3.1 Principios generales de diseño de algoritmos paralelos

Computación de Altas Prestaciones

Grado en Ingeniería de Computadores

Raúl Durán Díaz

Curso académico 2018–2019

Índice

1. Técnicas de descomposición

1.1. Descomposición recursiva

Estrategia «divide y vencerás»

- Se divide un problema en un conjunto de sub-problemas independientes.
- Cada sub-problema se vuelve a dividir recursivamente, hasta alcanzar un tamaño fácilmente resoluble.
- Se combinan los resultados hacia arriba.

Comentario 1. Esta estrategia produce tareas concurrentes de modo natural, pues cada conjunto está formado de subproblemas independientes.

Ejemplo: ***búsqueda recursiva del mínimo***

Consideremos el siguiente algoritmo secuencial:

```
int Serial_min(int *A, int n)
{
int min = A[0], i;

   for (i = 1; i < n; i++)
      if (A[i] < min)
         min = A[i];

   return min;
}
```

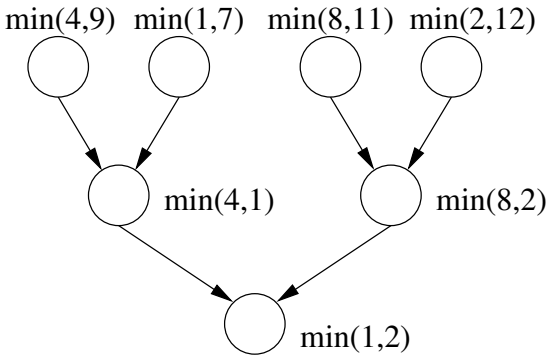
Ejemplo: ***búsqueda recursiva del mínimo***

Usamos la estrategia «divide y vencerás» mediante recursión:

```
int Recursive_min(int *A, int n)
{
int lmin, rmin;

   if (n == 1) return A[0];

   lmin = Recursive_min(A, n/2);
   rmin = Recursive_min(&A[n/2], n - n/2);

   return (lmin < rmin)? lmin:rmin;
}
```

Ejemplo: ***búsqueda recursiva del mínimo***

Supongamos $A = \{4,9,1,7,8,11,2,12\}$. El árbol de dependencias sería:



1.2. Descomposición de datos

Descomposición inducida por los datos

Definición 2. Cuando podemos inducir una descomposición de tareas por medio de la partición de los datos hablamos de **descomposición de datos**.

Tenemos dos fases:

- Examen de los datos (de entrada, de salida, o intermedios) para detectar formas sencillas de particionarlos.
- Descomposición de las tareas, en función de la partición seleccionada.

Descomposición de los datos de salida

En el producto de matrices:

$$\begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \cdot \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix} = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix}$$

podemos inducir una descomposición del cómputo en función de los datos de salida:

$$\begin{array}{ll} \text{Tarea 1:} & C_{11} = A_{11}B_{11} + A_{12}B_{21} \\ \text{Tarea 2:} & C_{12} = A_{11}B_{12} + A_{12}B_{22} \\ \text{Tarea 3:} & C_{21} = A_{21}B_{11} + A_{22}B_{21} \\ \text{Tarea 4:} & C_{22} = A_{21}B_{12} + A_{22}B_{22} \end{array}$$

► Observemos que las tareas son independientes.

(Otra) Descomposición de los datos de salida

Se podría hacer lo mismo con granularidad más fina:

$$\begin{array}{ll} \text{Tarea 1:} & C_{11} = A_{11}B_{11} \\ \text{Tarea 2:} & C_{11} = C_{11} + A_{12}B_{21} \\ \text{Tarea 3:} & C_{12} = A_{11}B_{12} \\ \text{Tarea 4:} & C_{12} = C_{12} + A_{12}B_{22} \\ \text{Tarea 5:} & C_{21} = A_{21}B_{11} \\ \text{Tarea 6:} & C_{21} = C_{21} + A_{22}B_{21} \\ \text{Tarea 7:} & C_{22} = A_{21}B_{12} \\ \text{Tarea 8:} & C_{22} = C_{22} + A_{22}B_{22} \end{array}$$

► Observemos las dependencias que existen entre estas tareas.

