Projetos de Algoritmos Concorrentes

Cap. 3 de “Introduction to Parallel Computing”, 2ed. Edition. Grama-Gupta-Karypis-Kumar

Algoritmos Sequenciais e concorrentes

- Algoritmo Sequencial:
  - Sequência de passos para resolver um problema

- Algoritmo concorrente (definição aproximada) visando obter melhor desempenho:
  - Sequência de passos para resolver um problema +
  - **Decomposição em tarefas**: Decompor os passos em tarefas que possam ser executadas simultaneamente
  - **Mapeamento de tarefas**: Mapear as tarefas simultâneas em processadores
  - **Distribuição dos dados**: Em quais tarefas residem os dados de entrada, saída e intermediários
  - **Sincronização e Comunicação**: Tarefas aguardam o término de outras tarefas e eventuais trocas de dados

Decomposição em Tarefas

- **Dividir a computação em passos menores** (tarefas, *tasks*) algumas das quais possam ser executados concorrentemente (decomposição, *decomposition*)
- Deseja-se ao máximo gerar tarefas independentes
  - Características do problema podem limitar esta característica
  - Exemplos ilustrativos a seguir:
    - Multiplicação de matriz densa A, nxn, por vetor b, nx1, resultando vetor y, nx1
    - Pesquisa (*query*) em um banco de dados

Tarefas concorentes na Multiplicação de Matriz por vetor

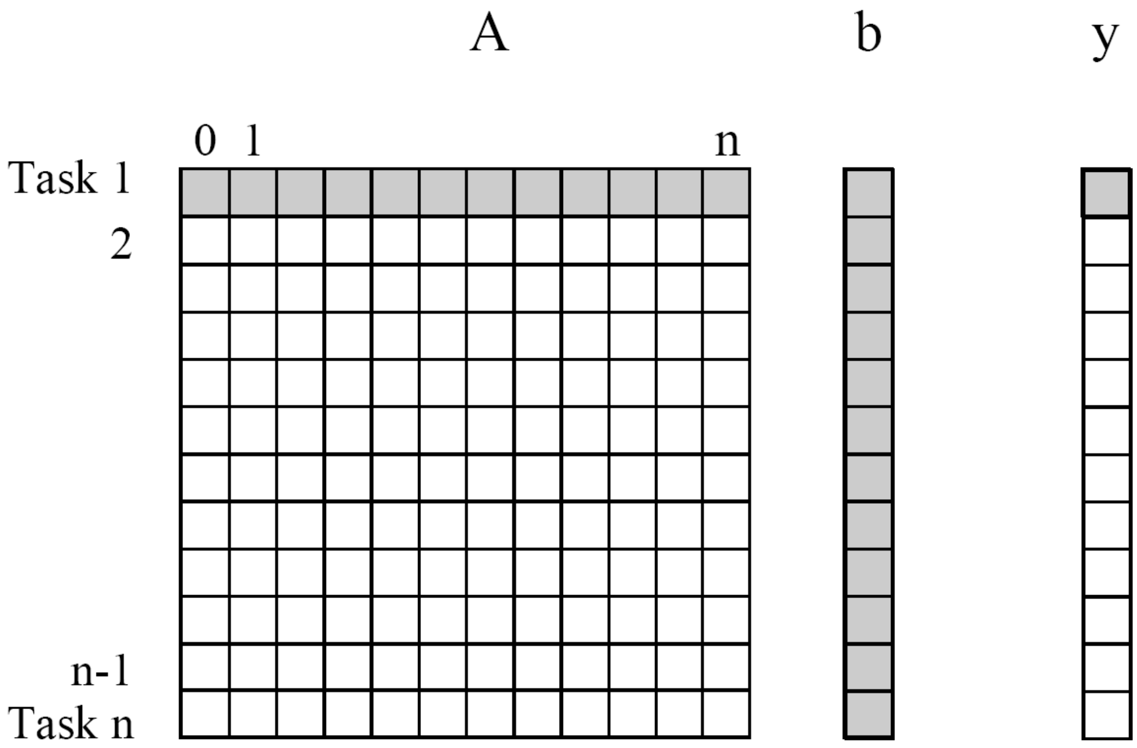


Figure 3.1 Decomposition of dense matrix-vector multiplication into n tasks, where n is the number of rows in the matrix. The portions of the matrix and the input and output vectors accessed by Task 1 are highlighted.

Granularidade e Grau Máximo de Concorrência

- O número máximo de tarefas que podem ser executadas simultaneamente (concorrentemente ou em paralelo) definem o **grau máximo de concorrência** do algoritmo
  - Conforme a forma que dividiu-se as tarefas da multiplicação da matriz por vetor conclui-se que o grau máximo de concorrência é n
- O tamanho (ou carga) da tarefa define o **grão** (*grain*) do algoritmo.
  - O grão é medida relativa, não absoluta. Um algoritmo possui grão grosso (*coarse grain*) ou grão fino (*fine grain*) com relação a outro.
- O algoritmo a seguir possui grão grosso quando comparado ao anterior

Tarefas concorentes na Multiplicação de Matriz por vetor (Grão grosso)

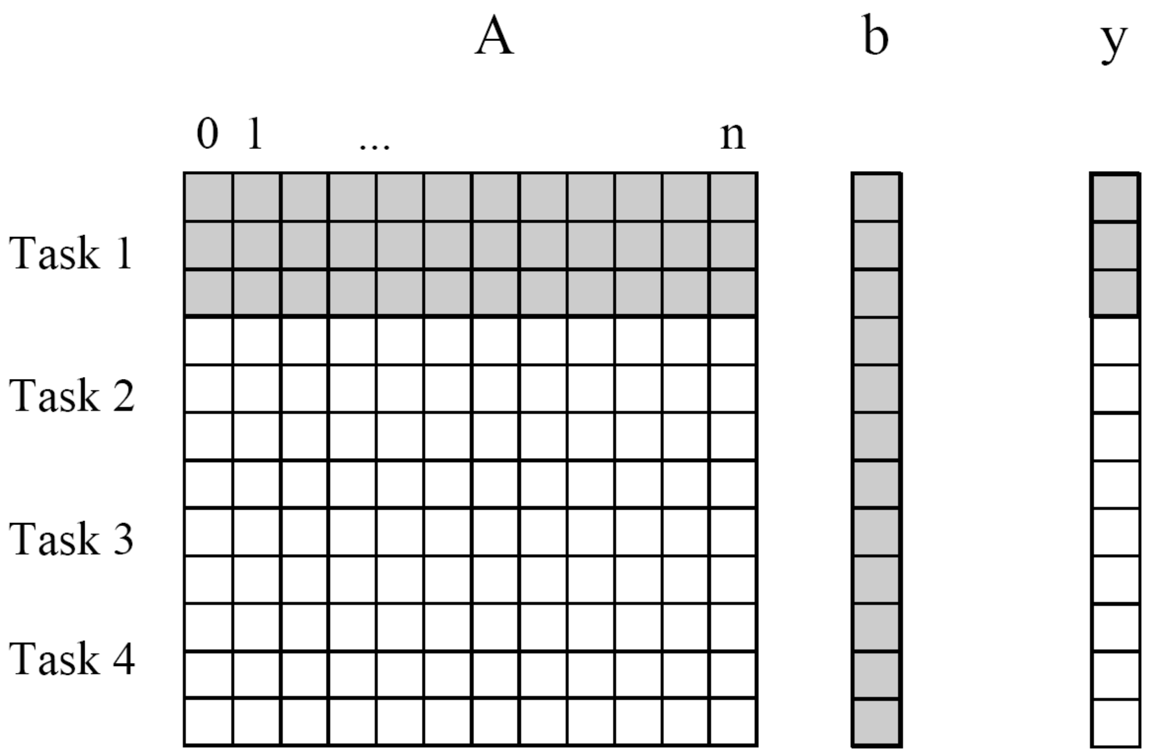


Figure 3.4 Decomposition of dense matrix-vector multiplication into four tasks. The portions of the matrix and the input and output vectors accessed by Task 1 are highlighted.

