Projeto de Algoritmos Paralelos

Computação Paralela

Prof. Mário O. de Menezes

Introdução a Algoritmos Paralelos

Algoritmo Paralelo

Passos Típicos para se construir um algoritmo paralelo:

- identificar quais pedaços do trabalho podem ser feitas concorrentemente.
- particionar o trabalho concorrente em processadores independentes
- distribuir a entrada do programa, a saída e os dados intermediários
- coordenar os acessos aos dados compartilhados: evitar conflitos
- garantir ordem correta do trabalho utilizando sincronização

Por que “típicos”? Alguns dos passos podem ser omitidos

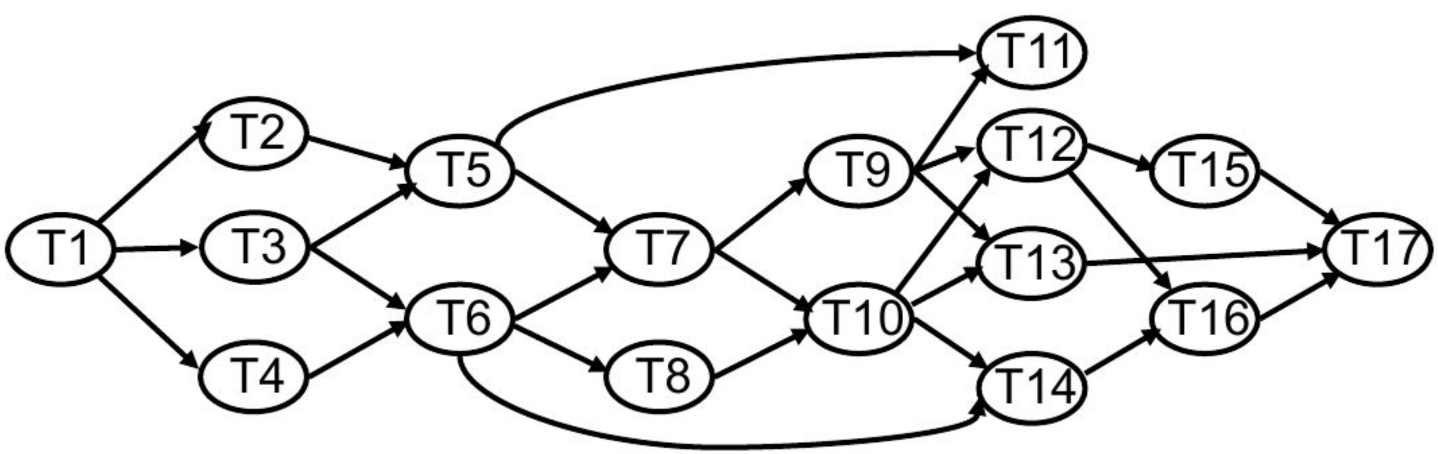
- se os dados estão em memória compartilhada, não é necessário distribuí-los
- se for utilizar MPI, pode não haver dados compartilhados
- o mapeamento do trabalho nos processadores pode ser feito estaticamente pelo programador ou dinamicamente pelo *runtime*.

Nesta aula

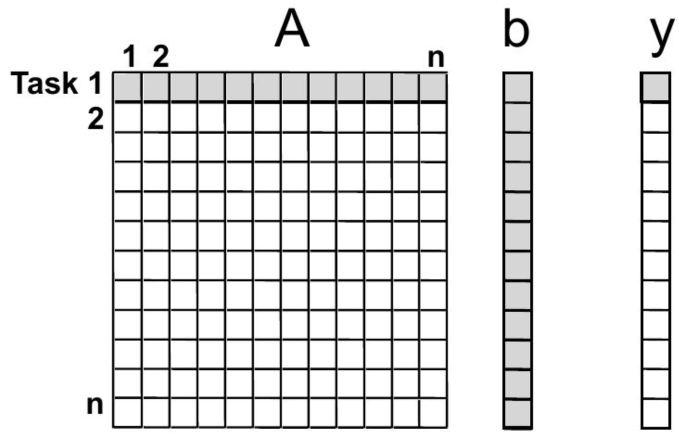
- Introdução a algoritmos paralelos
    - decomposição de tarefas
    - processos e mapeamento
    - processos *versus* processadores
- Técnicas de decomposição - parte 1
    - decomposição recursiva
    - decomposição de dados

Decompondo o trabalho para execução paralela

- Dividir o trabalho em tarefas que podem ser executadas concorrentemente
- Muitas decomposições diferentes são possíveis para qualquer computação
- Tarefas podem ter o mesmo tamanho, tamanhos diferentes ou tamanhos indeterminados
- Tarefas podem ser independentes ou ter uma ordem não trivial
- Conceitualizamos as tarefas e a ordem de execução como um **Grafo Acíclico Direcionado (DAG)** de *dependência de tarefas*
  - nós = tarefas
  - arestas = controle de dependência



Exemplo: Multiplicação Matriz-Vetor (Matriz Densa)



- Computação de cada elemento do vetor resultado **y** é independente
- Fácil decompor o produto da matriz densa pelo vetor em tarefas:
  - uma por elemento de **y**
- Observações
  - tamanho da tarefa é uniforme
  - sem dependência de controle entre as tarefas
  - tarefas compartilham ***b***

Questão: Será *n* o número máximo de tarefas possíveis?

Exemplo: Processamento de uma consulta de BD

- Considere a execução da seguinte query:

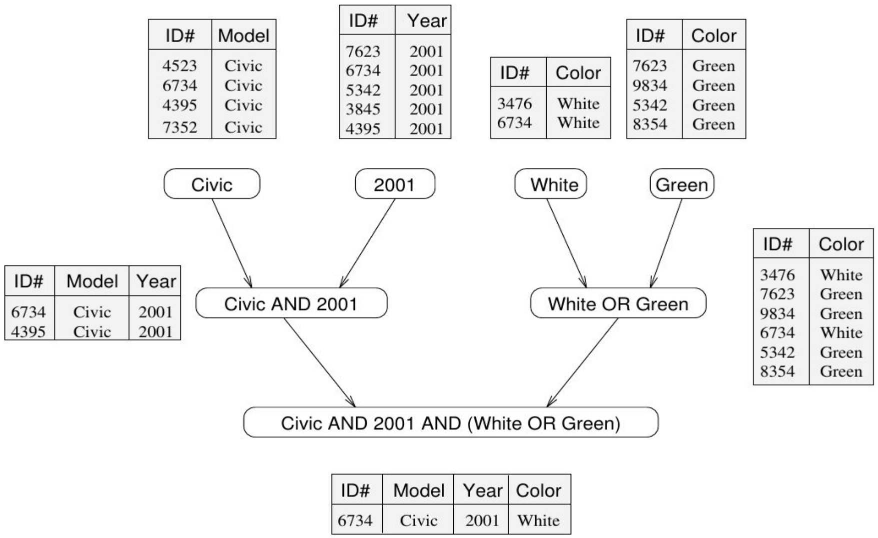
```
MODEL = "CIVIC" AND YEAR = 2001 AND (COLOR = "GREEN" OR COLOR = "WHITE")
```

no seguinte banco de dados:

