comunicación con los siguientes 4 ejes:

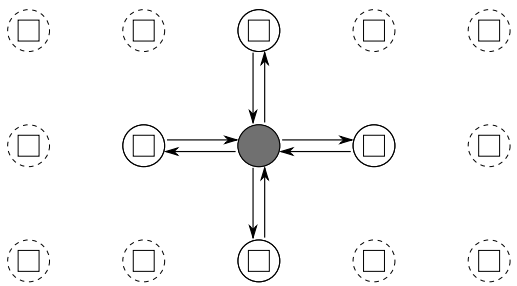
1. **Local/Global**:
   En comunicación local, cada tarea se comunica con un número pequeño de otras tareas (generalmente, tareas "vecinas"); en comunicación global cada tarea se comunica con muchas o todas las tareas
2. **Estructurada/No estructurada**:
   En comunicación estructurada, la tarea y sus vecinos forman una estructura regular de comunicación; en no estructurada, las topologías pueden ser arbitrarias
3. **Estática/Dinámica**:
   La topología e identidad de las tareas no cambia en comunicación estática
4. **Síncrona/Asíncrona**:
   En comunicación síncrona las tareas cooperan en forma coordinada y pre-establecida. En comunicación asíncrona, las tareas intercambian información en modo competitivo y aleatoreo

Comunicación local

Esta comunicación surge cuando los canales unen productores y consumidores vecinos

- Ejemplo: iteración de Jacobi en método de diferencias finitas:

$$x_{i,j}^{(t+1)} = \frac{4x_{i,j}^{(t)} + x_{i-1,j}^{(t)} + x_{i+1,j}^{(t)} + x_{i,j-1}^{(t)} + x_{i,j+1}^{(t)}}{8}$$



- Luego, cada tarea es productor y consumidor a la vez, y ejecuta el siguiente algoritmo:

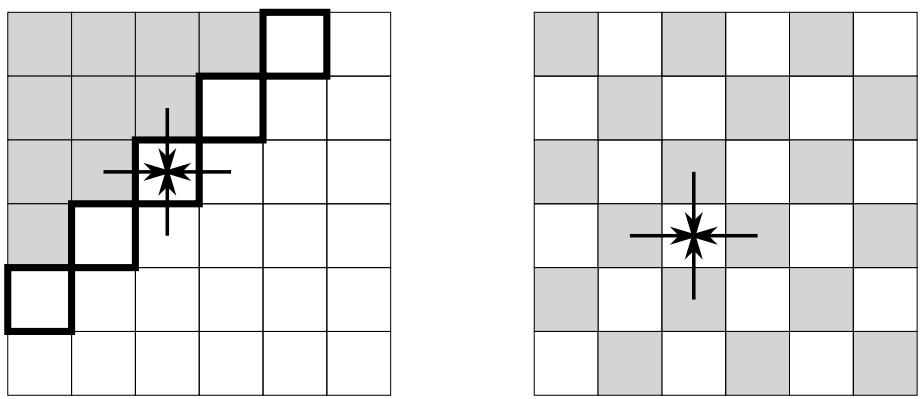
```
for t=0 to T-1 {
    send x_{i,j}(t) a cada vecino
    receive x_{i-1,j}(t), x_{i+1,j}(t), x_{i,j-1}(t), x_{i,j+1}(t)
    de vecinos
    calcular x_{i,j}(t+1)
}
```

Comunicación local (cont)

- En el ejemplo anterior, existe máxima concurrencia en la comunicación
- En la iteración Gauss-Seidel, la actualización de un valor $x_{i,j}$ toma en cuenta las actualizaciones más recientes de sus vecinos:

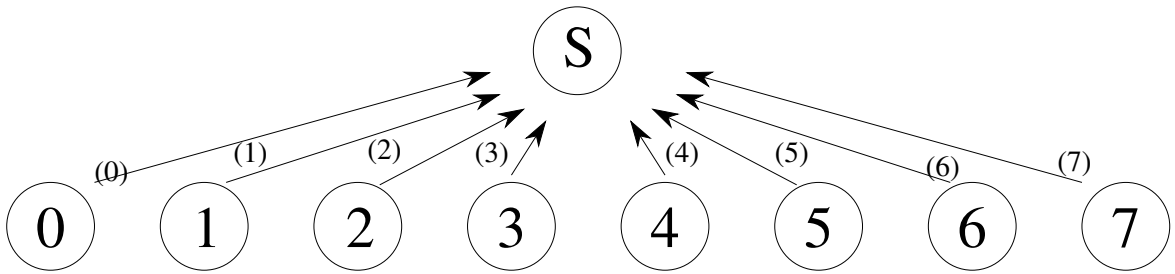
$$x_{i,j}^{(t+1)} = \frac{4x_{i,j}^{(t)} + x_{i-1,j}^{(t+1)} + x_{i+1,j}^{(t)} + x_{i,j-1}^{(t+1)} + x_{i,j+1}^{(t)}}{8}$$

