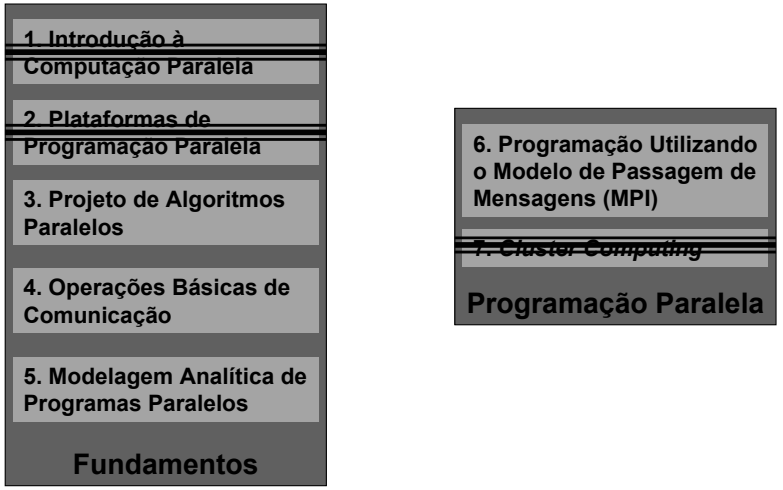
Projeto de Algoritmos Paralelos

Fundamentos de Projeto

Aula 10
Alessandro L. Koerich

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

Programa do PA



Introdução

- O desenvolvimento de algoritmos é um componente crítico na resolução de problemas usando computadores.
- **Algoritmo**: receita ou seqüência de passos básicos para resolver um dado problema

Introdução

- **Algoritmo Paralelo**: mais do que somente a especificação de passos.
- **Concorrência**: o projetista deve especificar conjuntos de passos que possam ser executados simultaneamente.
- Na prática temos...

Introdução

- A especificação deve incluir:
  - Identificação de porções do trabalho que possam ser realizadas concorrentemente;
  - Mapeamento dos "pedaços" concorrentes para trabalharem sobre múltiplos processos executados em paralelo;
  - Distribuição dos dados de entrada, saída e intermediários associados com o programa;

Introdução

- A especificação deve incluir (cont.):
  - Gerenciamento do acesso aos dados compartilhados por múltiplos processadores;
  - Sincronização dos processadores em vários estágios da execução do programa paralelo

Introdução

- **Objetivo**: Discutir o processo de projeto e implementação de algoritmos paralelos.
- **Note**: existem ferramentas e compiladores para a paralelização automática. Não serão abordadas.

Fundamentos

Conceito chave no projeto de algoritmos paralelos

Dividir a computação em computações menores e atribuí–las a diferentes processadores para a execução paralela.

Fundamentos

- Decomposição: dividir a computação em partes menores, onde todas ou algumas delas, possam ser executadas em paralelo.
- Tarefa (*tasks*): unidades de computação definidas pelo programador, onde a computação principal é subdividida por meio de decomposição. (ex. 3.1+fig. 3.1)
- Grafo de dependência de tarefas: nós representam as tarefas e as linhas direcionadas indicam a dependência entre eles. (ex. 3.2+fig. 3.2 3.3)

Fundamentos

Example 3.1 Dense matrix-vector multiplication

Consider the multiplication of a dense $n \times n$ matrix A with a vector b to yield another vector y. The ith element $y[i]$ of the product vector is the dot-product of the ith row of A with the input vector b; i.e., $y[i] = \sum_{j=1}^{n} A[i, j].b[j]$. As shown later in Figure 3.1, the computation of each $y[i]$ can be regarded as a task. Alternatively, as shown later in Figure 3.4, the computation could be decomposed into fewer, say four, tasks where each task computes roughly $n/4$ of the entries of the vector y. ■

Fundamentos

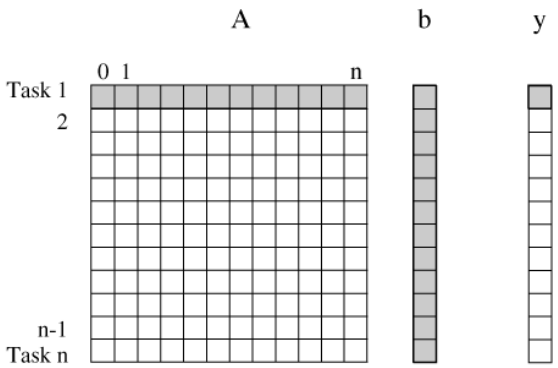


Figure 3.1 Decomposition of dense matrix-vector multiplication into n tasks, where n is the number of rows in the matrix. The portions of the matrix and the input and output vectors accessed by Task 1 are highlighted.