Reduzir o grão

- Como reduzir o grão da multiplicação de matriz por vetor?
  - O algoritmo abaixo possui o menor grão dentre todos os algoritmos vistos

1. Calcule em paralelo $C_{i,j} = A_{i,j} b_j$ para $i,j=1,\ldots,n$
2. Calcule em paralelo $y_i = \sum_{j=1}^{n} C_{i,j}$ para $i=1,\ldots,n$

- Quais são os graus máximos de concorrência dos passos 1 e 2?
  - $O(n^2)$

Pesquisa em Banco de Dados

Na base de dados abaixo, encontre todos os “Civic” do ano “2001” nas cores “Verde” ou “Branca”

ID#	Model	Year	Color	Dealer	Price
4523	Civic	2002	Blue	MN	$18,000
3476	Corolla	1999	White	IL	$15,000
7623	Camry	2001	Green	NY	$21,000
9834	Prius	2001	Green	CA	$18,000
6734	Civic	2001	White	OR	$17,000
5342	Altima	2001	Green	FL	$19,000
3845	Maxima	2001	Blue	NY	$22,000
8354	Accord	2000	Green	VT	$18,000
4395	Civic	2001	Red	CA	$17,000
7352	Civic	2002	Red	WA	$18,000

Table 3.1 A database storing information about used vehicles.

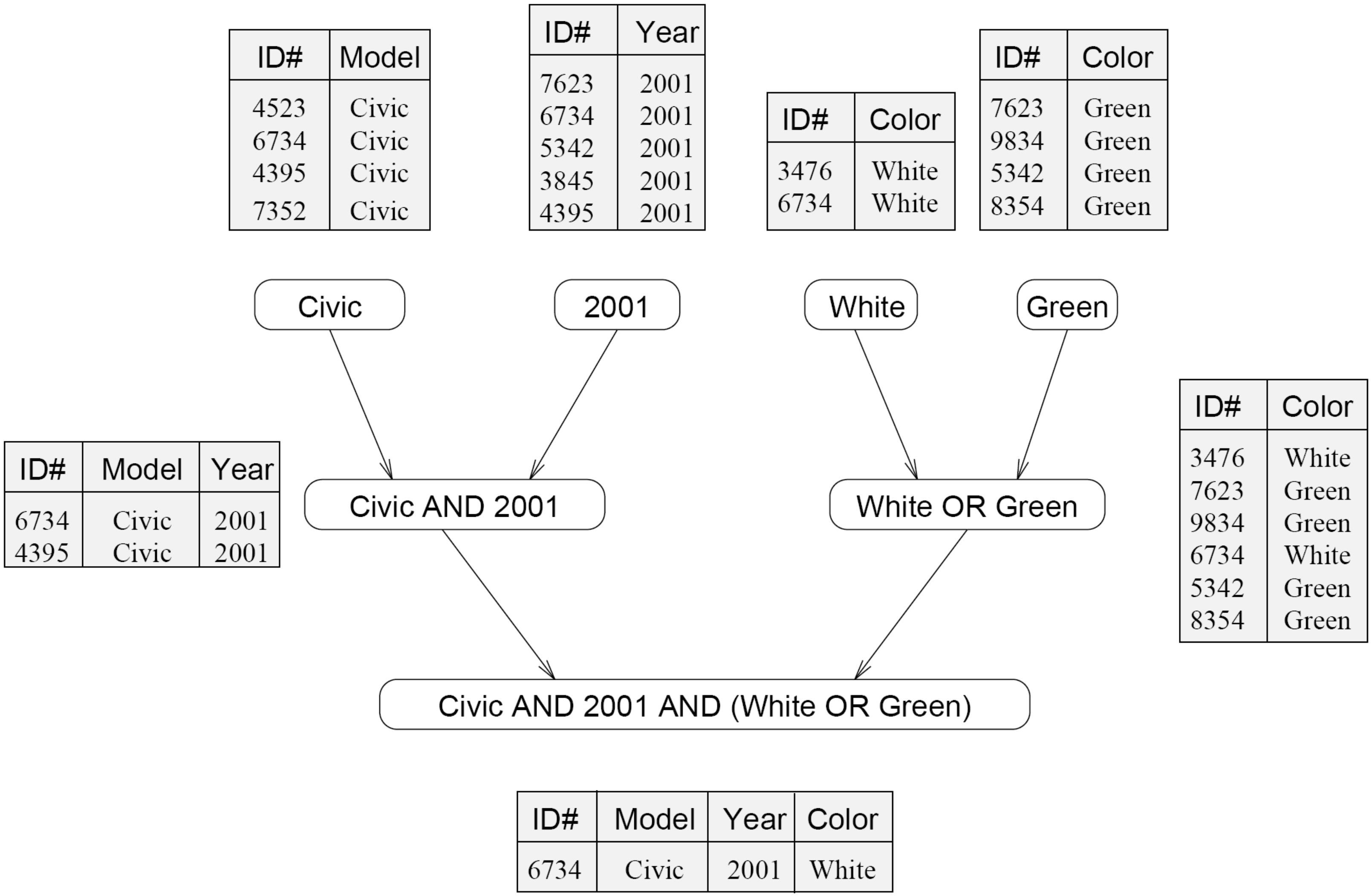
Pesquisa em Banco de Dados (cont.)

- **Passo 1**: Compute as seguintes (sub) tabelas:
  - Encontre todos os “Civic”
  - Encontre todos os carros de 2001
  - Encontre todos os carros verdes
  - Encontre todos os carros brancos
- **Passo 2**: Compute as intersecções e uniões de:
  - “Civic” e “2001” e (“verde” ou “branco”)

Grafo de Dependências de Tarefas

- Grafo dirigido, onde:
  - nós representam tarefas;
  - aresta do nó *i* para o nó *j* representa dependência de execução do nó *j* com relação ao nó *i* (i.e., *i* deve ser executado antes que *j*)
- Um nó pode ser executado *sse* todos os nós incidentes a esse nó já foram executados
- Grafo acíclico dirigido ou DAG (*Direct Acyclic Graph*)

Grafo de Dependências de Tarefas (caso 1)



ID#	Model
4523	Civic
6734	Civic
4395	Civic
7352	Civic

ID#	Year
7623	2001
6734	2001
5342	2001
3845	2001
4395	2001

ID#	Color
3476	White
6734	White

ID#	Color
7623	Green
9834	Green
5342	Green
8354	Green

ID#	Model	Year
6734	Civic	2001
4395	Civic	2001

ID#	Color
3476	White
7623	Green
9834	Green
6734	White
5342	Green
8354	Green

ID#	Model	Year	Color
6734	Civic	2001	White

Figure 3.2 The different tables and their dependencies in a query processing operation.