Descomposiciones por datos de entrada o intermedios

- Análogamente se pueden generar las tareas observando los datos de entrada o los intermedios.
- Un ejemplo es la búsqueda del número mínimo: no se conoce el resultado con antelación y dividimos el conjunto de números candidatos para generar las tareas.

Definición 3. Las descomposiciones de datos se agrupan bajo la regla **el dueño calcula**. Es una forma gráfica de expresar que cada tarea «es dueña» de los datos sobre los que calcula (partición de datos de entrada) o que calcula (partición de datos de salida) o de ambos.

1.3. Descomposición funcional

Descomposición funcional (I)

Definición 4. La **descomposición funcional** se enfoca sobre las distintas tareas que deben ser realizadas, que deben ser disjuntas.

- Si a cada tarea le podemos asignar sus propios datos, que no se solapan (o muy poco) con los de las demás tareas, hemos tenido éxito en nuestra descomposición.
- Si se solapan, ello implicaría una excesiva comunicación. En este caso, debemos considerar una descomposición por datos.

Descomposición funcional (II)

Comentario 5. La descomposición funcional es útil para estructurar las aplicaciones en forma de módulos que interaccionan a través de interfaces claras y simples. A su vez, cada módulo puede admitir una descomposición por datos. De esta manera, realizamos una descomposición sobre dos ejes.

1.4. Descomposición por exploración

Descomposición por exploración

Definición 6. La **descomposición por exploración** se usa cuando la forma de resolver el problema es explorar exhaustivamente el espacio de soluciones (solución tipo *fuerza bruta*).

Comentario 7. Obsérvese una diferencia fundamental con respecto a la descomposición de datos: mientras que en esta, el cálculo debe realizarse hasta su completa terminación, en la exploración solo una de las tareas encuentra la solución, lo que comunica a las demás para que cesen en su búsqueda.

Descomposición: una *checklist*

- ¿Define mi partición muchas más tareas que procesadores hay? En caso contrario, habrá poco espacio para el paralelismo y falta flexibilidad.
- ¿Evita mi partición cálculos y/o almacenamientos redundantes? En caso contrario, puede haber un problema de escalabilidad.
- ¿Son las tareas de tamaño similar? En caso contrario, tendremos problemas de equilibrado de carga.
- ¿Se escala el número de tareas con el tamaño del problema? Idealmente, al aumentar el tamaño debe aumentar el número de tareas, no su tamaño.

2. Técnicas de asignación

Fuentes de sobrecarga

Objetivos de la asignación
Las tareas han de ser asignadas a procesos de forma que se minimice el tiempo total de ejecución, es decir, las sobrecargas deben reducirse al mínimo.

Fuentes de sobrecarga:

- las esperas causadas por el desequilibrio de carga computacional entre las distintas tareas.

  ⇒ Objetivo: equilibrar las cargas.

- las interacciones entre las tareas, en forma de intercambio de datos o de sincronización;

  ⇒ Objetivo: reducir al mínimo las interacciones.

Tipos de asignaciones

Asignación estática
Consiste en asignar las tareas a procesos antes de la ejecución del algoritmo. Esto solo es posible si se sabe con antelación cuáles son las tareas a ejecutar. Dependiendo del tamaño de las tareas y sus interacciones, se puede llegar a obtener una buena asignación.

Asignación dinámica
Consiste en asignar las tareas durante la ejecución del algoritmo. En este caso conseguir una buena asignación es normalmente más difícil. Si las tareas se generan dinámicamente, han de ser asignadas dinámicamente también.

2.1. Asignación estática

Asignación basada en partición de datos

- En este tipo de descomposición, las tareas son «dueñas» de los datos según la regla «el dueño computa».
- Por tanto, asignar datos a procesos es equivalente, en este caso, a asignar tareas a procesos.

  ▶ Vamos a aplicarlo a matrices, por ser un caso muy frecuente.

Asignación por bloques

Definición 8. Se asignan porciones contiguas de datos a cada proceso. Si la matriz tiene dimensión d, cada proceso recibe una porción contigua de datos a lo largo de alguna de las dimensiones de la matriz.

Ejemplo 9. Si tenemos una matriz $n \times n$, elegimos una dimensión, por ejemplo, las filas, repartimos de modo que cada proceso sea «dueño» de n/p filas. Análogamente se puede hacer por columnas.