Fundamentos

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.

Example 3.2 Database query processing

Table 3.1 shows a relational database of vehicles. Each row of the table is a record that contains data corresponding to a particular vehicle, such as its ID, model, year, color, etc. in various fields. Consider the computations performed in processing the following query:

MODEL="Civic" AND YEAR="2001" AND (COLOR="Green" OR COLOR="White")

This query looks for all 2001 Civics whose color is either Green or White. On a relational database, this query is processed by creating a number of intermediate tables. One possible way is to first create the following four tables: a table containing all Civics, a table containing all 2001-model cars, a table containing all green-colored cars, and a table containing all white-colored cars. Next, the computation proceeds by combining these tables by computing their pairwise intersections or unions. In particular, it computes the intersection of the Civic-table with the 2001-model year table, to construct a table of all 2001-model Civics. Similarly, it computes the union of the green- and white-colored tables to compute a table storing all cars whose color is either green or white. Finally, it computes the intersection of the table containing all the 2001 Civics with the table containing all the green or white vehicles, and returns the desired list. ■

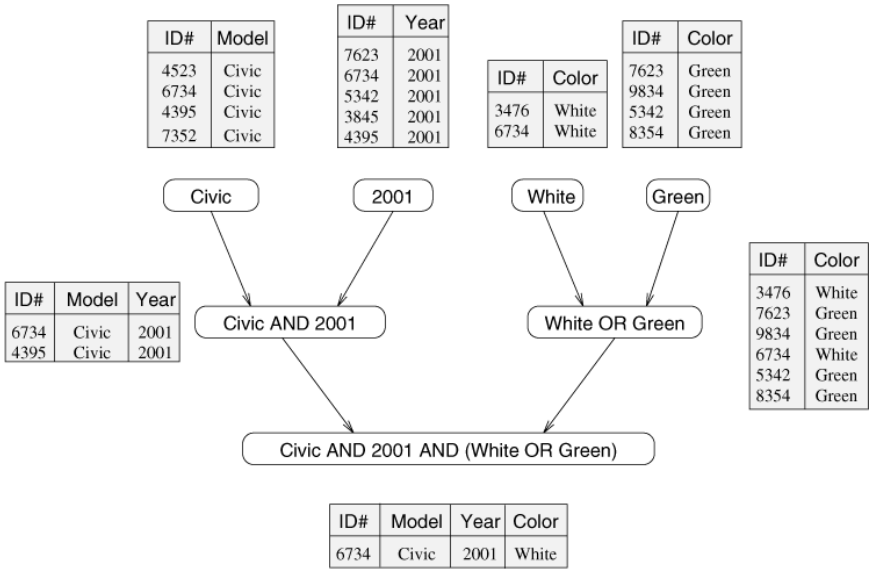


Figure 3.2 The different tables and their dependencies in a query processing operation.

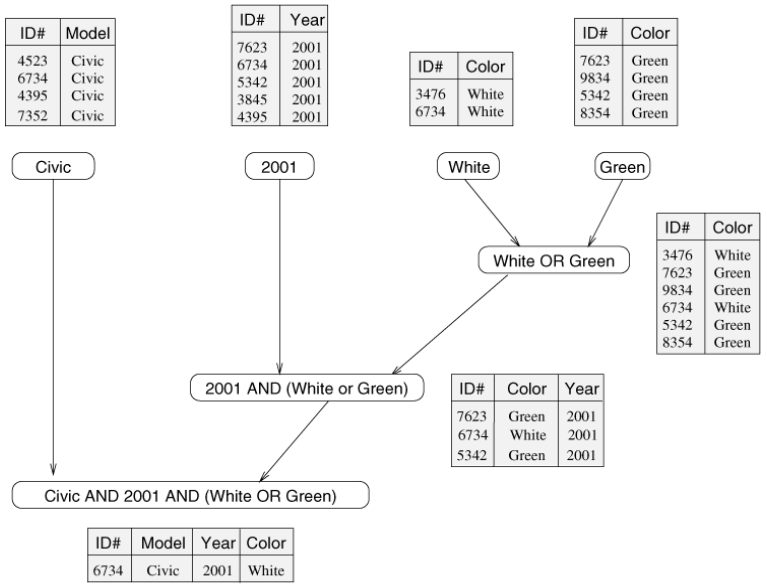


Figure 3.3 An alternate data-dependency graph for the query processing operation.