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

Exemplo: Processamento de uma consulta a BD

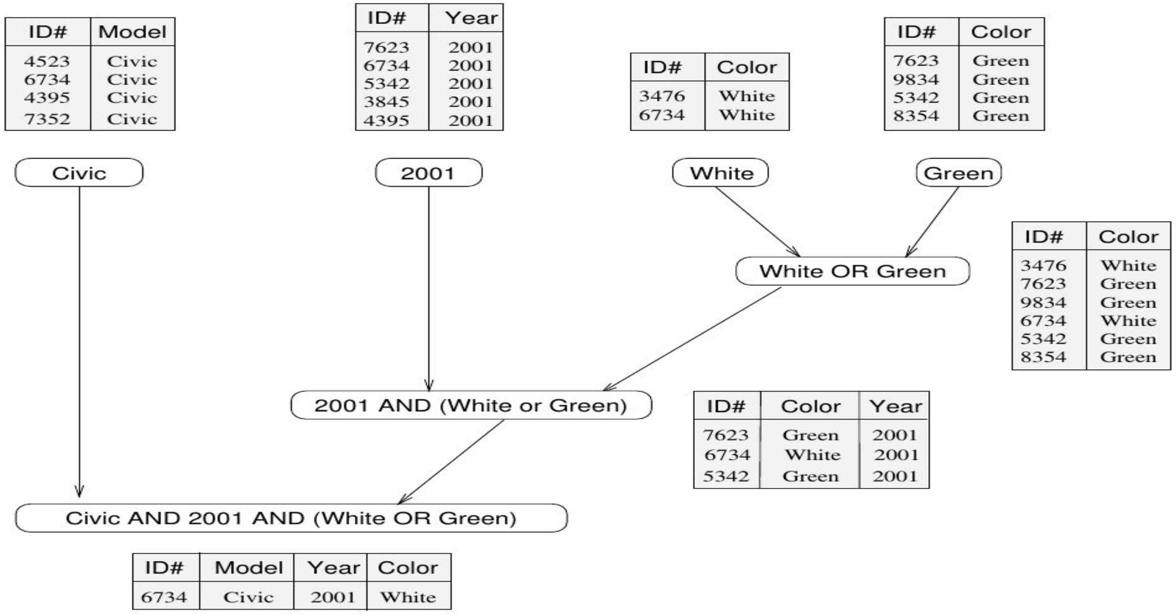
A execução de uma query pode ser dividida em subtarefas de várias maneiras. Cada tarefa pode ser pensada como a geração de uma tabela intermediária de entradas que satisfazem uma cláusula particular.



Decomposição da query dada em um determinado número de tarefas. Arestas neste grafo denotam que a saída de uma tarefa é necessária para completar a próxima

Exemplo: Processamento de uma consulta de BD

O mesmo problema pode ser decomposto de outras maneiras em subtarefas.

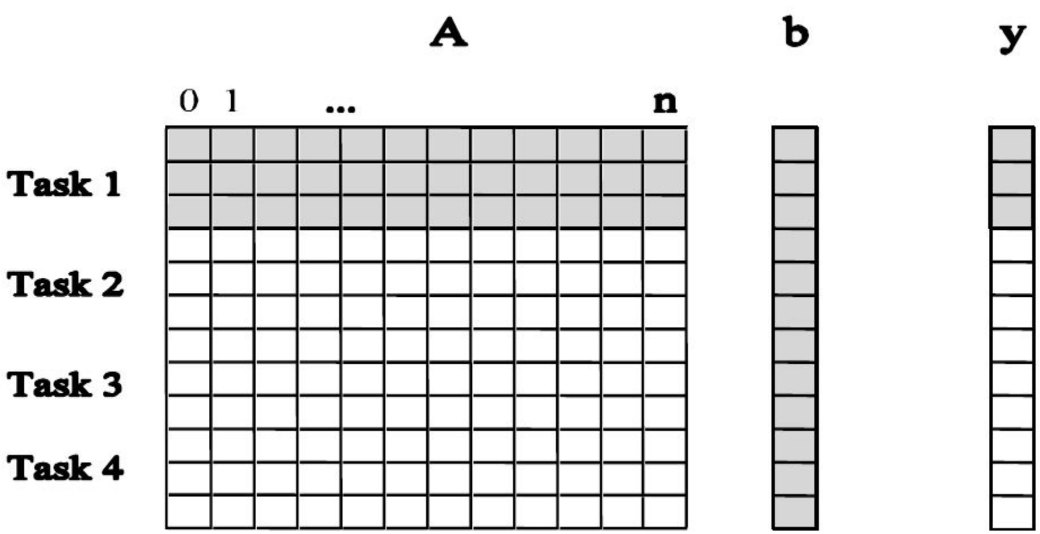


Uma maneira alternativa de decompor o problema dado em subtarefas, com as dependências de dados.

Decomposições diferentes podem levar a diferenças significativas com relação ao seu desempenho paralelo.