Asignación por bloques multidimensionales

Ejemplo 10. Si tenemos una matriz $n \times n$, podemos repartir en bloques de tamaño $n/p_1 \times n/p_2$, de modo que el número total de procesos sea $p = p_1 \times p_2$.

Comentario 11. En general, una matriz de dimensión d se podrá repartir a lo largo de d dimensiones con el efecto de que, cuanto más alta es la dimensión, mayor número de procesos podemos involucrar.

Asignación cíclica

- Se particiona la matriz en muchos bloques, (muchos) más que el número de procesos disponibles.
- Se asignan las particiones a procesos de modo sucesivo (una a cada uno) de modo que cada proceso va recibiendo particiones no contiguas.

Ejemplo 12. En una asignación cíclica unidimensional para p procesos, se divide la matriz en αp grupos de $n/(\alpha p)$ filas (o columnas) que se asignan a los procesos sucesivamente, de modo que el grupo i es asignado al proceso $P_{i \bmod p}$.

Asignación cíclica pura

- Obsérvese que si $\alpha = 1$, entonces cada proceso recibe solo una fila (o columna) que es lo mínimo posible. Hablamos, entonces, de *asignación cíclica pura*.
  - Ventaja: al ser una asignación de grano fino, puede mejorar el equilibrado de carga.
  - Inconveniente: puede resultar en demasiadas interacciones.

> ► *Conclusión*: se debe buscar un valor de α que proporcione un punto medio entre ambos extremos.

Asignación aleatoria

- Si el equilibrado de carga resulta deficiente, se puede realizar una asignación aleatoria de los bloques de datos a procesos, de modo que los tiempos de inacción se repartan alícuotamente entre todos.

Particionado de grafos

- Otros tipos de algoritmos no usan matrices, sino estructuras de datos cuyas interacciones son irregulares.
- Típicamente, en la simulación de procesos físicos necesitamos calcular ciertas magnitudes en puntos discretos del espacio.
- El espacio está discretizado y se representa por una malla donde las aristas representan los puntos vecinos.
- Los cómputos en cada punto involucran los puntos topológicamente próximos a él, lo cual constituye el patrón de interacciones.

Particionado de grafos

- Si suponemos que los cómputos en cada punto son similares, basta con asignar el mismo número de puntos a cada proceso.
- Sin embargo, a cada proceso se le deben asignar los puntos 'próximos', de modo que se minimice la interacción.

Particionado de grafos

- En resumen:
  - dividir el grafo en p partes cada una de ellas con aproximadamente el mismo número de vértices;
  - minimizar el número de aristas que conectan partes pertenecientes a diferentes procesos, lo que equivale a minimizar la interacción.

2.2. Asignación dinámica

Asignación dinámica

- Es importante cuando:
  - la asignación estática implicaría un gran desequilibrio de carga;
  - o las propias tareas se van generando dinámicamente.
- Típicamente se divide en *centralizado* y *distribuido*.

Esquema centralizado

- Un proceso actúa de **máster** y gestiona un *pool* de tareas.
- Cada nueva tarea que aparece se va agregando a ese *pool*.
- Los procesos que actúan como **esclavos** retiran una o más tareas del *pool* y las ejecutan. Al acabar, retira otra u otras, hasta acabar con todas.

► ¡Atención! Posible cuello de botella en el proceso máster.

Esquema distribuido

- Cada proceso puede generar o ejecutar tareas.
- Cada proceso puede enviar/recibir tareas a/de otros.
- Se elimina el problema del cuello de botella.

Esquema distribuido

Estos esquemas son difíciles de gestionar en la práctica.

- ¿Quién inicia las transacciones de tareas, el receptor o el emisor?
- ¿Cuántas tareas se transfieren, de qué tamaño?
- ¿En qué momento se realizan las transacciones?
- ...

3. Aspectos de la interacción

3.1. Patrones de comunicación entre procesos

Patrones típicos de comunicación

- Comunicación local $\Longleftrightarrow$ comunicación global.
- Comunicación estructurada $\Longleftrightarrow$ comunicación no estructurada.
- Comunicación estática $\Longleftrightarrow$ comunicación dinámica.
- Comunicación síncrona $\Longleftrightarrow$ comunicación asíncrona.

Patrón local–global

Comunicación local frente a global
En local, cada tarea se comunica con un pequeño conjunto de otras tareas, sus «vecinos»; por el contrario, en global cada tarea se comunica con muchas otras, tal vez con todas las demás.