Grafo de Dependências de Tarefas (caso 2)

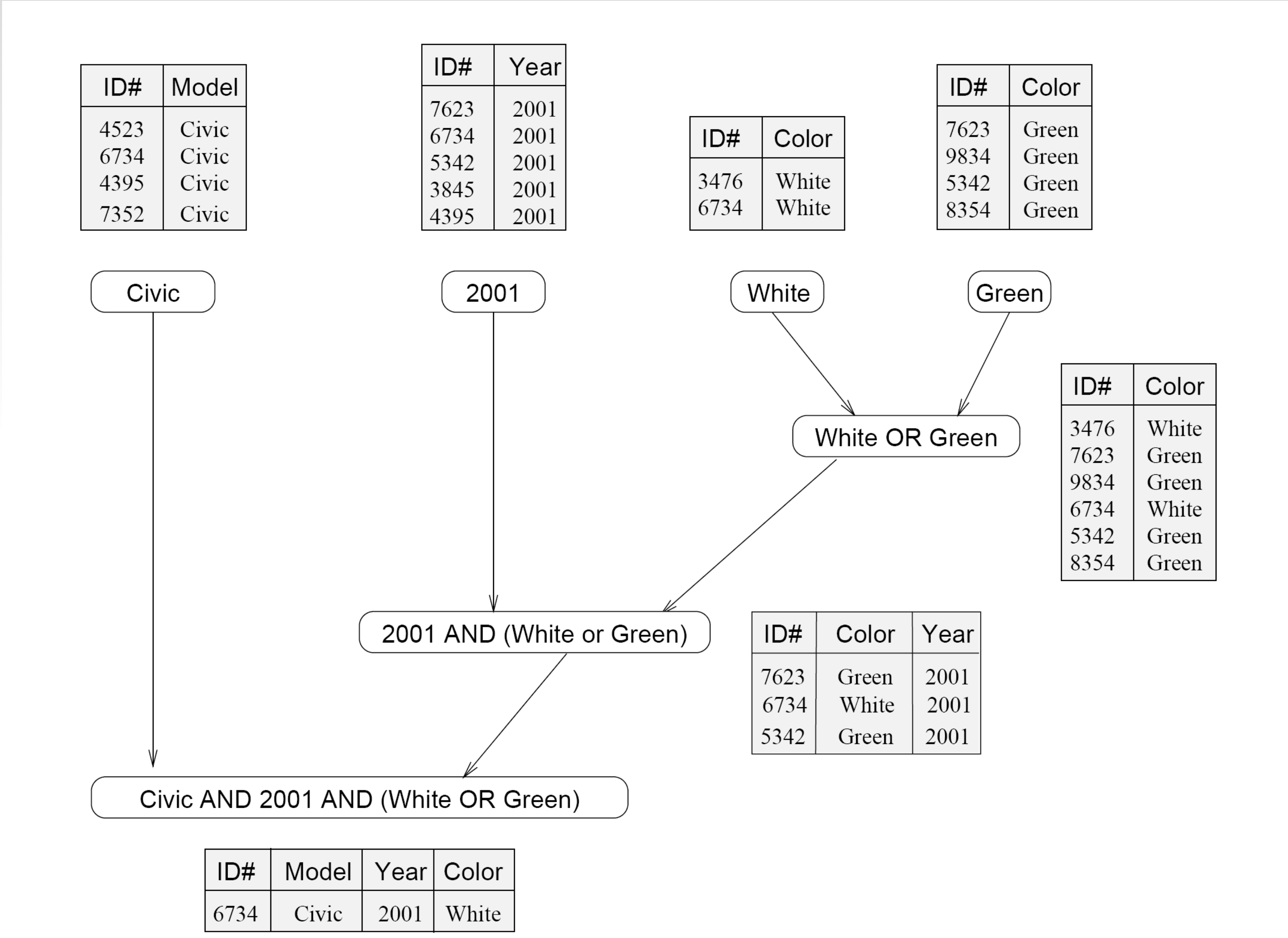


Figure 3.3 An alternate data-dependency graph for the query processing operation.

Grafo de Dependências de Tarefas e Grau Máximo de Concorrência

- Qual é a relação entre o grau máximo de concorrência e a granularidade de um algoritmo?
  - O aumento da granularidade geralmente aumenta o grau máximo de concorrência
  - Não necessariamente aumenta pois aumentar a granularidade pode gerar dependências
- Grau Máximo de Concorrência depende da forma do Grafo de Dependências de Tarefas
  - Grafos “estreitos” e “longos” tem menor grau máximo de concorrência que grafos “largos” e “curtos”
  - Quais são os graus máximos de concorrência das figuras 3.2 e 3.3?
    - 4 nos dois casos.
- Grau Máximo de Concorrência não é bom indicador do paralelismo existente no algoritmo
  - Fornece apenas o limite superior
  - Não fornece por quantos passos o limite se mantém
  - Investiguemos métricas mais acuradas de paralelismo de um algoritmo

Grafo de Dependências de Tarefas Rotulado

- Rotula-se os nós do grafo de dependências de tarefas com inteiros que representem a quantidade de trabalho (carga) no nó (ou tarefa)
  - Aproximação do tempo de execução da tarefa
  - No exemplo do banco de dados, será usado o número de registros de entrada em cada tarefa

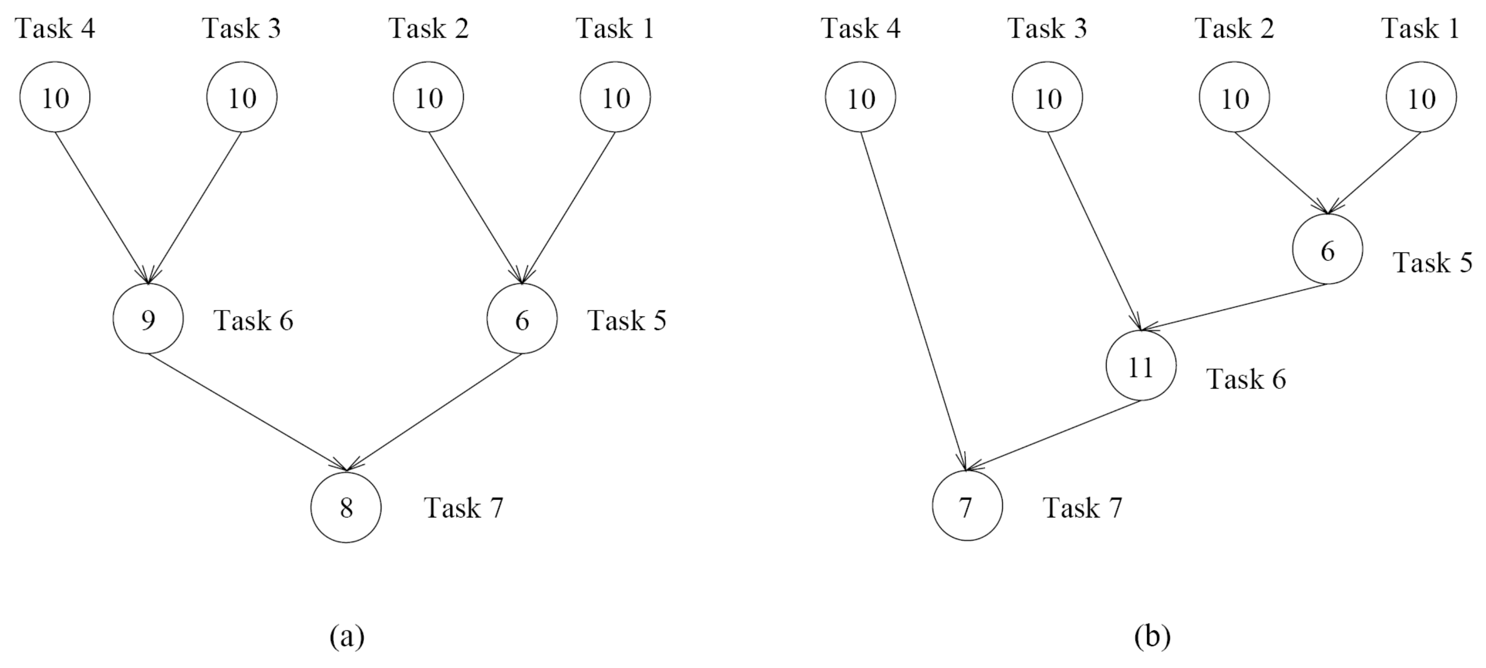


Figure 3.5 Abstractions of the task graphs of Figures 3.2 and 3.3, respectively.

