Projeto de Algoritmos Paralelos

Técnicas de Decomposição

Aula 14
Alessandro L. Koerich

Programação Paralela
Pontifícia Universidade Católica do Paraná (PUCPR)
Ciência da Computação – 6º Período

Programa do PA

Aula Anterior

- Operações básicas de comunicação

Técnicas de Decomposição

- **Decomposição**: É um dos passos fundamentais para resolver um problema em paralelo.
- Descrevemos algumas técnicas de decomposição para alcançar concorrência.
- Uma dada decomposição nem sempre pode levar ao melhor algoritmo paralelo para uma dado problema.

Técnicas de Decomposição

- As técnicas de decomposição podem ser classificadas como:
  - Decomposição recursiva;
  - Decomposição de dados;
  - Decomposição exploratória;
  - Decomposição especulativa;

Técnicas de Decomposição

- As técnicas de decomposição *recursiva* e de *dados* são ditas de "propósito geral"
  → decompõem uma grande variedade de problemas;
- As técnicas de decomposição *exploratória* e *especulativa* são ditas de "propósito especial"
  → classes específicas de problemas;

Decomposição Recursiva

- Método para induzir concorrência em problemas que podem ser resolvidos usando a estratégia *dividir & conquistar*.

1. dividir um problema em um conjunto de subproblemas independentes
2. cada um destes subproblemas é resolvido aplicando-se recursivamente uma divisão similar em subproblemas menores seguidos por uma combinação de seus resultados (ex. 3.4 + fig. 3.8).

Decomposição Recursiva

Example 3.4 Quicksort

Consider the problem of sorting a sequence A of n elements using the commonly used quicksort algorithm. Quicksort is a divide and conquer algorithm that starts by selecting a pivot element x and then partitions the sequence A into two subsequences A_0 and A_1 such that all the elements in A_0 are smaller than x and all the elements in A_1 are greater than or equal to x. This partitioning step forms the *divide* step of the algorithm. Each one of the subsequences A_0 and A_1 is sorted by recursively calling quicksort. Each one of these recursive calls further partitions the sequences. This is illustrated in Figure 3.8 for a sequence of 12 numbers. The recursion terminates when each subsequence contains only a single element. ■

Decomposição Recursiva

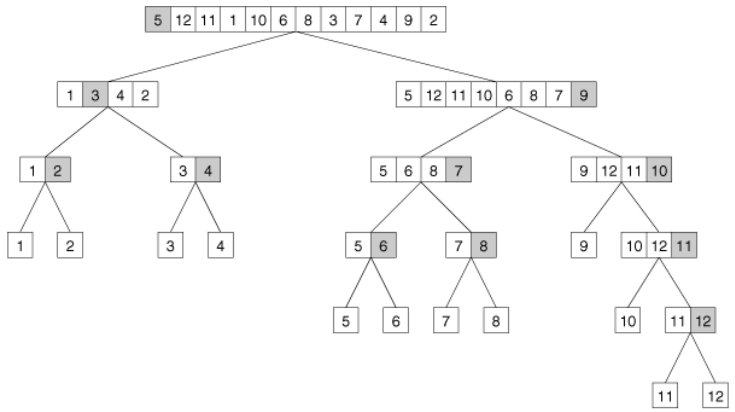

Figure 3.8 The quicksort task-dependency graph based on recursive decomposition for sorting a sequence of 12 numbers.

Decomposição Recursiva

- As vezes, também é possível reestruturar a computação para um problema cujo algoritmo não seja baseado na estratégia dividir e conquistar (alg. 3.1, fig. 3.9).

Decomposição Recursiva

```
1.  procedure SERIAL_MIN (A, n)
2.  begin
3.  min = A[0];
4.  for i := 1 to n − 1 do
5.      if (A[i] < min) min := A[i];
6.  endfor;
7.  return min;
8.  end SERIAL_MIN
```

Algorithm 3.1 A serial program for finding the minimum in an array of numbers A of length n.

```
1.  procedure RECURSIVE_MIN (A, n)
2.  begin
3.  if (n = 1) then
4.      min := A[0];
5.  else
6.      lmin := RECURSIVE_MIN (A, n/2);
7.      rmin := RECURSIVE_MIN (&(A[n/2]), n − n/2);
8.      if (lmin < rmin) then
9.          min := lmin;
10.     else
11.         min := rmin;
12.     endelse;
13. endelse;
14. return min;
15. end RECURSIVE_MIN
```

Algorithm 3.2 A recursive program for finding the minimum in an array of numbers A of length n.