- La cual reduce el nivel de concurrencia en la comunicación



Comunicación global

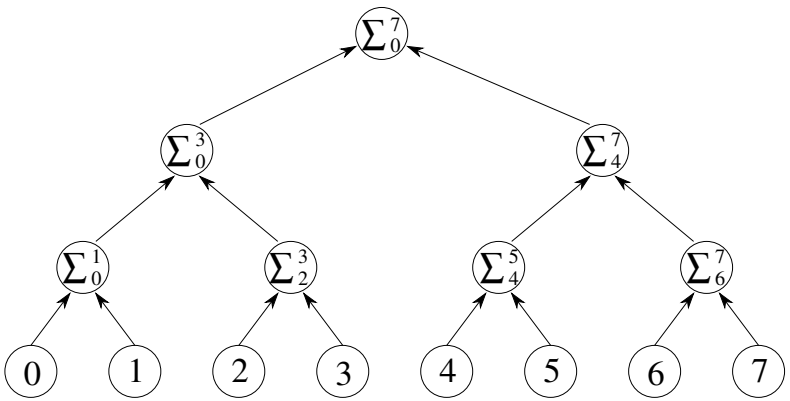
- Típicas operaciones globales son broadcast, reducciones y barreras
- No es necesario que todas las tareas participen
- Ejemplo: reducción global de suma $S = \sum_{i=0}^{N-1} x_i$



- Como la tarea que realiza la reducción no puede recibir más que un mensaje a la vez, la operación demora $O(N)$, para N números.
- El algoritmo centralizado no distribuye ni la comunicación ni la computación
- Es decir, es simplemente secuencial

Comunicación y computación distribuida

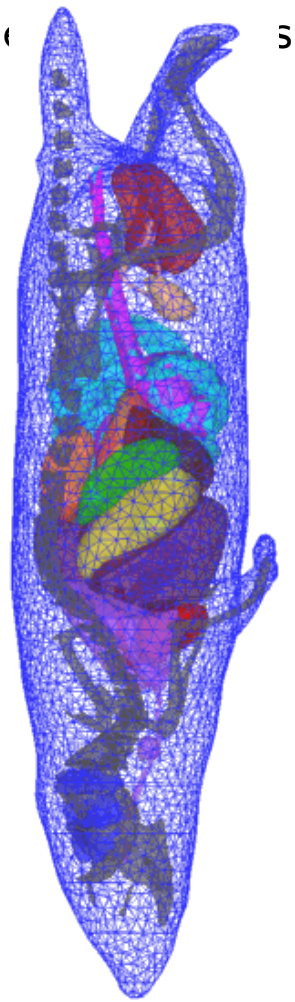
- Usamos la estrategia *dividir-y-reinar* para encontrar oportunidades de distribución la computación y comunicación
- La técnica consiste en dividir el problema en dos sub-problemas menores.
- Si estos sub-problemas no so aún lo suficientemente pequeños, aplicamos la divisón recursivamente.
- En nuestro ejemplo, dividimos la suma en dos:$S = \sum_{i=0}^{2^{n-1}-1} x_i + \sum_{i=2^{n-1}}^{2^n-1} x_i$
- Aplicando esta técnica recursivamente, la reducción es $O(\ln N)$




```
function dividir-y-reinar(P) {
   if (base(P)) return(solve(P));
   else {
        (L, R) = particionar(P);
        Ls = dividir-y-reinar(L);
        Rs = dividir-y-reinar(R);
        return( combinar(Ls, Rs) );
   }
}
```

Comunicación dinámica y no estructurada

- Todos los ejemplos anteriores involucran comunicación estática y estructurada:
  1. Se crea topología de grilla, árbol, arreglo, etc
  2. Los canales y la identificación de las tareas son si[illegible]smas
- Considere la generación de grillas tridimensionales



- En áreas de mayor curvatura, se generan más nodos que en áreas de menor curvatura
- La grilla computacional se puede adaptar a la generación de elementos
- El patrón de comunicación es irregular y cambia con el tiempo