Grau Médio de Concorrência

- Em um grafo de dependências de tarefas com nós rotulados, define-se:
  - **nó inicial** como qualquer nó sem arestas incidentes
  - **nó final** como qualquer nó sem arestas emergentes
  - **comprimento de um caminho** como a soma dos rótulos de todos os nós do caminho
  - **caminho crítico (*critical path*)** como o caminho de maior comprimento entre qualquer nó inicial e qualquer nó final

- Define-se **Grau Médio de Concorrência** como a divisão do trabalho total (soma de todos os rótulos) pelo comprimento do caminho crítico.
  - Aproximação para a quantidade média de paralelismo

Grau Médio de Concorrência

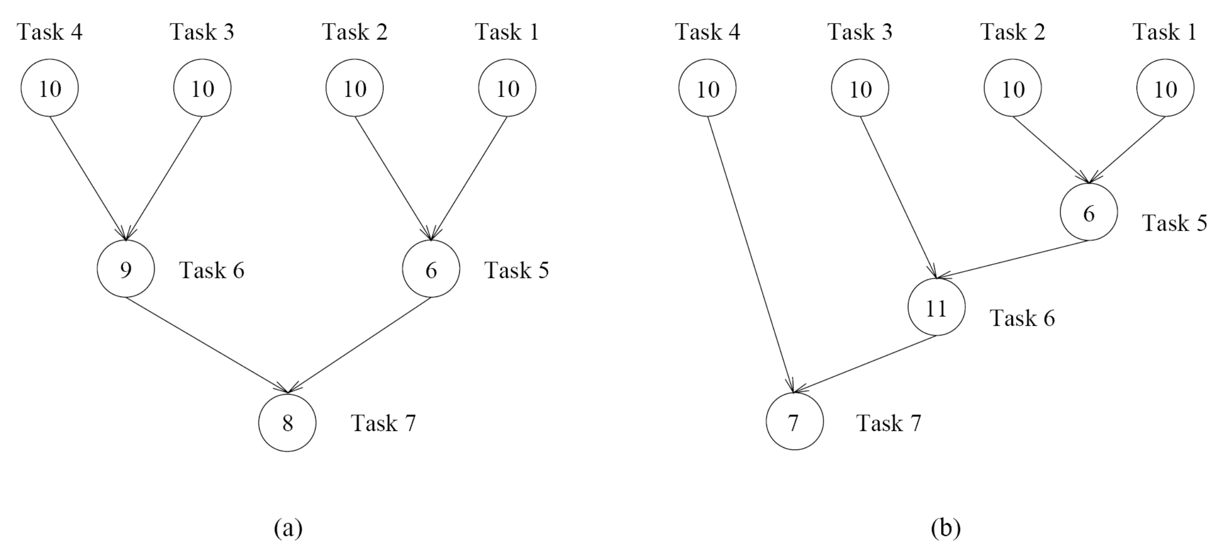


Figure 3.5 Abstractions of the task graphs of Figures 3.2 and 3.3, respectively.

	(a)	(b)
Quantos Caminhos Críticos	2	2
Comprimento do caminho crítico	27	34
Trabalho Total	63	64
Grau Médio de Concorrência	2,33	1,88

Grau Médio de Concorrência

- Supondo um cenário ideal onde:
  - Rótulos do grafo de dependências de tarefas representam o tempo de execução de cada tarefa;
  - Há tantos processadores quanto necessários;
  - Tempo nulo para trafegar dados entre processadores, comunicação entre processadores e para iniciar e terminar uma tarefa;
- Então:
  - Pode-se relacionar o tempo de execução sequencial da tarefa com a soma de todos os rótulos do grafo
    - Válido se o algoritmo paralelo não acrescentar operações
  - Pode-se relacionar o tempo de execução paralelo com o comprimento do caminho crítico
    - O caminho crítico é o maior tempo de execução
  - Pode-se relacionar o grau médio de concorrência com o *speed-up* máximo

Grafo de Interação de Tarefas

- O grafo de dependências de tarefas não contém todas as informações da computação paralela
  - Por exemplo, como as tabelas do banco de dados chegam aos nós iniciais
- Define-se Grafo de Interação de Tarefas como um grafo onde
  - nós representam tarefas
  - arestas representam dados comunicados entre as duas tarefas
  - o grafo pode ser direcionado (se desejamos anotar a direção da troca de dados) ou não
- O Grafo de Interação de Tarefas também é o **Grafo de Dependência de Dados**
- O Grafo de Interação de Tarefas pode ser idêntico ao Grafo de Distribuição de Tarefas ou não

Matriz Esparsa vezes Vetor Denso

- Considere o produto de uma matriz esparsa A por um vetor denso b
  - Cada tarefa computa um elemento de y, o vetor resultante
  - Cada tarefa contem uma linha de A e o elemento de b na mesma linha
  - Os elementos de b trafegam pelas tarefas
  - O grafo de decomposição de tarefas é trivial, mas não o grafo de interações de tarefas