Decomposição Recursiva

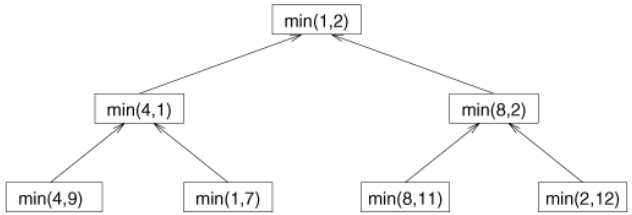

Figure 3.9 The task-dependency graph for finding the minimum number in the sequence {4, 9, 1, 7, 8, 11, 2, 12}. Each node in the tree represents the task of finding the minimum of a pair of numbers.

Decomposição De Dados

- É um método poderoso e normalmente utilizado para produzir concorrência em algoritmos que operam sobre grandes estruturas de dados.
- A decomposição é feita em 2 passos:

  1. os dados sobre os quais a computação é realizada são particionados;

  2. a partição dos dados é utilizada para induzir um partição da computação em tarefas.

Decomposição De Dados

- As operações que estas tarefas realizam sobre diferentes partições dos dados são geralmente similares.
- A partição dos dados pode ser realizada de muitas maneiras diferentes.
- Em geral devemos explorar e avaliar todas as maneiras e determinar qual leva a uma decomposição computacional mais eficiente.

Decomposição De Dados

Partição dos Dados de Saída

- cada elemento da saída pode ser computado independentemente dos outros como uma função das entradas.
- a partição dos dados de saída induz automaticamente uma decomposição dos problemas em tarefas
- a cada tarefa é atribuída o trabalho de computar uma porção da saída (ex. 3.5 3.6 fig. 3.10 3.11 3.12).

Decomposição De Dados

Example 3.5 Matrix multiplication

Consider the problem of multiplying two $n \times n$ matrices A and B to yield a matrix C. Figure 3.10 shows a decomposition of this problem into four tasks. Each matrix is considered to be composed of four blocks or submatrices defined by splitting each dimension of the matrix into half. The four submatrices of C, roughly of size $n/2 \times n/2$ each, are then independently computed by four tasks as the sums of the appropriate products of submatrices of A and B. ■

Decomposição De Dados

$$\begin{pmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{pmatrix} \cdot \begin{pmatrix} B_{1,1} & B_{1,2} \\ B_{2,1} & B_{2,2} \end{pmatrix} \rightarrow \begin{pmatrix} C_{1,1} & C_{1,2} \\ C_{2,1} & C_{2,2} \end{pmatrix}$$

(a)

Task 1: $C_{1,1} = A_{1,1}B_{1,1} + A_{1,2}B_{2,1}$
Task 2: $C_{1,2} = A_{1,1}B_{1,2} + A_{1,2}B_{2,2}$
Task 3: $C_{2,1} = A_{2,1}B_{1,1} + A_{2,2}B_{2,1}$
Task 4: $C_{2,2} = A_{2,1}B_{1,2} + A_{2,2}B_{2,2}$

(b)

Figure 3.10 (a) Partitioning of input and output matrices into 2×2 submatrices. (b) A decomposition of matrix multiplication into four tasks based on the partitioning of the matrices in (a).