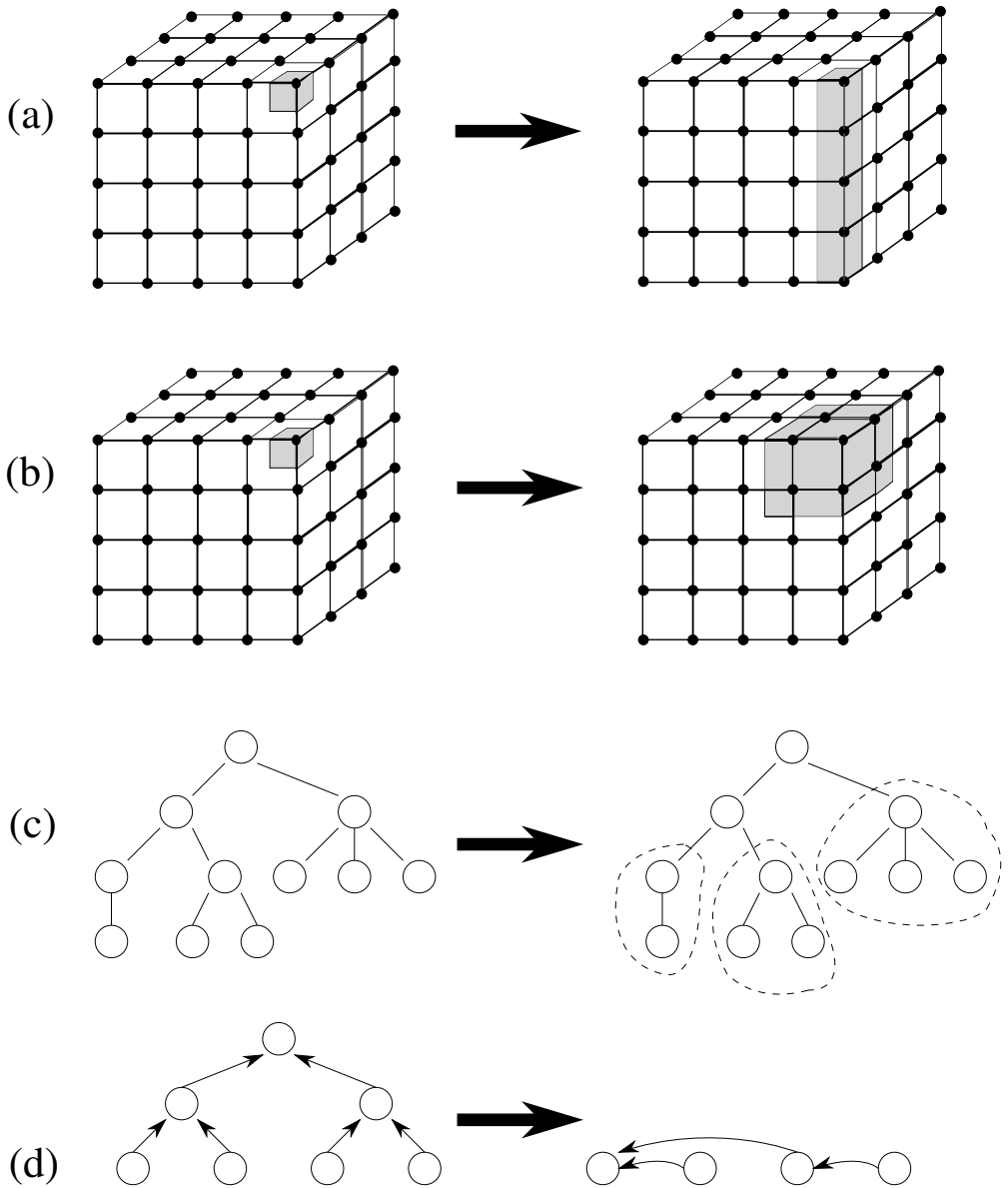
Comunicación asíncrona

- En comunicación asíncrona las tareas que poseen datos requeridos por otras, no saben cuándo estos datos deben ser enviados
- No hay acuerdo previo para el patrón de comunicaciones
- Consumidores deben pedir en forma explícita los datos a los productores
- Ejemplo: repositorio distribuido y compartido por múltiples tareas
  1. El repositorio es distribuido entre todas la tareas; las tareas solicitan datos ubicados en otras tareas; además, deben interrumpir periódicamente su procesamiento para servir requerimientos de datos de otras tareas
  2. El repositorio es distribuido sobre un subconjunto de tareas que sólo prestan servicios de almacenamiento (lectura/escritura)
  3. Si existe memoria compartida, las tareas utilizan mecanismos de sincronización para mantener la consistencia de los datos

Aglomeración

- En las etapas anteriores, el objetivo es crear el máximo número posible de tareas, y así descubrir oportunidades de paralelización
- Durante aglomeración, revisamos el particionamiento y comunicación de tal forma de obtener un algoritmo que se ejecute eficientemente en la infraestructura computacional disponible
- En particular, consideramos si
  1. es útil y factible combinar tareas pequeñas en tareas más grandes
  2. replicar datos y/o comunicación