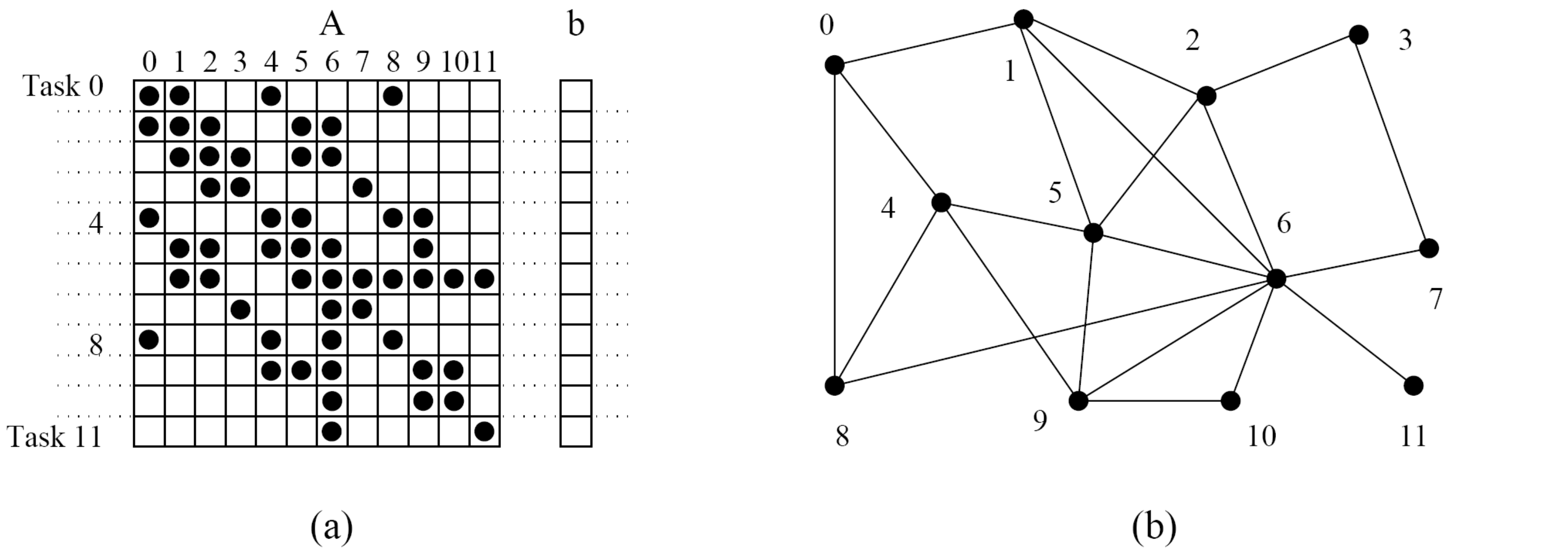


Figure 3.6 A decomposition for sparse matrix-vector multiplication and the corresponding task-interaction graph. In the decomposition Task i computes $\sum_{0\le j\le 11,\, A[i,j]\neq 0} A[i,j].b[j]$.

Matriz Esparsa vezes Vetor Denso (cont.)

- Como a matriz é esparsa, apenas os elementos não nulos devem ser multiplicados
- Os valores de b que não estão na tarefa devem ser requisitados as demais
- O grafo de interações indica as tarefas que compartilham o mesmo elemento de b:
  - Exemplo: a tarefa 4 deve receber b_0, b_5, b_8 e b_9 e enviar b_4 para as tarefas 0, 5, 8 e 9

Mapeamento de Tarefas à Processos

- **Deve-se agregar conjuntos de tarefas em processos (mapeamento)**, onde um processo é um agente computacional que realiza trabalho
- Porque mapear para processos e não para processadores?
  - Tipicamente, sistema operacional "esconde" processadores, mapeando processos em processadores
  - Sobra ao usuário:
    - Agregar tarefas em processos (mapeamento);
    - Deixar o mapeamento de processos para processadores a cargo do sistema operacional

Mapeamento de Tarefas à Processos (cont.)

- Mapear tarefas a processos visa equilibrar objetivos conflitantes:
  - **maximizar tarefas executadas simultaneamente**
    - Atuar sobre o grafo de dependências de tarefas, atribuindo tarefas no mesmo nível do grafo a processos distintos (ordenação topológica)
  - **minimizar comunicação entre tarefas**
    - Atuar sobre o grafo de interação de tarefas, agrupando tarefas que trocam dados no mesmo processo o máximo possível

- Porque conflitantes?
  - Exemplo: para minimizar comunicação entre tarefas, basta atribuir todas as tarefas a um único processo

Ex. de Mapeamentos

- Grafo de Tarefas do "*query*"

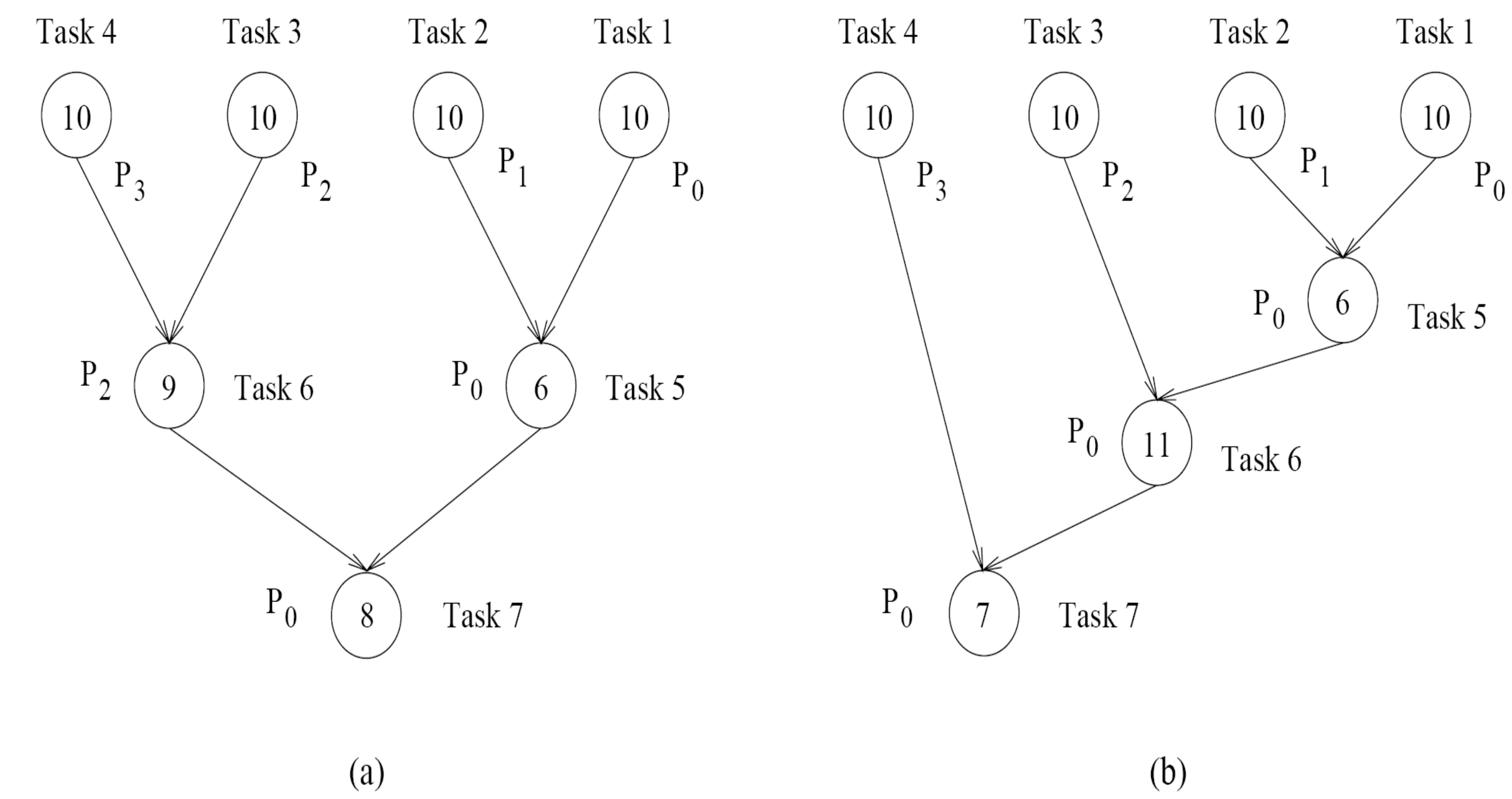


Figure 3.7 Mappings of the task graphs of Figure 3.5 onto four processes.

Ex. de Mapeamentos (cont.)