Decomposição De Dados

Decomposition I	Decomposition II
Task 1: $C_{1,1} = A_{1,1}B_{1,1}$	Task 1: $C_{1,1} = A_{1,1}B_{1,1}$
Task 2: $C_{1,1} = C_{1,1} + A_{1,2}B_{2,1}$	Task 2: $C_{1,1} = C_{1,1} + A_{1,2}B_{2,1}$
Task 3: $C_{1,2} = A_{1,1}B_{1,2}$	Task 3: $C_{1,2} = A_{1,2}B_{2,2}$
Task 4: $C_{1,2} = C_{1,2} + A_{1,2}B_{2,2}$	Task 4: $C_{1,2} = C_{1,2} + A_{1,1}B_{1,2}$
Task 5: $C_{2,1} = A_{2,1}B_{1,1}$	Task 5: $C_{2,1} = A_{2,2}B_{2,1}$
Task 6: $C_{2,1} = C_{2,1} + A_{2,2}B_{2,1}$	Task 6: $C_{2,1} = C_{2,1} + A_{2,1}B_{1,1}$
Task 7: $C_{2,2} = A_{2,1}B_{1,2}$	Task 7: $C_{2,2} = A_{2,1}B_{1,2}$
Task 8: $C_{2,2} = C_{2,2} + A_{2,2}B_{2,2}$	Task 8: $C_{2,2} = C_{2,2} + A_{2,2}B_{2,2}$

Figure 3.11 Two examples of decomposition of matrix multiplication into eight tasks.

Decomposição De Dados

Example 3.6 Computing frequencies of itemsets in a transaction database
Consider the problem of computing the frequency of a set of itemsets in a transaction database. In this problem we are given a set T containing n transactions and a set I containing m itemsets. Each transaction and itemset contains a small number of items, out of a possible set of items. For example, T could be a grocery stores database of customer sales with each transaction being an individual grocery list of a shopper and each itemset could be a group of items in the store. If the store desires to find out how many customers bought each of the designated groups of items, then it would need to find the number of times that each itemset in I appears in all the transactions; i.e., the number of transactions of which each itemset is a subset of. Figure 3.12(a) shows an example of this type of computation. The database shown in Figure 3.12 consists of 10 transactions, and we are interested in computing the frequency of the eight itemsets shown in the second column. The actual frequencies of these itemsets in the database, which are the output of the frequency-computing program, are shown in the third column. For instance, itemset {D, K} appears twice, once in the second and once in the ninth transaction. ■

Decomposição De Dados

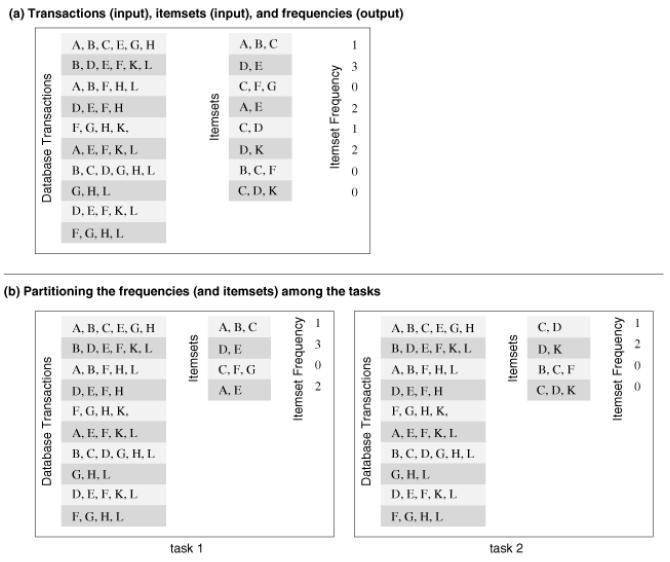

Figure 3.12 Computing itemset frequencies in a transaction database.

Decomposição De Dados

Partição dos Dados de Entrada

- em alguns casos não é possível particionar os dados de saída (Ex. min, máx, ordenação)
- porém, geralmente é possível particionar os dados de entrada e induzir concorrência.
- uma tarefa é criada para cada partição dos dados de entrada e esta tarefa realiza o máximo de computação possível usando estes dados locais (ex. 3.6 fig. 3.13a)