Aglomeración (cont)



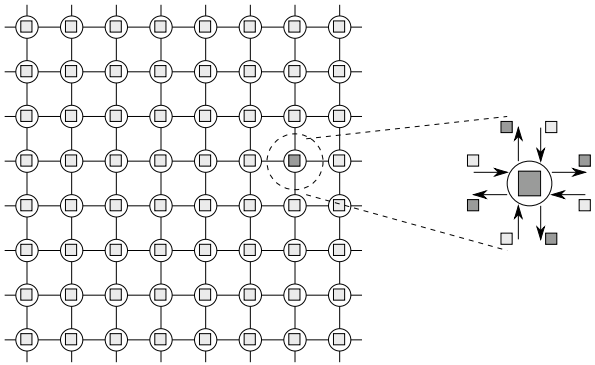
Aumentando la granularidad

- Un número muy grande de tareas implicaría
  - un excesivo costo de comunicación de datos
  - costo adicional en el overhead de comunicación
  - un delay en la computación
  - costo adicional para crear los procesos
- Si el número de vecinos de comunicación por tarea es pequeño, el número de operaciones de comunicación puede reducirse incrementando la granularidad, es decir aglomerando varias tareas en una
- El efecto resultante es llamado *Efecto Superficie-a-Volumen*

Efecto Superficie-a-Volumen

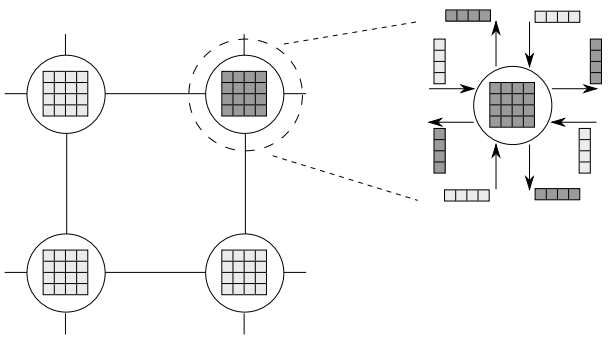
- Los requerimientos de comunicación de una tarea son proporcionales a la "superficie" (perímetro en 2D) de su subdominio
- Los requerimientos de computación son proporcionales al "volumen" (área en 2D) del mismo subdominio

(a)



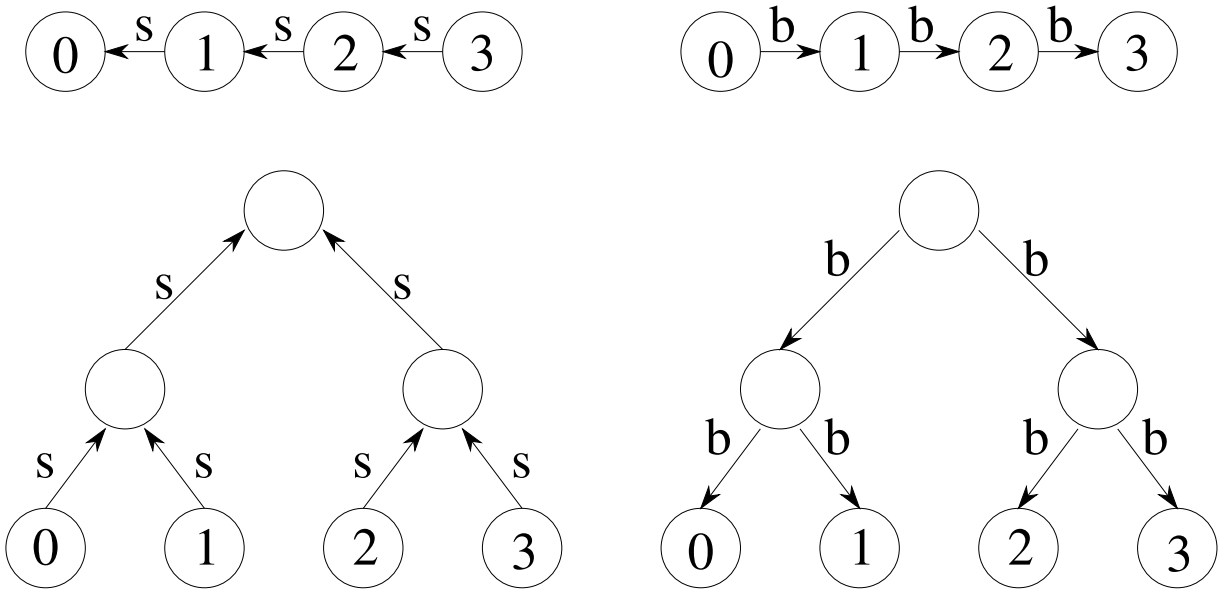
- La cantidad de comunicación por unidad de computación disminuye a medida que la tarea crece
- En (a) $64 \times 4 = 256$ mensajes son enviados. En (b) sólo $4 \times 4 = 16$