- As tarefas iniciais podem ser mapeadas em processos distintos arbitrários
- As tarefas seguintes pode ter mapeamento arbitrário, entretanto, preferencialmente deve-se mapear para processos que executam tarefas ligadas por uma aresta
- Exemplo, considerando a figura 3.7(b):
  - A tarefa 5 se for mapeada em P0 irá requerer comunicação entre P0 e P1 apenas
  - Se a mesma tarefa 5 for mapeada em P2, será necessária a comunicação entre P2, P0 e P1

Sumário: Terminologia de Algoritmos Paralelos

- Decomposição de algoritmo paralelo em tarefas
- Granularidade do algoritmo
- Grau máximo de concorrência de um algoritmo paralelo
- Grafo de dependências de tarefas
- Rotular o grafo de dependências com aproximação do tempo de execução
- Caminho crítico no grafo de dependências
- Grau médio de concorrência
- Grafo de interação de tarefas
- Mapeamento de tarefas em processos e de processos em processadores

Técnicas de Decomposição de Tarefas

Cap. 3.2 de Grama-Gupta-Karypis-Kumar

Técnicas de Decomposição mais comuns

- **Objetivo: identificar a concorrência disponível em um problema e decompor o problema em tarefas que podem ser executadas em paralelo**
- Algumas técnicas de decomposição de problemas para gerar algoritmos paralelos:
  - Decomposição Recursiva
  - Decomposição de Dados
  - Decomposição Exploratória
  - Decomposição Especulativa

Decomposição Recursiva

- Aplicar "*divide and conquer*" recursivamente
  - Divide o problema em conjuntos de subproblemas menores, independentes e similares
  - Aplica novamente a técnica a cada subproblema até ser trivial
  - Concorrência na solução simultânea dos subproblemas
- Ex: Quicksort
  - Ordenar sequência A de n elementos
  - Quicksort: selecione elemento de A (pivô, representado por x). Divida A em duas sequências, A_0 e A_1, tais que A_0 contém todos os elementos de $A<x$ e A_1 contém todos os elementos de $A \geq x$. Aplique quicksort recursivamente para A_0 e A_1.
  - Cada aplicação de quicksort a cada sequência é uma tarefa independente das outras sequências. A recursão gera múltiplas tarefas concorrentes.

Decomposição Recursiva para o algoritmo *Quicksort*

- Grafo de dependências de tarefas em *quicksort*

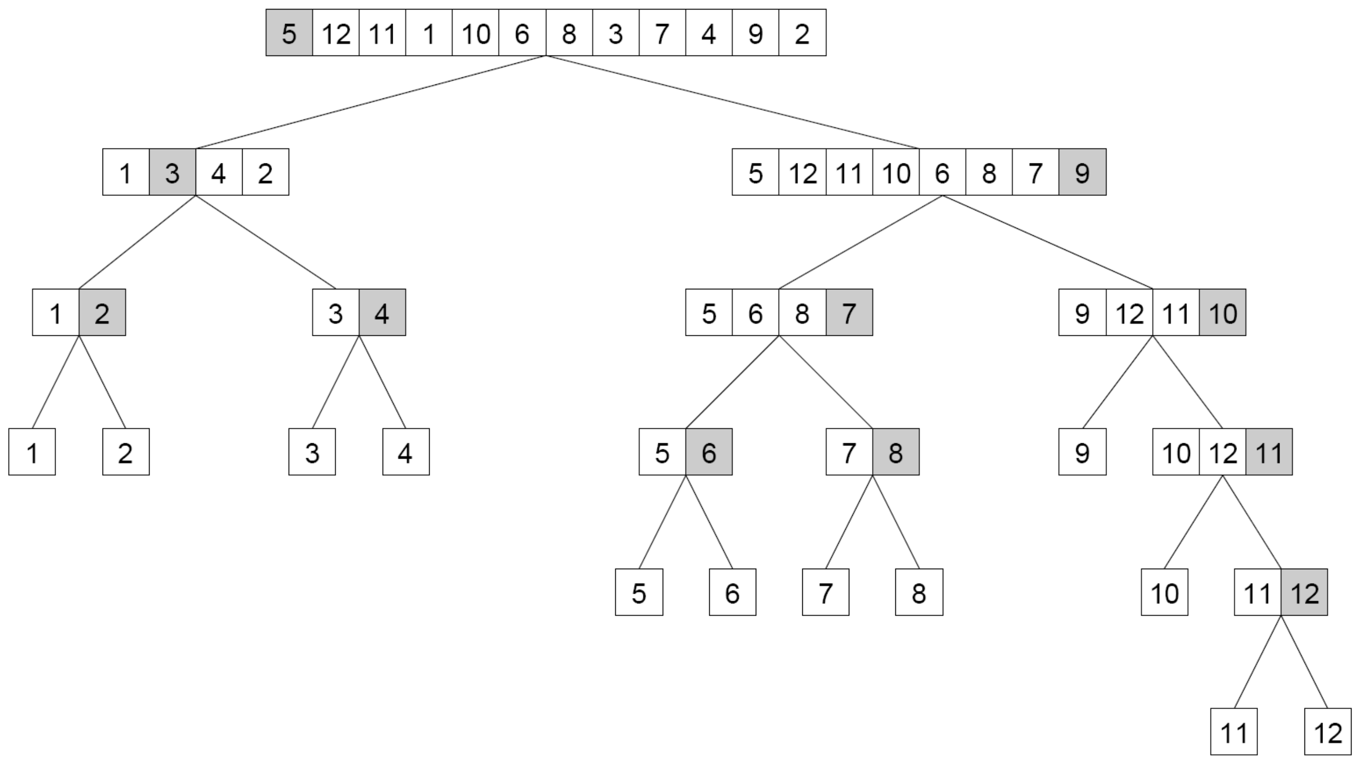


Figure 3.8 The quicksort task-dependency graph based on recursive decomposition for sorting a sequence of 12 numbers.

Balanceamento de carga da Decomposição Recursiva

- Grafo de dependências não balanceado
  - Decomposição recursiva não necessariamente gera grafos balanceados, mas há casos em que gera

- Ex: Mínimo de um conjunto de *n* números
  - Algoritmo sequencial transformado em paralelo por decomposição recursiva
  - Se $n=1$, retorna o único valor possível
  - Se $n>1$, retorna o mínimo entre os $n/2$ primeiros números e os $n-n/2$ últimos números
  - Concorrência na execução simultânea dos subproblemas de mesmo nível no grafo de dependências de tarefas
  - Grafo balanceado se *n* for potência de base 2

Decomposição Recursiva: Mínimo

```
procedure SERIAL_MIN (A, n)
begin
min = A[0];
for i := 1 to n − 1 do
    if (A[i] < min) min := A[i];
endfor;
return min;
end SERIAL_MIN
```

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

```
procedure RECURSIVE_MIN (A, n)
begin
if (n = 1) then
    min := A[0];
else
    lmin := RECURSIVE_MIN (A, n/2);
    rmin := RECURSIVE_MIN (&(A[n/2]), n − n/2);
    if (lmin < rmin) then
        min := lmin;
    else
        min := rmin;
    endelse;
endelse;
return min;
end RECURSIVE_MIN
```

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

Decomposição Recursiva: mínimo (cont.)