Decomposição De Dados

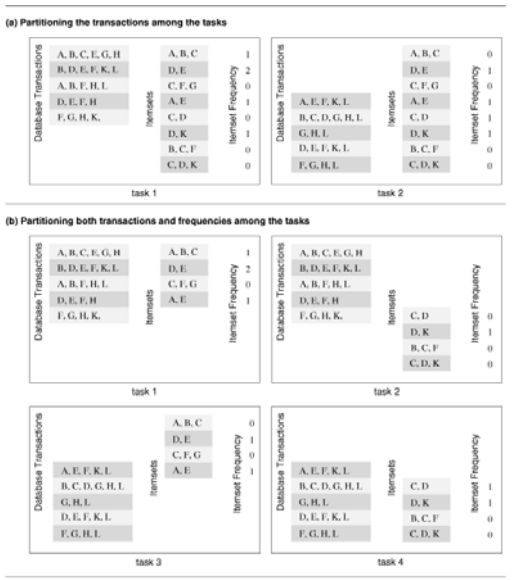

Figure 3.13 Some decompositions for computing itemset frequencies in a transaction database

Decomposição De Dados

Partição dos Dados de Entrada e Saída

- Em alguns casos nos quais é possível particionar os dados de saída, particionando os dados de entrada pode levar a uma concorrência adicional (fig. 3.13b)

Decomposição De Dados

Particionando Dados Intermediários

- algoritmos são geralmente estruturados como computações multi–estágios de modo que a saída de um estágio é a entrada do subseqüente.
- a decomposição pode ser produzida particionando os dados de entrada ou saída de uma estágio intermediário do algoritmo.
- geralmente, os dados intermediários não são gerados explicitamente em um algoritmo seqüencial → exige uma reestruturação do algoritmo (fig. 3.14 3.15 3.16)

Decomposição De Dados

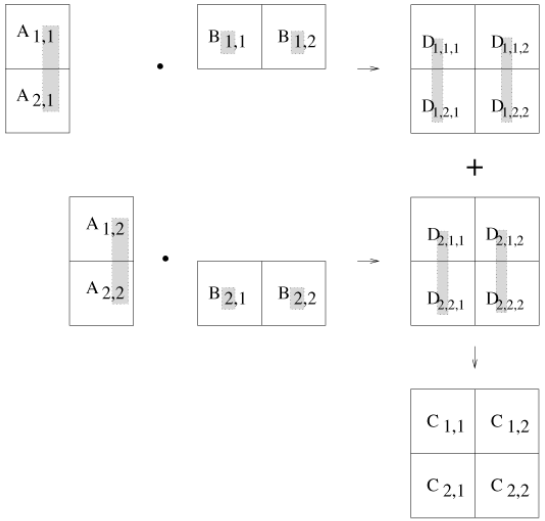

Figure 3.14 Multiplication of matrices A and B with partitioning of the three-dimensional intermediate matrix D.

Decomposição De Dados

Stage I

$$\begin{pmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{pmatrix} \cdot \begin{pmatrix} B_{1,1} & B_{1,2} \\ B_{2,1} & B_{2,2} \end{pmatrix} \rightarrow \begin{pmatrix} \begin{pmatrix} D_{1,1,1} & D_{1,1,2} \\ D_{1,2,2} & D_{1,2,2} \end{pmatrix} \\ \begin{pmatrix} D_{2,1,1} & D_{2,1,2} \\ D_{2,2,2} & D_{2,2,2} \end{pmatrix} \end{pmatrix}$$

Stage II

$$\begin{pmatrix} D_{1,1,1} & D_{1,1,2} \\ D_{1,2,2} & D_{1,2,2} \end{pmatrix} + \begin{pmatrix} D_{2,1,1} & D_{2,1,2} \\ D_{2,2,2} & D_{2,2,2} \end{pmatrix} \rightarrow \begin{pmatrix} C_{1,1} & C_{1,2} \\ C_{2,1} & C_{2,2} \end{pmatrix}$$

A decomposition induced by a partitioning of D

Task 01: $D_{1,1,1} = A_{1,1}B_{1,1}$
Task 02: $D_{2,1,1} = A_{1,2}B_{2,1}$
Task 03: $D_{1,1,2} = A_{1,1}B_{1,2}$
Task 04: $D_{2,1,2} = A_{1,2}B_{2,2}$
Task 05: $D_{1,2,1} = A_{2,1}B_{1,1}$
Task 06: $D_{2,2,1} = A_{2,2}B_{2,1}$
Task 07: $D_{1,2,2} = A_{2,1}B_{1,2}$
Task 08: $D_{2,2,2} = A_{2,2}B_{2,2}$
Task 09: $C_{1,1} = D_{1,1,1} + D_{2,1,1}$
Task 10: $C_{1,2} = D_{1,1,2} + D_{2,1,2}$
Task 11: $C_{2,1} = D_{1,2,1} + D_{2,2,1}$
Task 12: $C_{2,2} = D_{1,2,2} + D_{2,2,2}$

Figure 3.15 A decomposition of matrix multiplication based on partitioning the intermediate three-dimensional matrix.