Granularidade da Decomposição das Tarefas

- Granularidade: número de tarefas em que um problema é decomposto
- *Fine-grain* = grande número de tarefas
- *Coarse-grain* = pequeno número de tarefas
- Exemplos de granularidade para a multiplicação da matriz densa pelo vetor:
  - fine-grain: cada tarefa representa um elemento individual em **y**
  - coarser-grain: cada tarefa computa 3 elementos em **y**



Grau de Concorrência

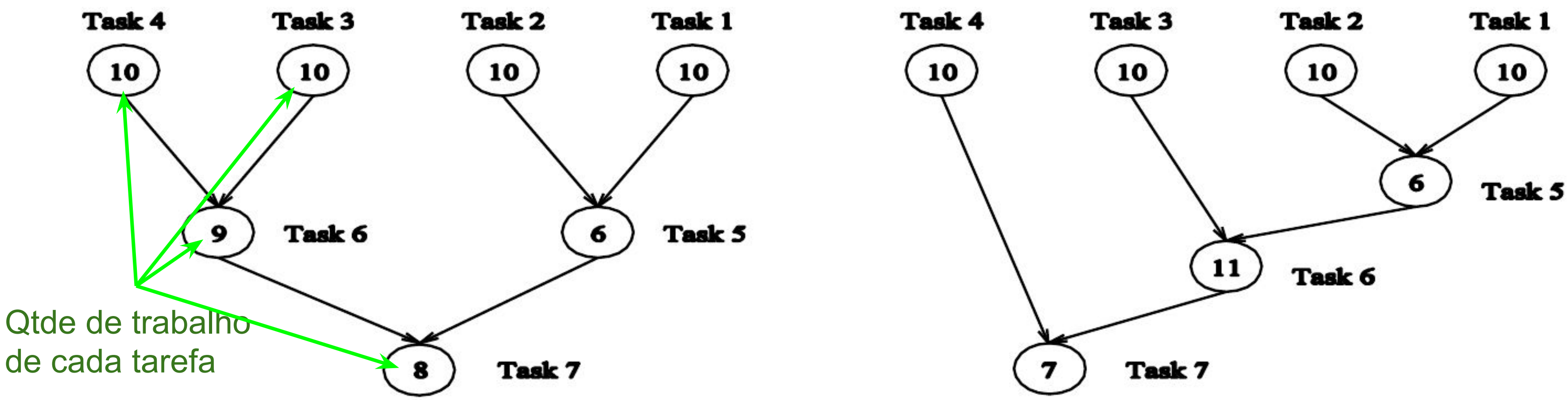
- Definição: número de tarefas que podem executar em paralelo
- Pode mudar durante a execução do programa
- Métricas
  - grau máximo de concorrência
    - número máximo de tarefas concorrentes em qualquer ponto da execução
  - grau médio de concorrência
    - número médio de tarefas que podem ser processadas em paralelo
- Grau de concorrência *vs* granularidade de tarefas
  - O grau de concorrência aumenta conforme a decomposição se torna mais fina em granularidade e vice versa.

Caminho Crítico

- Aresta no gráfico de dependência das tarefas representa a serialização das tarefas
- Caminho Crítico = caminho mais longo ponderado através do grafo
- Comprimento do Caminho Crítico = limite inferior para o tempo de execução paralelo.

Comprimento de Caminho Crítico

Exemplos: grafos de dependência para tarefas de consulta em um BD



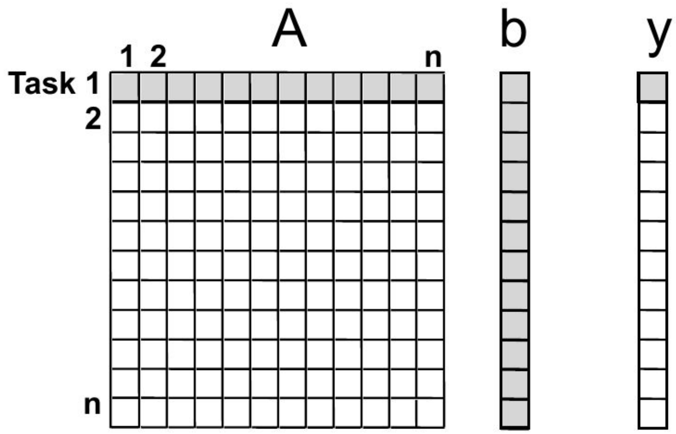
Questões:

- Quais são as tarefas no caminho crítico para cada grafo de dependência?
- Quais é o caminho mais curto de tempo de execução paralela para cada decomposição?
- Quantos processadores são necessários para se conseguir o tempo mínimo de execução?
- Qual é o grau máximo de concorrência?
- Qual é o paralelismo médio?

Comprimento de Caminho Crítico

- Nós sem nenhuma aresta chegando: nós de início
- Nós sem nenhuma aresta saindo: nós de final
- O comprimento mais longo direcionado entre quaisquer pares de nós de início e final é chamado de *caminho crítico*.
- A soma dos pesos dos nós (qtde de trabalho de cada tarefa) ao longo deste caminho é conhecido como **comprimento de caminho crítico**.
- A razão do total de trabalho para o **comprimento de caminho crítico** é o grau médio de concorrência.
- Portanto, um caminho crítico mais curto favorece um mais alto grau de concorrência.
- Para o gráfico da esquerda, o comprimento de caminho crítico é 27 e 34 para o da direita.
- O total de trabalho necessário para resolver o problema é de 63 para o gráfico da esquerda e 64 para o da direita.
- O grau médio de concorrência é então: 63/27 = 2.33 e 64/34 = 1.88, respect.

Comprimento de Caminho Crítico



Questões:

- Como seria um grafo de dependência de tarefas para o MMDV?
- Quais são as tarefas no caminho crítico para cada grafo de dependência?
- Quais é o caminho mais curto de tempo de execução paralela para cada decomposição?
- Quantos processadores são necessários para se conseguir o tempo mínimo de execução?
- Qual é o grau máximo de concorrência?
- Qual é o paralelismo médio?

Limites no Desempenho Paralelo

- O que limita o tempo de execução paralelo?
  - granularidade mínima de tarefas
    - p.explo, multiplicação de matriz densa por vetor $\leq n^2$ tarefas concorrentes
  - dependências entre tarefas
  - custos adicionais de paralelização
    - custo de comunicação entre tarefas
  - fração do trabalho da aplicação que não pode ser paralelizado
    - Lei de Amdahl, conforme já vimos
- Medidas de desempenho paralelo
  - Speedup (T_1/T_p)
  - Eficiência de paralelização = $T_1/(pT_p)$