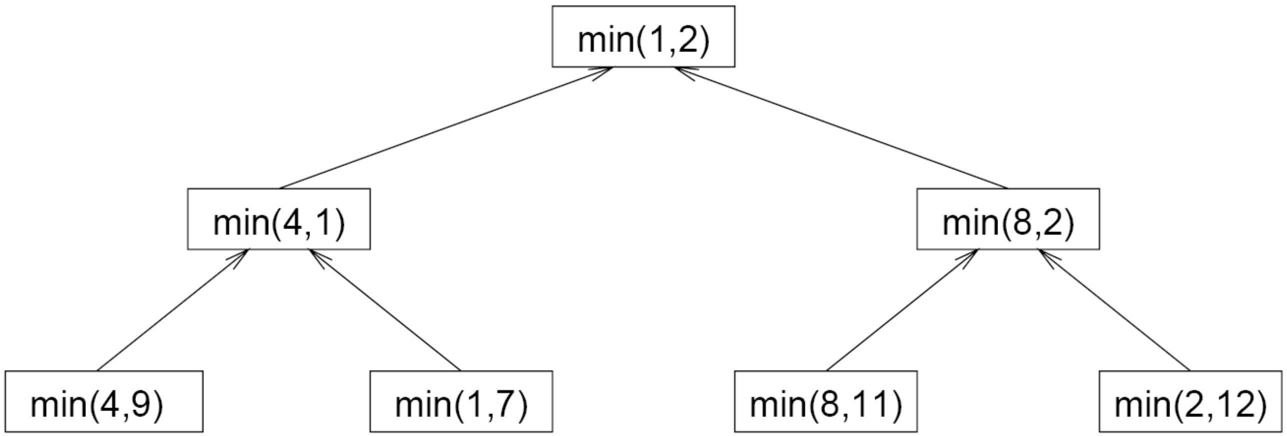


Figure 3.9 The task-dependency graph for finding the minimum number in the sequence {4, 9, 1, 7, 8, 11, 2, 12}. Each node in the tree represents the task of finding the minimum of a pair of numbers.

Decomposição de Dados

- Decomponha os dados pelas tarefas; em seguida, decomponha a computação em tarefas de acordo com a posição de dados
  - *Data decomposition* é técnica exaustivamente utilizada, principalmente em máquinas de memória distribuída
  - Também conhecido como decomposição do domínio (*domain decomposition*) do problema
    - Quais dados decompor?
      - Domínio de Entrada
      - Domínio de Saída
      - Domínios Intermediários
    - Tipicamente, requer a replicação dos outros domínios
- Ex: Mínimo de um conjunto de n números decompondo a entrada
  - Atribua pares de números a uma tarefa
  - Cada tarefa encontra mínimo dos seus dados;
  - Recorra no problema com tamanho $n/2$;

Decompor a Saída

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

(a)

$$\begin{aligned}
\text{Task 1: } C_{1,1} &= A_{1,1}B_{1,1} + A_{1,2}B_{2,1} \\
\text{Task 2: } C_{1,2} &= A_{1,1}B_{1,2} + A_{1,2}B_{2,2} \\
\text{Task 3: } C_{2,1} &= A_{2,1}B_{1,1} + A_{2,2}B_{2,1} \\
\text{Task 4: } C_{2,2} &= A_{2,1}B_{1,2} + A_{2,2}B_{2,2}
\end{aligned}$$

(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).

- Grau máximo de concorrência = 4

Decompor Dados Intermediários

- No produto de matrizes, define-se $D_{k,i,j} = A_{i,k}\, B_{k,j}$;
- Decomponha em D
- Então $C_{i,j} = \text{soma}(D_{*,i,j})$

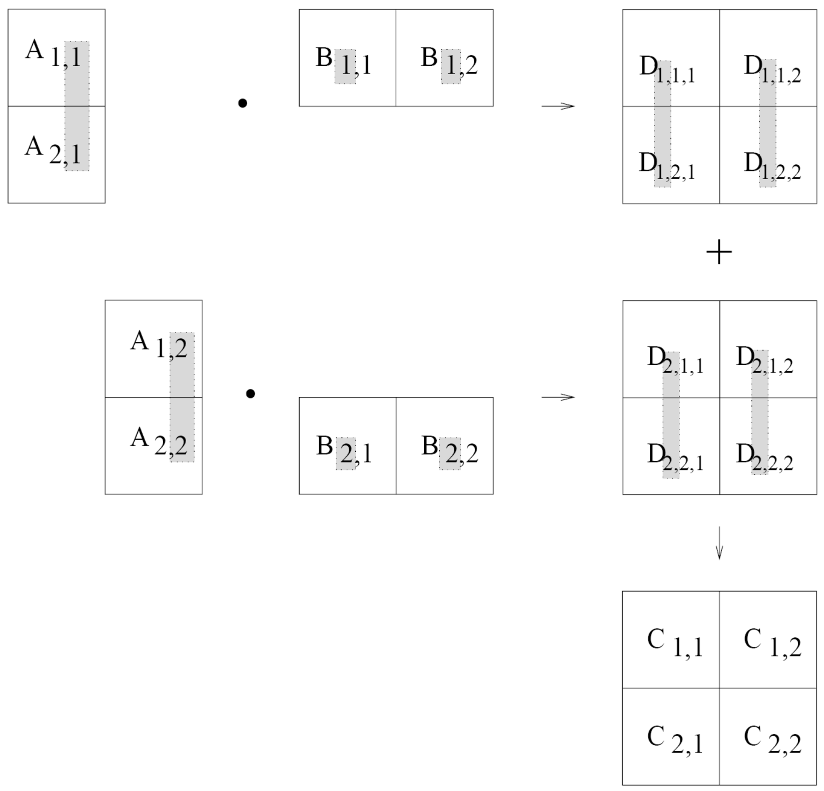


Figure 3.14 Multiplication of matrices A and B with partitioning of the three-dimensional intermediate matrix D.

- Grau máximo de concorrência = 8

Decompor Dados Intermediários (cont.)

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.

Decompor Dados Intermediários
Dependências de tarefas

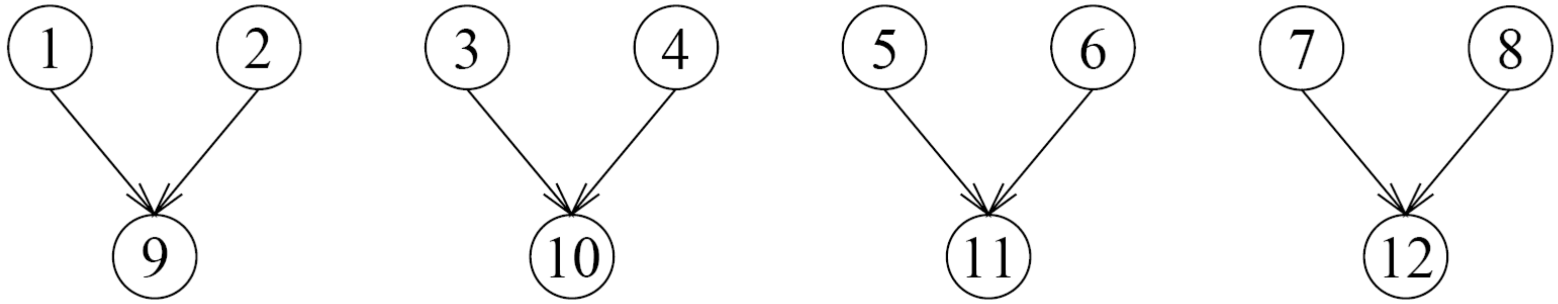


Figure 3.16 The task-dependency graph of the decomposition shown in Figure 3.15.