Decomposição De Dados

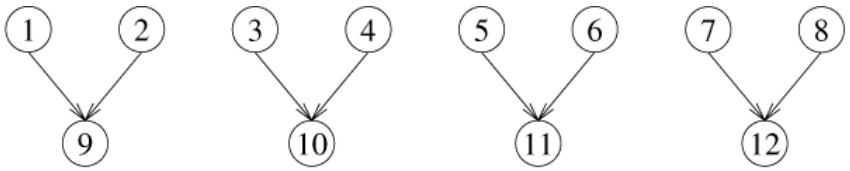

Figure 3.16 The task-dependency graph of the decomposition shown in Figure 3.15.

Decomposição De Dados

Regra do proprietário (owner–computes rule):

- cada partição realiza todas as computações envolvendo os dados de que ele é dono.
- a regra pode ter diferentes significados, dependendo da natureza ou da partição dos dados.
  - partições dos dados de entrada – uma tarefa realiza todas as computações que podem ser realizadas usando estes dados
  - partições dos dados de saída – uma tarefa computa todos os dados na partição atribuída a ele.

Decomposição Exploratória

- É utilizada para decompor problemas cujo a computação fundamental corresponde a uma busca em um espaço de soluções.
- Particionamos os espaço de busca em partes menores e buscamos em cada uma destas partes concorrentemente, até que as soluções desejadas sejam encontradas (ex. 3.7 fig. 3.17 3.18 3.19).

Decomposição Exploratória

Example 3.7 The 15-puzzle problem

The 15-puzzle consists of 15 tiles numbered 1 through 15 and one blank tile placed in a 4×4 grid. A tile can be moved into the blank position from a position adjacent to it, thus creating a blank in the tile's original position. Depending on the configuration of the grid, up to four moves are possible: up, down, left, and right. The initial and final configurations of the tiles are specified. The objective is to determine any sequence or a shortest sequence of moves that transforms the initial configuration to the final configuration. Figure 3.17 illustrates sample initial and final configurations and a sequence of moves leading from the initial configuration to the final configuration. ■

Decomposição Exploratória

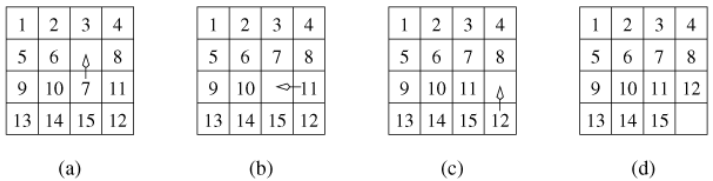

Figure 3.17 A 15-puzzle problem instance showing the initial configuration (a), the final configuration (d), and a sequence of moves leading from the initial to the final configuration.

Decomposição Exploratória

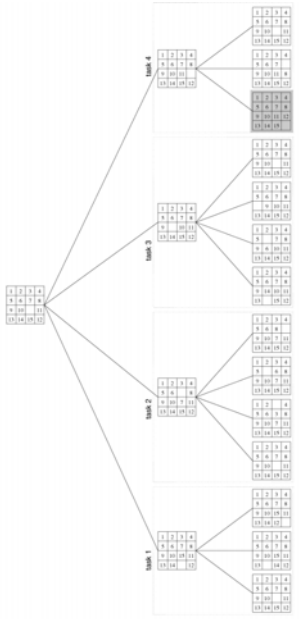

Figure 3.18 The states generated by an instance of the 15-puzzle problem.