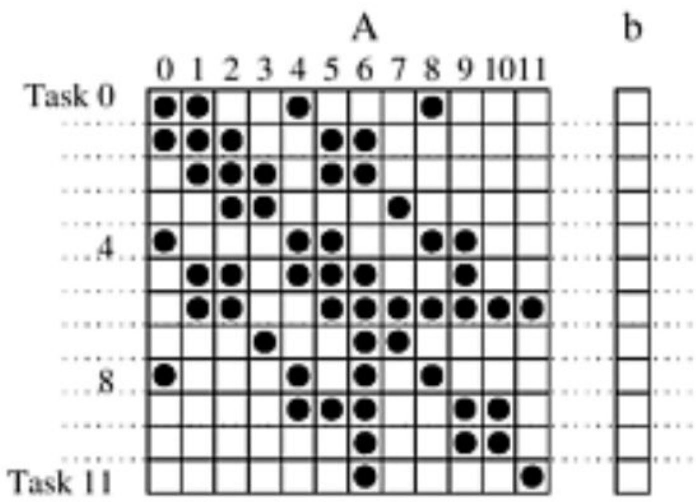
Grafos de Interação entre Tarefas

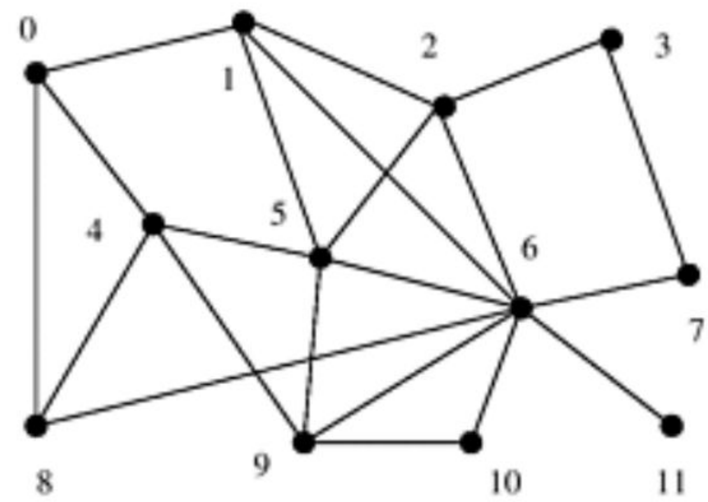
- Tarefas geralmente trocam dados umas com as outras
    - exemplo: multiplicação da matriz densa por vetor
        - se o vetor **b** não é replicado em todas as tarefas, elas terão que se comunicar para receberem os elementos de **b**.
- Grafo de interação entre as tarefas
    - nós: tarefa
    - aresta: interação ou troca de dados
- Grafo de interação de tarefas *vs* grafo de dependência de tarefas
    - grafo de interação de tarefas representam **dependência de dados**
    - grafo de dependência de tarefas representam **dependências de controle**

Exemplo de Grafo de Interação entre Tarefas

Multiplicação de Matriz esparsa por vetor

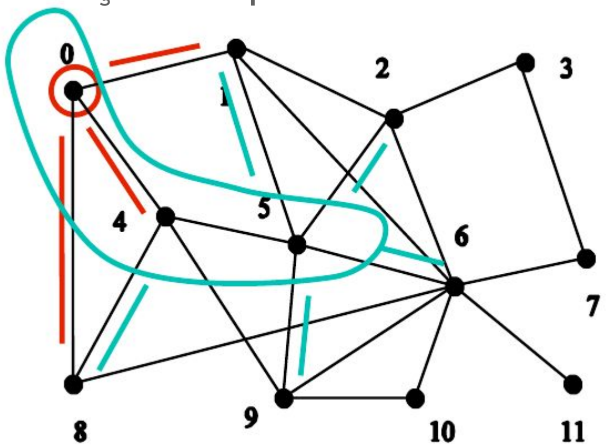
- Computação de cada elemento do resultado: tarefa independente
- Somente elementos não-zero da matriz esparsa A participam.
- O vetor **b** deve ser particionado entre as tarefas ....
  - estrutura do grafo de interação entre tarefas = grafo da matriz A (i.e., o grafo para o qual A representa a estrutura de adjacência)





Grafos de Interação, Granularidade e Comunicação

- Granularidade mais fina de tarefas aumenta o custo de comunicação
- Exemplo: grafo de interação do produto da matriz esparsa por vetor



- Hipóteses:
  - cada nó leva uma unidade de tempo para processar
  - cada interação (aresta) causa um custo adicional de uma unidade de tempo.
- Se o nó 0 é uma tarefa: comunicação = 3; computação = 4
- Se os nós 0, 4 e 5 compõem uma tarefa: comunicação = 5; computação = 15
  - decomposição de granularidade mais grosseira (*coarser*) -> menor razão comunicação/computação

Processos e Mapeamento

- Geralmente
  - Núm de tarefas > núm de elementos de processamento disponíveis
  - Algoritmos paralelos devem mapear tarefas a processos
- Por que processos ao invés de processadores?
  - agregar tarefas em processos
    - processo = agente de processamento ou computação que realiza o trabalho
    - atribui coleções de tarefas e dados associados a um processo
  - sistema operacional mapeia processos a processadores físicos
    - alguns sistemas operacionais permitem que se especifique uma ligação entre processos e processadores.

Processos e Mapeamentos

- Mapear tarefas a processos é crítico para o desempenho paralelo.
- Em que bases se deve escolher o mapeamento?
  - utilizando grafos de dependência de tarefas
    - escalone tarefas independentes a processos separados
      - ociosidade mínima
      - balanceamento de carga ótimo
  - utilizando grafos de interações de tarefas
    - queremos que os processos tenham uma interação mínima um com o outro
      - comunicação mínima