Patrón estructurada–no estructurada

Comunicación estructurada frente a no estructurada
Cuando cada tarea forma una estructura regular con sus vecinos (una rejilla, un árbol) hablamos de comunicación estructurada; en caso contrario, no hay estructura, y el patrón de comunicaciones sigue un grafo arbitrario.

Patrón estática–dinámica

Comunicación estática frente a dinámica
En estática, cada tarea se comunica con unos vecinos fijos, que no cambian con el tiempo; en dinámica, las tareas han de comunicarse con diversas tareas, sin seguir un patrón fijo.

Patrón síncrona–asíncrona

Comunicación síncrona frente a asíncrona
En síncrona, emisores y receptores se ejecutan coordinadamente y cooperan en la transmisión; en asíncrona, el receptor necesita datos de modo independiente del emisor y necesitaremos algún buffer intermedio que nos permita ponerlos de acuerdo.

3.2. Métodos para reducir la sobrecarga

¡Reto!

►	**¿Como reducir la sobrecarga en las interacciones?**

Maximizar la localidad de los datos

- Favorecer el uso de datos locales a los procesos, especialmente los más recientemente accedidos (localidad temporal).
- Minimizar el volumen de datos intercambiados. Para ello:
  - minimizar la cantidad de datos compartidos por los procesos;
  - promover el uso de variables intermedias locales.
- Minimizar el número de intercambios, realizando operaciones que agreguen muchas intercomunicaciones sobre una sola. Así se amortiza el coste de arranque de la operación sobre una gran cantidad de datos intercambiados.

Minimizar la contención

- La contención ocurre cuando muchos procesos tratan de acceder al mismo recurso simultáneamente y dicho acceso solo se puede realizar secuencialmente.
  - *ejemplos*: múltiples transmisiones sobre una misma línea; múltiples accesos a la misma posición de memoria; etc.
- La contención opera como una serialización encubierta.

Solapar computación-interacción

- Iniciar la interacción antes de que el dato sea verdaderamente necesario.
- Se necesita soporte hardware y software para que la interacción pueda progresar al tiempo que se sigue realizando cómputo.

Ejemplo 13. En un esquema de asignación dinámica, un proceso esclavo podría pedir más tareas cuando detecta que, en breve, se le va a acabar el que tiene. De esa manera, anticipa la petición que se realiza concurrentemente con la tarea todavía pendiente.

Replicar datos y/o cómputos

- Cuando un dato es necesario en muchos procesos y es de solo lectura, suele ser mejor replicarlo en todos los procesos. Esto implica un mayor coste en ocupación de memoria.
- Análogamente, si muchos o todos los procesos necesitan ciertos datos intermedios, puede ser mejor que todos los calculen frente a que lo haga uno solo y lo transmita a los demás.

Usar operaciones de comunicación colectiva optimizadas

- Difusión colectiva.
- Dispersión/reunión.
- Reducciones.

Solapar interacciones con interacciones

- Si el hardware lo permite, se pueden solapar comunicaciones entre diversos procesos, para obtener así un tiempo total más pequeño.
- Útil, sobre todo, en operaciones de comunicación colectiva.

Comunicaciones: una *checklist*

- ¿Realizan todas las tareas aproximadamente el mismo número de operaciones de comunicación? En caso contrario, tendremos dificultades para escalar el problema.
- ¿Con cuántos vecinos se comunica cada tarea? En general es preferible que sean pocos. En caso contrario, hay que intentar reformular la comunicación global en términos locales.
- ¿Pueden proceder en paralelo las comunicaciones? En caso contrario, podemos tener un problema de ineficiencia y de falta de escalabilidad.
- ¿Permite el patrón de comunicaciones que las tareas se ejecuten realmente en paralelo? En caso contrario, habrá problemas de escalabilidad y de eficiencia y hay que reorganizar el algoritmo.

4. Aspectos del mapeo

Un resumen de posibilidades

- Paralelismo de datos.
- Paralelismo de tareas.
- Modelo trabajo en cadena.
- Modelo cliente-servidor.
- Modelo maestro-esclavo.
- Modelo *pool* de tareas.
- Modelo productor-consumidor.

También se pueden considerar combinaciones de esos modelos.