Decomposição Exploratória

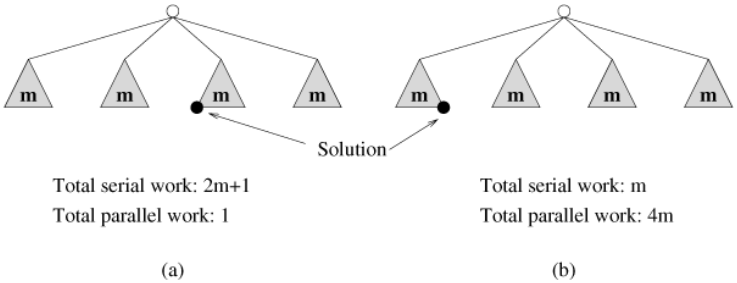

Figure 3.19 An illustration of anomalous speedups resulting from exploratory decomposition.

Decomposição Especulativa

- É usada quando um programa pode aceitar uma ou várias ramificações computacionalmente significantes, dependendo da saída de outras computações precedentes.
- Nesta situação, enquanto uma tarefa está realizando a computação cuja saída será usada para decidir a próxima computação, outras tarefas podem começar concorrentemente a computação do próximo estágio. Ex: Switch (ex. 3.8 fig. 3.20)

Decomposição Especulativa

- Podemos ter um desperdício de computação, pois avaliamos condições que não serão verdadeiras.
- Minimizar o desperdício: computar somente as condições mais prováveis.

Decomposição Especulativa

Example 3.8 Parallel discrete event simulation

Consider the simulation of a system that is represented as a network or a directed graph. The nodes of this network represent components. Each component has an input buffer of jobs. The initial state of each component or node is idle. An idle component picks up a job from its input queue, if there is one, processes that job in some finite amount of time, and puts it in the input buffer of the components which are connected to it by outgoing edges. A component has to wait if the input buffer of one of its outgoing neighbors if full, until that neighbor picks up a job to create space in the buffer. There is a finite number of input job types. The output of a component (and hence the input to the components connected to it) and the time it takes to process a job is a function of the input job. The problem is to simulate the functioning of the network for a given sequence or a set of sequences of input jobs and compute the total completion time and possibly other aspects of system behavior. Figure 3.20 shows a simple network for a discrete event solution problem. ■

Decomposição Especulativa

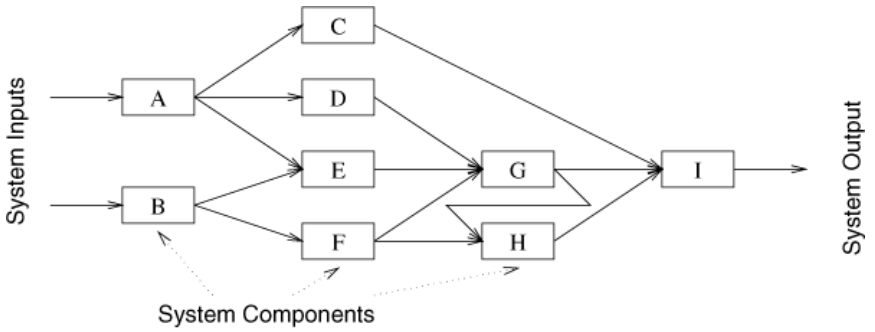

Figure 3.20 A simple network for discrete event simulation.

Resumo: Técnicas de Decomposição

- Os métodos de decomposição descritos permitem:
  - Identificar a concorrência que está disponível em um problema;
  - Decompo–la em tarefas que podem ser executadas em paralelo.
- Os métodos de decomposição vistos, não são exclusivos e podem ser utilizadas em conjunto → *decomposição híbrida* (fig. 3.21)

Resumo: Técnicas de Decomposição

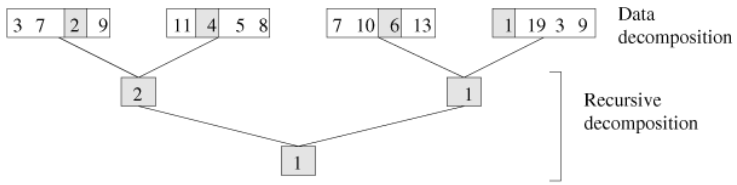

Figure 3.21 Hybrid decomposition for finding the minimum of an array of size 16 using four tasks.