(b)



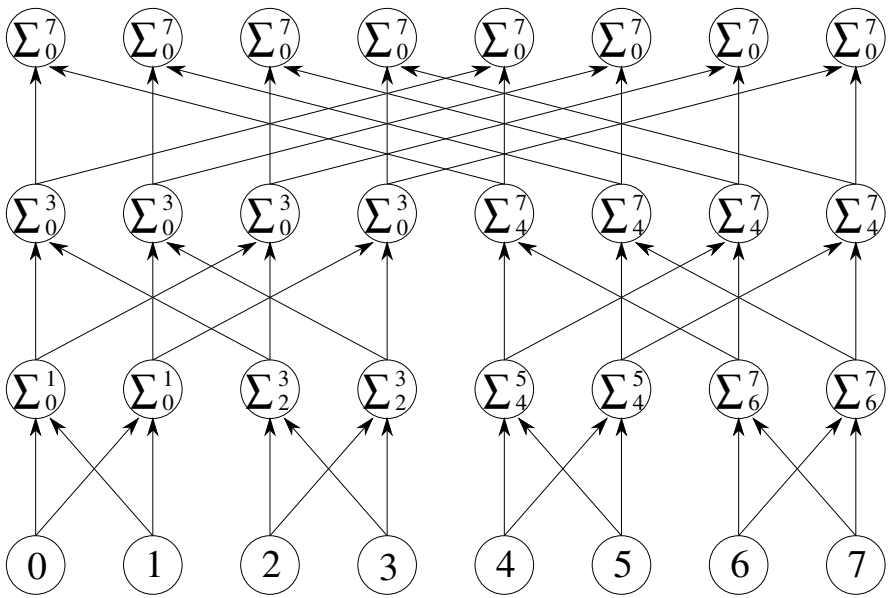
Replicando computación

- ¿Qué tan malo es que dos o más tareas realicen la misma computación para un conjunto pequeño de operaciones
- No tanto, especialmente si se reducen las operaciones de comunicación
- Ejemplo: reducción global de suma: con una topología de arreglo o árbol
- El arreglo lineal demora $2(N-1)$ etapas y el árbol demora $2 \log N$



Replicando computación (cont)

- El algoritmo anterior es óptimo en el sentido que no realiza operaciones innecesarias de comunicación ni de computación
- Si replicamos el patrón de comunicación árbol, obtenemos la estructura de comunicación *butterfly*:



- El cual realiza la reducción en $O(\log N)$ etapas.

Número de tareas

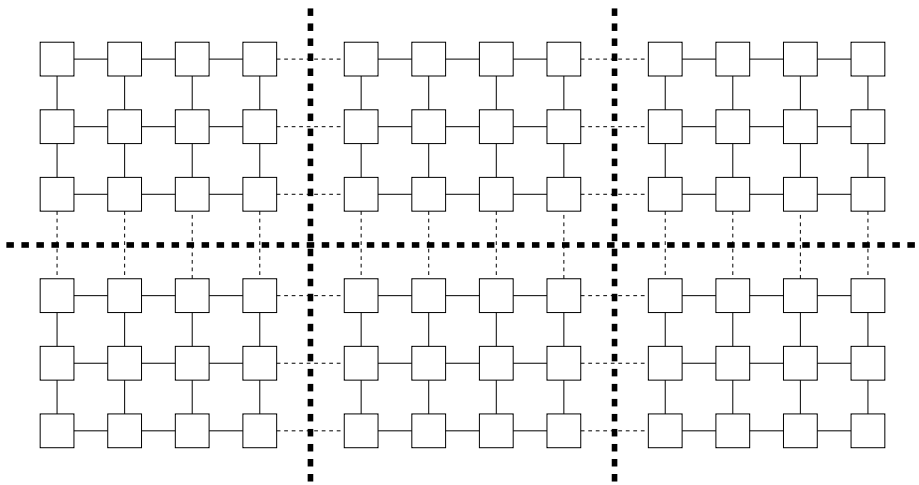
- El propósito de aglomeración es acercar el diseño a una posible implementación, en alguna arquitectura paralela
- En este sentido, disminuir el número de tareas parece algo razonable
- Sin embargo, existe buenas razones para dejar un número grande de tareas:
  1. Flexibilidad y portabilidad respecto de la arquitectura paralela
  2. Más oportunidad de traslapar cómputo con comunicación
  3. Más oportunidades de mapeo a procesadores y por ende mejor balance de carga
- En general, se recomienda que el número de tareas sea al menos un orden de magnitud mayor al número de procesadores