- Granularidade da decomposição: o número e o tamanho das tarefas nas quais um problema pode ser decomposto.
  - Decomposição em um **grande número** de **pequenas tarefas ➡ granularidade fina** (*fine–grained*).
  - Decomposição em um **pequeno número** de **grandes tarefas ➡ granularidade grossa** (*coarse–grained*)
  - (fig 3.1 3.4)

Fundamentos

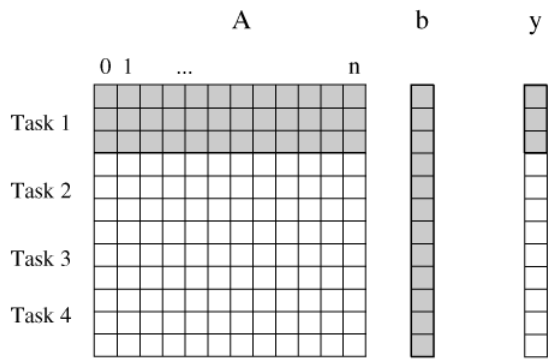


Figure 3.4 Decomposition of dense matrix-vector multiplication into four tasks. The portions of the matrix and the input and output vectors accessed by Task 1 are highlighted.

Fundamentos

- Grau máximo de concorrência: número máximo de tarefas que podem ser executadas simultaneamente em um programa paralelo a qualquer tempo.
- Geralmente é menor que o número total de tarefas devido a dependências.
- É quatro para os grafos das Fig 3.2 e 3.3

Fundamentos

- Grau médio de concorrência: o número médio de tarefas que podem ser executas concorrentemente durante a execução total do programa (fig. 3.5)
- Em geral grau de concorrência ↑ com a ↓ na granularidade das tarefas (fina)
  - Fig 3.1 = granularidade pequena e alto grau de concorrência
  - Fig 3.4 = granularidade mais alta e grau de concorrência menor

Fundamentos

- Porém o grau de concorrência depende também da forma do grafo de dependência de tarefas.
  - Fig 3.5 = grafos de tarefas de Fig 3.2 e Fig 3.3
  - Número em cada nó representa a quantidade de trabalho para completar a tarefa.
  - Concorrência média (a) = 2.33
  - Concorrência média (b) = 1.88

Porém, ambos tem a mesma decomposição !!!

Fundamentos

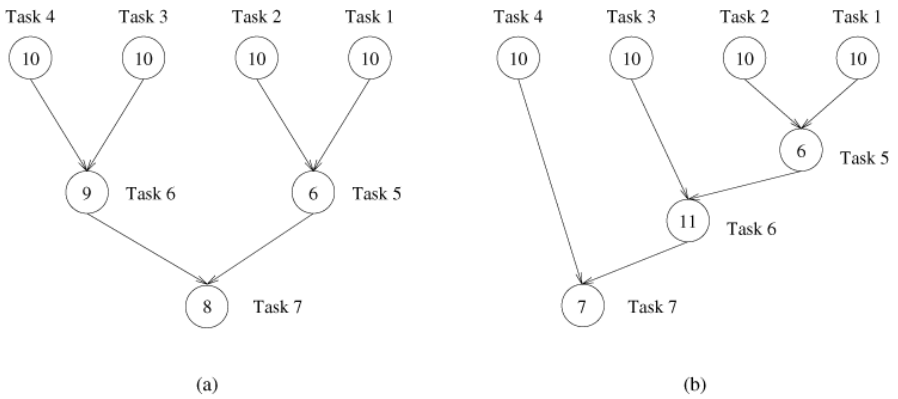


Figure 3.5 Abstractions of the task graphs of Figures 3.2 and 3.3, respectively.