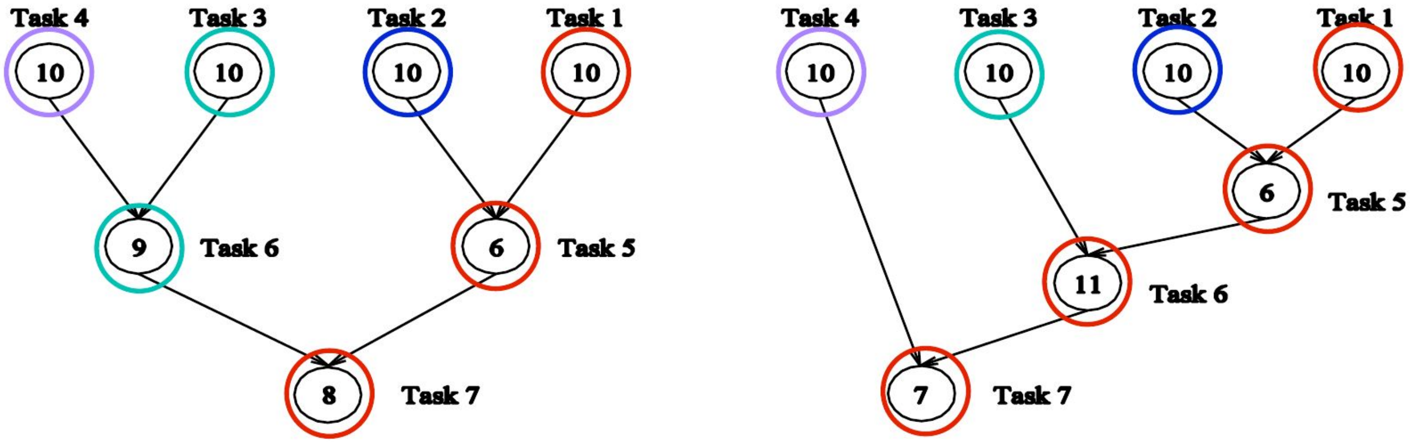
Processos e Mapeamento

Um bom mapeamento deve minimizar o tempo de execução paralelo através de:

- Mapeamento de tarefas independentes a diferentes processos
- Atribuindo tarefas no caminho crítico a processos *ASAP*
- Minimização da interação entre processos
  - mapeia tarefas com interações densas ao mesmo processo.

- Dificuldade: critérios frequentemente conflitam uns com os outros.
  - p.explo, nenhuma decomposição minimiza interações mas sem *speedup*!

Exemplo: Processos e Mapeamento



Exemplo: mapeando consultas ao banco de dados a processos.

- Considere os grafos de dependência em níveis
  - nenhum nó em um nível depende de outro
  - computar os níveis utilizando sort topológico
- Atribui todas as tarefas dentro de um nível a diferentes processos

Técnicas de Decomposição

Técnicas de Decomposição

Como devemos decompor uma tarefa em várias subtarefas?

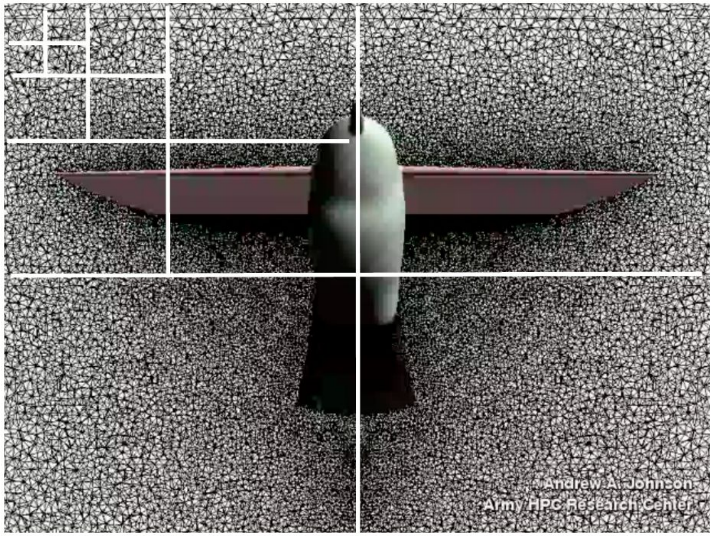
- Não existe uma receita universal
- Na prática, uma variedade de técnicas são utilizadas, incluindo:
  - decomposição recursiva
  - decomposição de dados
  - decomposição exploratória
  - decomposição especulativa

Decomposição Recursiva

Adequada para problemas solucionáveis utilizando a abordagem dividir-e-conquistar

Passos

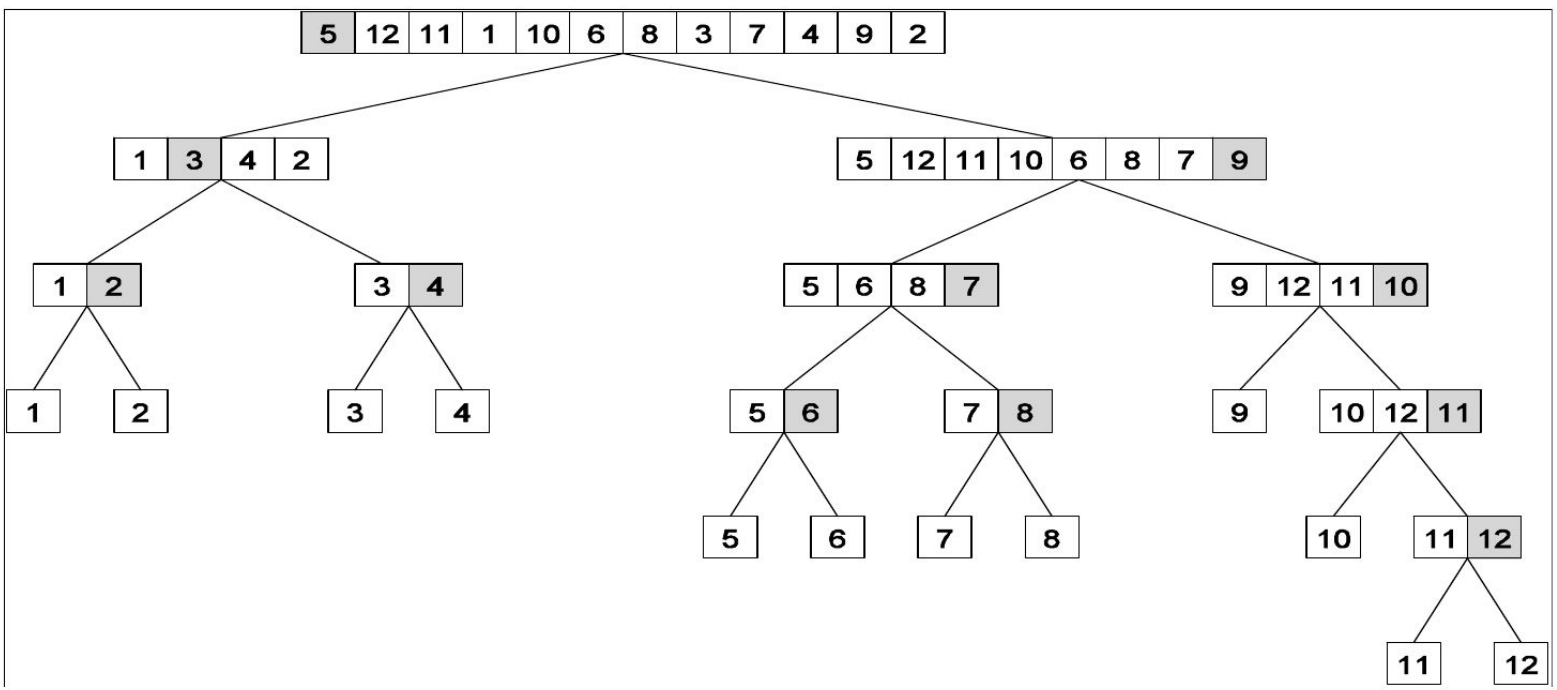
1. decomponha um problema em um conjunto de sub-problemas
2. recursivamente decomponha cada sub-problema
3. pare a decomposição quando a granularidade mínima desejada é alcançada