Mapeo

- Durante esta etapa, especificamos dónde se ejecutará cada tarea
- Este problema no existe en uniprocesadores o sistemas multiprocesadores de memoria compartida
- La meta de mapeo es minimizar el tiempo total de ejecución
- Usamos dos estrategias para lograr la meta:
  1. Colocamos tareas con computación concurrente en procesadores distintos
  2. Asignamos tareas que se comunican frecuentemente en el mismo procesador
- El problema de mapeo es NP-completo, es decir no existe un algoritmo polinomial que resuelva el problema general
- Sin embargo, para aplicaciones sencilla, existen algoritmos eficientes

Mapeo en una grilla

1. Continuando con el ejemplo de diferencias finitas, cada tarea realiza la misma cantidad de computación y comunicación
2. Luego, mapeamos las tareas explotando el efecto Superfice-a-Volumen



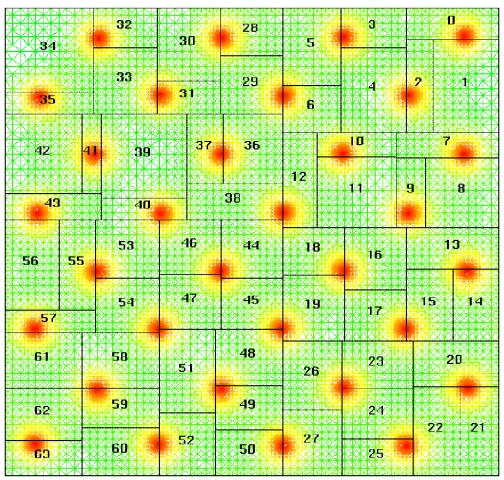
3. Las líneas segementadas definen las fronteras de los procesadores
4. Luego de este mapeo, es posible que sea conveniente re-agrupar tareas que son mapeadas al mismo procesador

Algoritmos de balance de carga

- En aplicaciones complejas, no estructuradas y dinámicas, se usan algoritmos de balance de carga para realizar el mapeo
- En realidad, realizan una tarea similar a la de descomposición
- El algoritmo puede ser:
  1. **Local**: la carga se re-distribuye usando información de la carga actual de los vecinos
  2. **Global**: necesitamos conocer la carga de todas la tareas para reasignar la computación
  3. **Estática**: el mapeo se determina analizando el problema antes que se ejecute y no cambia durante la ejecución
  4. **Dinámica**: se re-asigna la carga cada cierto tiempo

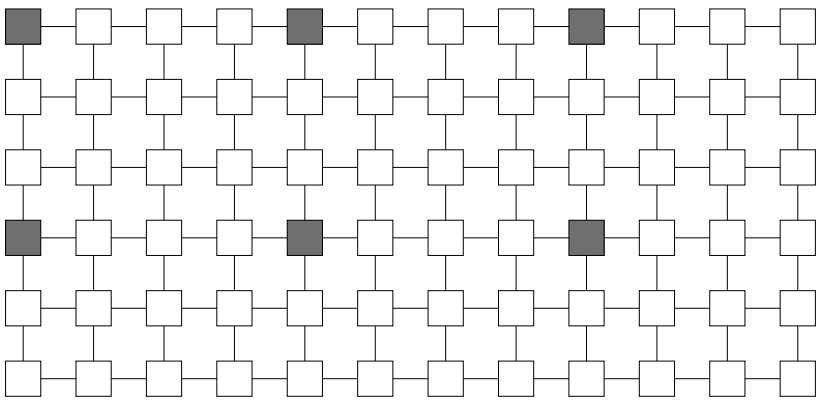
Bisección recursiva

- Intenta particionar el dominio en sub-dominios con costos de computación similares, minimizando los costos de comunicación, es decir el número de canales que cruzan las fronteras
- **Bisección recursiva de coordenada** generalmente se aplica a grillas irregulares y realiza el particionamiento basándose en las coordenadas de los nodos.
- **Bisección recursiva no balanceada** realiza la partición en dominios cuyas razones de aspecto son mejores, y así evitar dominios muy "alargados y delgados"



Mapeos cíclicos

1. Si los costos de computación varían y existe bastante localidad espacial en los niveles de carga, entonces podemos usar mapeo cíclico y *scattered*
2. Volviendo al ejemplo de diferencias finitas