Owner Computes

- Técnica extensivamente utilizada em decomposição de dados
  - Tanto para decompor domínio de entrada, saída ou intermediário
- Cada tarefa realiza as computações sobre os dados de sua partição
  - Quem é “dono” dos dados computa sobre eles
- Ex: As duas decomposições do produto de matrizes utilizam “*owner computes*”

Decomposição Exploratória

- Gerar paralelismo em um espaço de busca
  - Encontrar uma trajetória em um espaço de busca de uma posição inicial para uma posição final
  - A partir de um estado inicial, geram-se estados intermediários que por sua vez geram outros estados intermediários e assim sucessivamente até que um dos estados intermediários seja o estado final desejado
  - Cada estado intermediário é uma nova tarefa, que gera outras tarefas e assim sucessivamente. Ao encontrar o estado final, interrompe-se a geração de tarefas

- Ex: Quebra cabeças de 15 posições

Decomposição Exploratória: exemplo

- Quebra cabeças de 15 posições
  - Considere uma grade 4x4 com uma posição vazia e as demais posições numeradas de 1 a 15. A posição vazia pode ser trocada por uma das quatro posições adjacentes. Dada uma configuração inicial e uma final, encontrar uma sequência de movimentos da posição vazia que leve da configuração inicial à final, se isso for possível

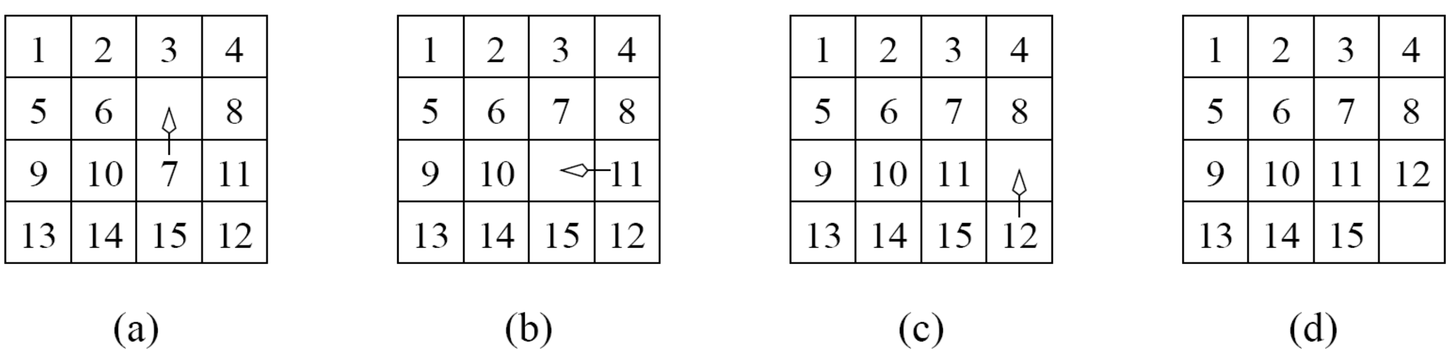


Figure 3.17 A 15-puzzle problem instance showing the initial configuration (a), the final configuration (d), and a sequence of moves leading from the initial to the final configuration.

Algoritmo sequencial

- Dada a configuração inicial:
  - Gere todas as configurações possíveis movendo a posição vazia para cada uma das posições adjacentes possíveis (podem haver 4, 3 ou 2 descendentes)
  - Há uma nova configuração que está um movimento mais perto da configuração final, se o problema possuir solução. Recorra.

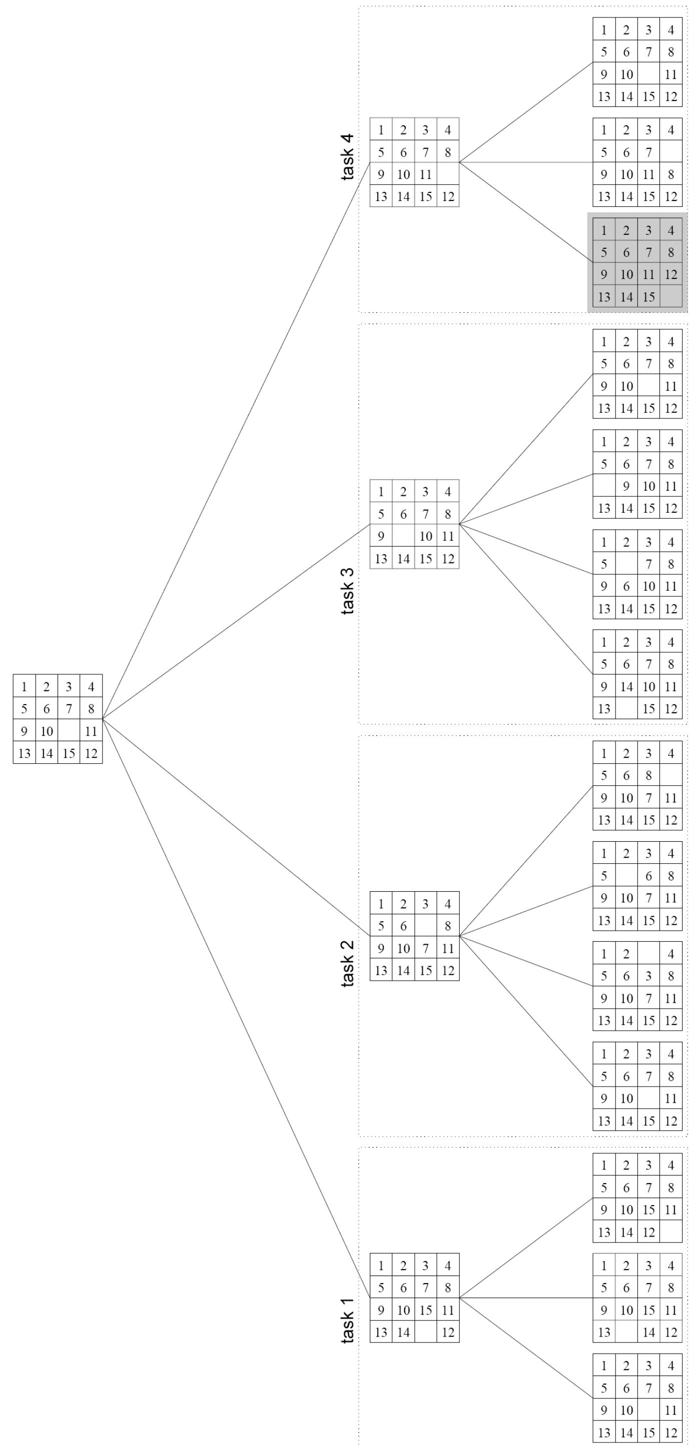


Figure 3.18 The states generated by an instance of the 15-puzzle problem.

Algoritmo paralelo

- Se a configuração inicial for idêntica à final, nada a fazer.

- Caso contrário, gere, sequencialmente, todas as configurações possíveis a partir da configuração inicial. Continue gerando (e testando o término) até gerar número de configurações idêntico ao número de tarefas desejada.

- Atribua uma tarefa para cada configuração gerada. Trabalhe sequencialmente em cada tarefa, gerando as configurações para a própria tarefa e testando o término. Quando uma tarefa atingir a configuração final, avisa as outras.

Decomposição Especulativa

- Gerar paralelismo em um *switch*, antes de conhecer qual *branch* tomar
  - Dispara concorrentemente uma tarefa para cada *branch* do *switch*
    - Ao conhecer qual *branch* tomar, cancela a execução dos outros *branch*

- Uso mais conhecido: execução especulativa de instruções em uma CPU
  - Por ex, executar especulativamente os dois *branch* de um *if statement* antes do término da execução do *if*

Sumário: Técnicas de Decomposição de Problemas para gerar concorrência

- Decomposição Recursiva
- Decomposição de dados
  - Também conhecida como *domain decomposition*
  - Decomposição do domínio de entrada, de saída ou intermediário
  - *Owner Computes* é princípio aplicado a qualquer decomposição de dados
- Decomposição Exploratória
- Decomposição Especulativa