Decomposição Recursiva para o Quicksort

Em cada nível:

1. Selecione um pivô
2. Particione o conjunto em torno do pivô
3. Recursivamente ordene cada subvetor

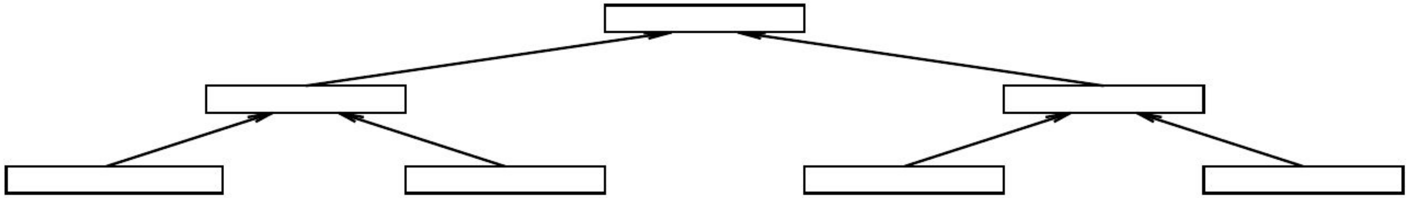


Decomposição Recursiva para se achar o Mínimo

Encontrando o mínimo em um vetor utilizando-se a técnica divida-e-conquista

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

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



Aplicável a outras operações associativas, e.g., soma, E, ...

Decomposição de Dados

- Passos
  1. Identifique os dados nos quais as computações são realizadas
  2. Particione os dados entre as várias tarefas
     a. o particionamento induz a decomposição do problema
- Dados podem ser particionados de várias maneiras
  - particionamento apropriado é crítico para o desempenho paralelo
- Decomposição baseada em
  - dados de entrada
  - dados de saída
  - dados de entrada + saída
  - dados intermediários

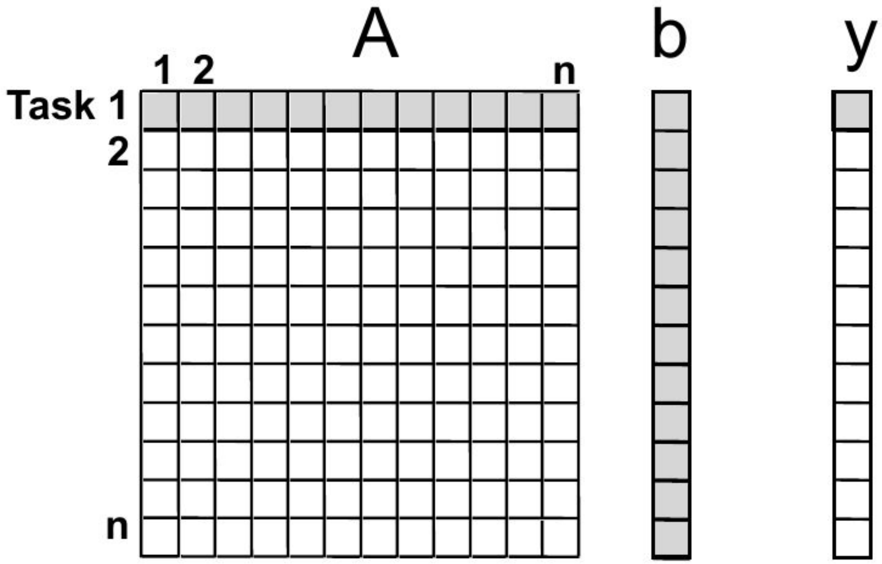
Decomposição Baseada em Dados de Entrada

- Aplicável se cada saída é computada como uma função da entrada
- Pode ser a única decomposição natural se a saída não é conhecida
    - exemplos
        - encontrar o mínimo em um conjunto ou outras reduções
        - ordenar um vetor
- Associar uma tarefa com cada partição dos dados de entrada
    - tarefa realiza computação na sua parte dos dados
    - processamento subsequente combina os resultados parciais das tarefas anteriores
- As soluções para tarefas induzidas por particionamento dos dados de entrada podem não resolver diretamente o problema original.
    - pode ser necessária uma etapa subsequente para combinar os resultados.

Decomposição baseada em Dados de Saída

- Se cada elemento da saída pode ser calculado independentemente
- Particionar os dados de saída entre as tarefas
- Fazer cada tarefa realizar a computação para suas saídas

Exemplo:
multiplicação de
matriz densa
por vetor



Exemplo: Decomposição de dados de saída

- Multiplicação de Matriz: C = A x B
- Computação de C pode ser particionada em quatro tarefas

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

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

Outras decomposições de tarefas são possíveis

Particionamento de Dados Intermediários

- Se a computação é uma sequência de transformações
  - **(de dados de entrada em dados de saída)**
- Pode-se decompor baseado em dados necessários nas etapas intermediárias
  - o particionamento de dados intermediários pode levar, em alguns casos, a uma maior concorrência do que o particionamento dos dados de entrada ou de saída individualmente.
- Frequentemente, os dados intermediários não são gerados explicitamente no algoritmo serial que resolve o problema
  - é necessário reestruturar o algoritmo original para usar o particionamento de dados intermediários.