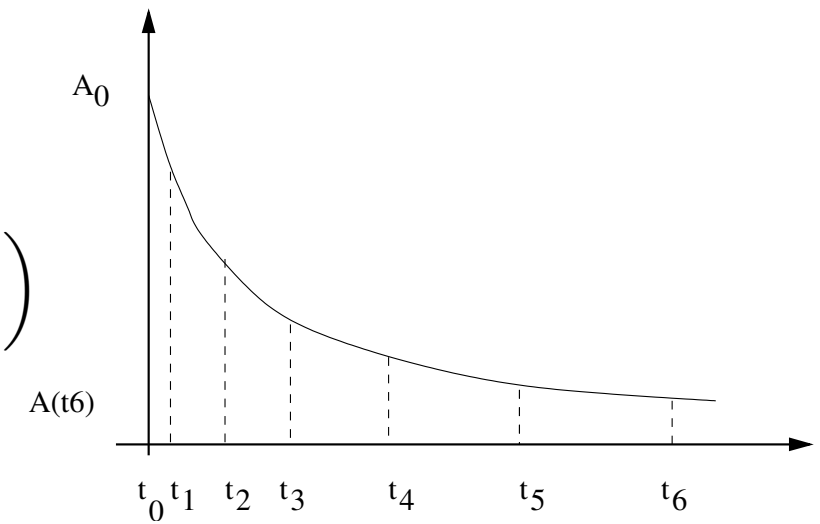


3. Si disponemos de P procesadores, cada procesador recibe cada P tareas de la grilla
4. Se debe verificar que la mejora en el balance de carga no haya introducido costos excesivos de computación

Balance ad-hoc

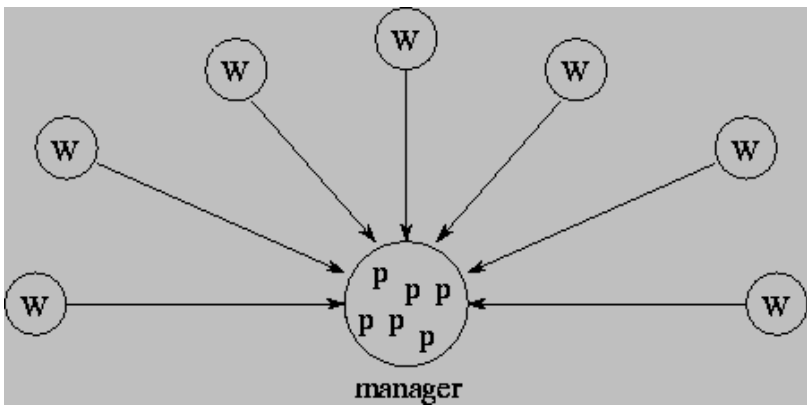
- No existe receta para el balance de carga
- A veces se debe estudiar el problema particular y crear un algoritmo que sólo es aplicable a dicho problema
- Ejemplo: para la simulación de experimentos de medicina nuclear, una partición de intervalos de tiempo iguales produce carga desbalanceada
- La carga se balancea asignando intervalos basados en el área bajo la curva

$$t_i = -\frac{1}{\lambda} \ln\left(\frac{(n-i)e^{-\lambda t_{i-1}} + e^{-\lambda t_n}}{n-(i-1)} \right)$$



Scheduling de tareas

- **Maestro/trabajadores**: una tarea mantiene el conocimiento de los trabajos que deben realizarse. En un sistema push, el maestro asigna tareas a los trabajadores. En un sistema pool los trabajadores solicitan tareas al maestro



- **Maestro/trabajador jerárquico** El esquema anterior se puede generalizar a varios submaestros que tiene asignado un subconjunto (disjunto) de trabajadores
- **Esquemas decentralizados**

Ejemplo: Histograma de datos 1D

- Considere un vector A_i, $0 \leq i \leq N$ tal que $0 \leq A_i \leq 255$
- Su histograma H_j, $0 \leq j \leq 255$ se puede calcular con el siguiente código secuencial

```
for (i=0; i < N; i++) {
    H[A[i]]++;
}
```

- Pero si los datos son reales, no se puede usar A_i para indexar H
- Es necesario especificar `bins` que almacenen la cantidad de números en un rango expecífico
- Sea `minval` y `maxval` los números menor y mayor de los datos, respectivamente, y suponga que deseamos `nbins`
- El ancho de cada bin es

```
binwidth = (maxval - minval)/nbins
```

- Y los valores máximos de cada bin son

```
for (b=0; b < nbins; b++)
    binmaxes[b] = minval + binwidth*(b+1);
```