Exemplo: Particionamento de Dados Intermediários

Multiplicação de Matriz Densa: Decomposição de dados intermediários: resulta em 8 + 4 tarefas

Stage I

$$\begin{pmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{pmatrix} \cdot \begin{pmatrix} B_{1,1} & B_{1,2} \\ B_{2,1} & B_{2,2} \end{pmatrix} \rightarrow \left(\begin{array}{c} \begin{pmatrix} D_{1,1,1} & D_{1,1,2} \\ D_{1,2,1} & D_{1,2,2} \end{pmatrix} \\ \begin{pmatrix} D_{2,1,1} & D_{2,1,2} \\ D_{2,2,1} & D_{2,2,2} \end{pmatrix} \end{array} \right)$$

Stage II

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

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: $C_{2,,2} = D_{1,2,2} + D_{2,2,2}$

Exemplo: Particionamento de Dados Intermediários

Tarefas: decomposição de dados intermediários na multiplicação de matriz densa

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: $D_{2,2,2} = A_{2,2} B_{2,2}$

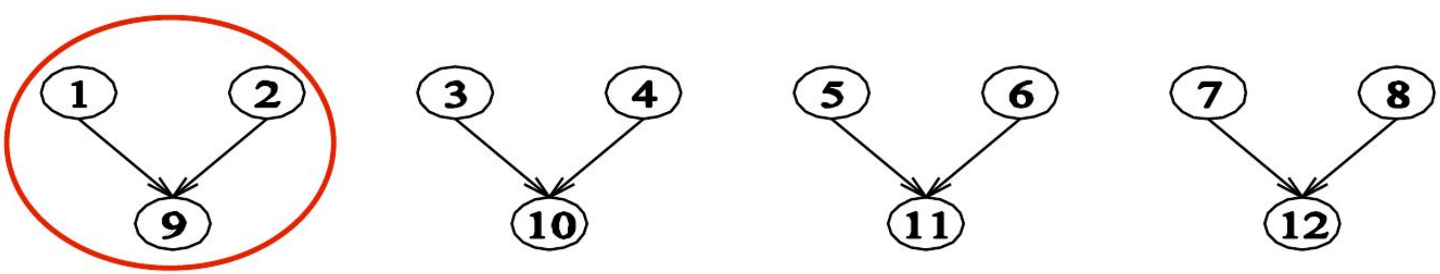
Task 09: $C_{1,1} = D_{1,1,1} + D_{2,1,1}$

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: $C_{2,,2} = D_{1,2,2} + D_{2,2,2}$

Grafo de Dependência de Tarefas

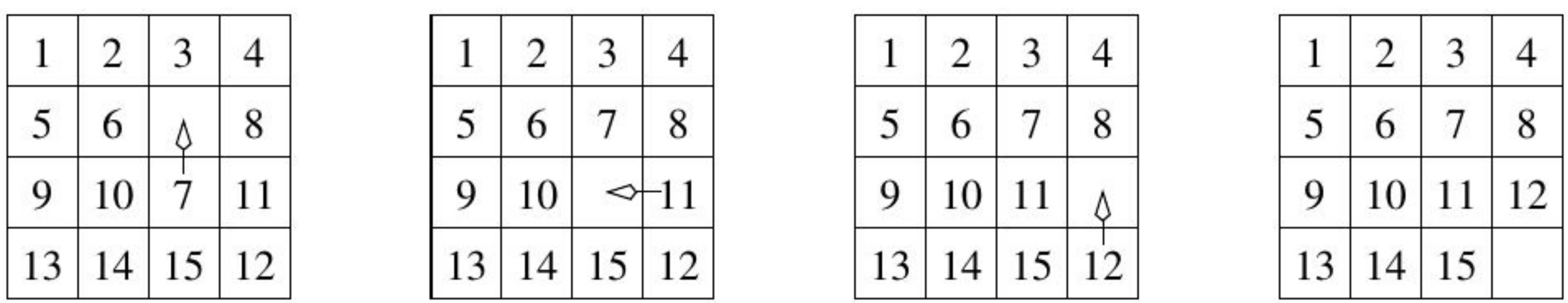


Decomposição Exploratória

- É utilizada para decompor problemas cuja computação subjacente corresponde a uma busca em um espaço de soluções.
- Na decomposição exploratória, particionamos o espaço de busca em partes menores e a busca é realizada em cada uma destas partes concorrentemente, até que a solução seja encontrada.

Decomposição Exploratória - exemplo

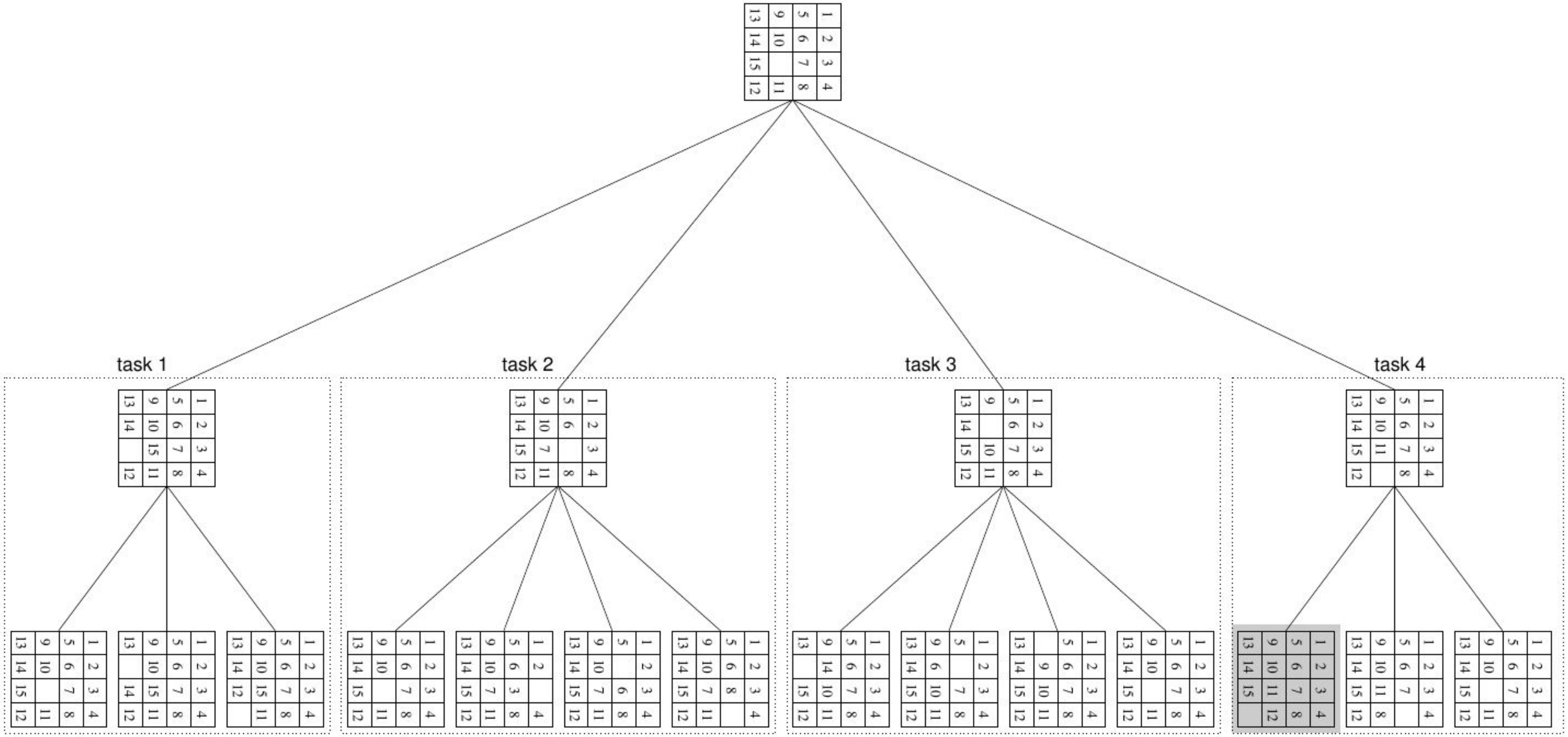
- Este jogo consiste de 15 peças numeradas de 1 a 15 e um espaço vazio, todos posicionados em uma grade 4 x 4.
- Uma peça pode ser movida para a posição em branco de uma posição adjacente a ela, assim criando um espaço vazio na sua posição original.
- Dependendo da configuração da grade, até 4 movimentos são possíveis: pra cima, pra baixo, pra esquerda, pra direita.
- As configurações inicial e final das peças são dadas (especificadas).
- O objetivo é determinar qualquer sequência ou uma menor sequência de movimentos que transforme a configuração inicial na configuração final.