Histograma 1D, secuencial

- El bin b acumula info de los datos tal que

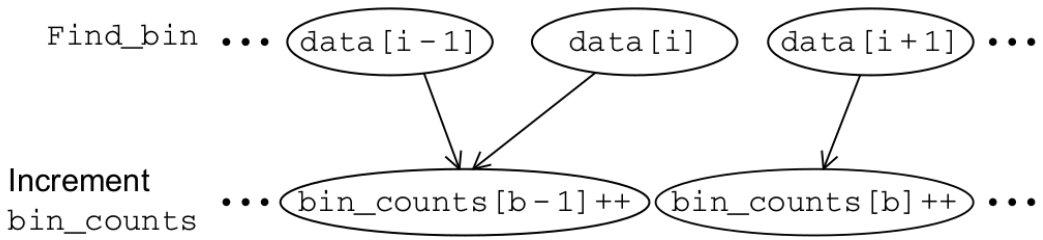
  `binmaxes[b-1] <= datos < binmaxes[b]`

- Note que es necesario tratar el bin 0 en forma particular
- Sea `Findbin()` la función que retorna el número del bin al que un dato pertenece
- Entonces, habiendo inicializado `bincounts[]=0`

```
for (i=0; i < N; i++) {
    bin = Findbin(data[i], binmaxes, nbins, minval);
    bincounts[bin]++;
}
```

Histograma 1D, secuencial

- Hay dos tareas en el loop: encontrar el bin, e incrementar en la entrada del histograma
- Si una tarea hace lo primero y otra hace los segundo, sería necesario una comunicación entre ambas



- Para un "i" fijo, estas dos tareas podrían ser agregadas en una sola

Histograma 1D, secuencial

- Pero al mapear estas tareas a diferentes procesadores, nos damos cuenta que dos tareas que están trabajando sobre datos que pertenecen al mismo bin, producirán una condición de carrera.

  `bin_counts[bin]++`
- Un posible solución es particionar `bin_counts[]`, lo cual requerirá comunicación frecuente

Histograma 1D, secuencial

- Otra alternativa es mantener copias locales de `bin_counts[]` en cada tarea
- Agregamos tareas nuevas para actualizar `loc_bin_counts[]` y nuevos canales de comunicación

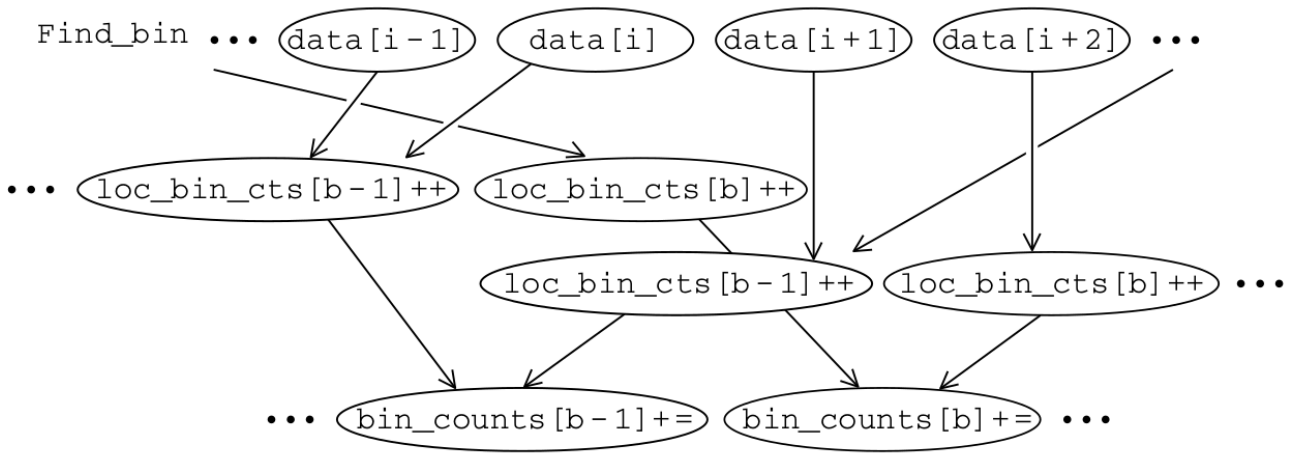


FIGURE 2.22

- Luego, los elementos de `data[]` son asignados en particiones de igual tamaño a las hebras
- Cada hebra construye un histograma parcial en `loc_bin_counts[]`

Histograma 1D, secuencial

- Los histogramas de cada tarea se deben **reducir** en un solo
- Si el número de tareas es pequeño, la suma puede ser realizada por una sola tarea
- Si el número de tareas es mucho más grande que el número de bins, se puede realizar una reducción global basada en árbol binario