(a) (b) (c) (d)

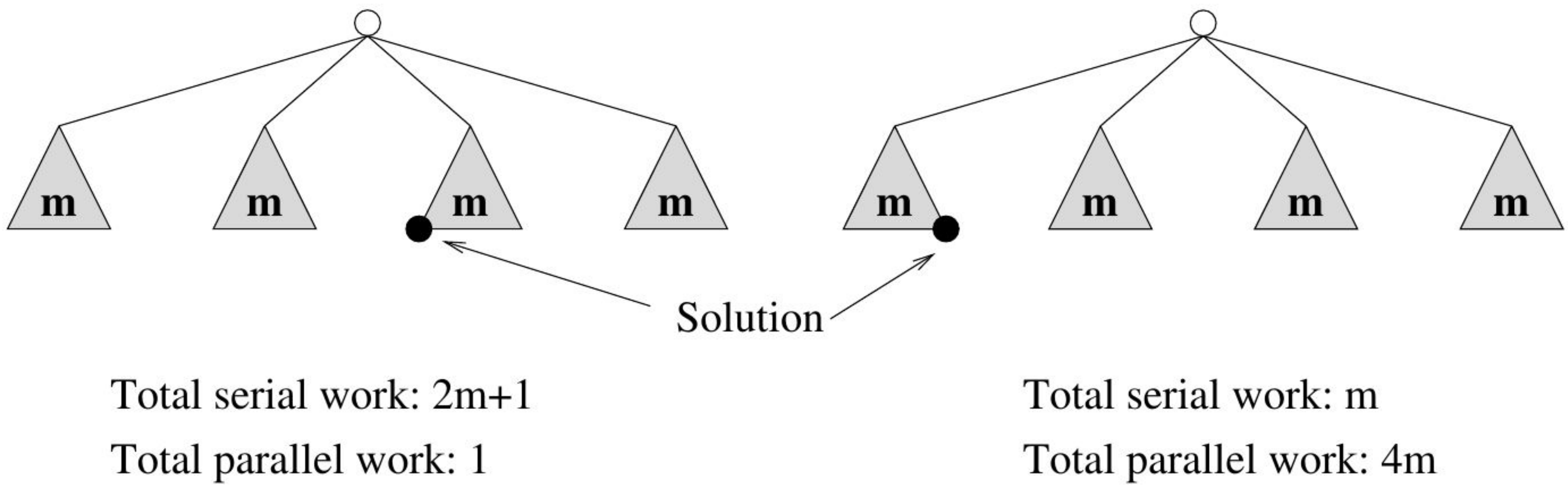
Decomposição Exploratória - exemplo

O espaço de estados pode ser explorado gerando-se vários estados sucessores do estado atual e enxergá-los como tarefas independentes.



Decomposição Exploratória - exemplo

- Em muitos casos da decomposição exploratória, a técnica de decomposição pode mudar a quantidade de trabalho realizada pela formulação paralela.
- Isto muda os resultados de speedups:
    - super linear
    - sub linear



Decomposição Especulativa

- Em algumas aplicações, as dependências entre as tarefas não são conhecidas *a priori*.
- Para este tipo de aplicação, é impossível identificar tarefas independentes.
- Existem geralmente duas abordagens para se lidar com tais aplicações:
  - Abordagens conservativas, que identificam tarefas independentes somente quando se tem garantia de que realmente não têm dependências, e
  - Abordagens otimistas, que agendam tarefas mesmo que elas possam estar, potencialmente, erradas.
- Abordagens conservativas podem produzir pouca concorrência e abordagens otimistas podem precisar de um mecanismo de *roll-back* no caso de erros.

Decomposição Especulativa - exemplo

Um exemplo clássico de decomposição especulativa é a simulação de eventos discretos.

- A estrutura de dados central em uma simulação de eventos discreta é uma lista de eventos ordenadas no tempo.
- Os eventos são extraídos exatamente na ordem temporal, processados e se necessário, eventos resultantes são inseridos de volta na lista de eventos.
- Considere, p.explo, o seu dia de hoje como um sistema de eventos discretos - vc acordou, se preparou, dirigiu até o trabalho, trabalhou, almoçou, trabalhou um pouco mais, voltou pra casa, jantou e foi dormir.
- Cada um destes eventos pode ser processados independentemente, contudo, na ida para o trabalho vc pode se deparar com um acidente infeliz e não chegar ao trabalho.
- Portanto, o agendamento otimista dos outros eventos deverá ser *rolled back*.