Fundamentos

- Caminho crítico: o caminho direto mais longo entre qualquer par de nós iniciais e finais.
  - Determina o grau médio de concorrência !!!

$$\text{Concorrência Média} = \frac{\text{Quantidade Total de Trabalho}}{\text{Caminho Crítico}}$$

- Comprimento do caminho crítico: soma dos pesos dos nós ao longo do caminho crítico
  - Fig. 3.5a = 27
  - Fig. 3.5b = 34

Fundamentos

Para ↓ o tempo de resolução de um problema podemos

↑ a granularidade da decomposição ➡

aumentar a concorrência ➡

realizar mais e mais tarefas em paralelo.

Porém, a granularidade é limitada !!!!!

Fundamentos

- Interação: interação entre tarefas sendo executadas em diferentes processadores.
  - A tarefas geralmente compartilham dados de entrada, saída ou intermediários.
  - Tarefas podem parecer independentes no grafo de dependência, porém, necessitam acessar dados comuns (Ex. vetor b na multiplicação de matrizes)

Fundamentos

- Grafo de interação de tarefas: o padrão de interação entre tarefas.
  - Nós: representa as tarefas
  - Vértices: conectam tarefas que interagem umas com as outras.
- Ex. 3.3 + Fig. 3.6

Fundamentos

Example 3.3 Sparse matrix-vector multiplication

Consider the problem of computing the product $y = Ab$ of a sparse $n \times n$ matrix A with a dense $n \times 1$ vector b. A matrix is considered sparse when a significant number of entries in it are zero and the locations of the non-zero entries do not conform to a predefined structure or pattern. Arithmetic operations involving sparse matrices can often be optimized significantly by avoiding computations involving the zeros. For instance, while computing the ith entry $y[i] = \sum_{j=1}^{n}(A[i, j] \times b[j])$ of the product vector, we need to compute the products $A[i, j] \times b[j]$ for only those values of j for which $A[i, j] \neq 0$. For example, $y[0] = A[0, 0].b[0] + A[0, 1].b[1] + A[0, 4].b[4] + A[0, 8].b[8]$.

One possible way of decomposing this computation is to partition the output vector y and have each task compute an entry in it. Figure 3.6(a) illustrates this decomposition. In addition to assigning the computation of the element $y[i]$ of the output vector to Task i, we also make it the "owner" of row $A[i, *]$ of the matrix and the element $b[i]$ of the input vector. Note that the computation of $y[i]$ requires access to many elements of b that are owned by other tasks. So Task i must get these elements from the appropriate locations. In the message-passing paradigm, with the ownership of $b[i]$, Task i also inherits the responsibility of sending $b[i]$ to all the other tasks that need it for their computation. For example, Task 4 must send $b[4]$ to Tasks 0, 5, 8, and 9 and must get $b[0]$, $b[5]$, $b[8]$, and $b[9]$ to perform its own computation. The resulting task-interaction graph is shown in Figure 3.6(b). ■

Fundamentos

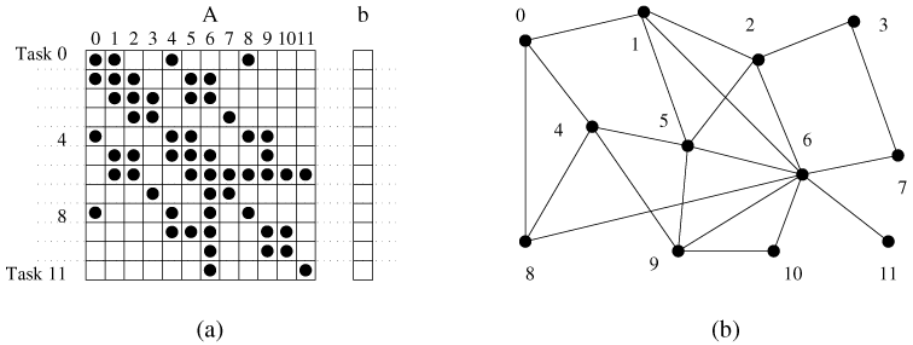


Figure 3.6 A decomposition for sparse matrix-vector multiplication and the corresponding task-interaction graph. In the decomposition Task i computes $\sum_{0 \leq j \leq 11, A[i,j] \neq 0} A[i, j].b[j]$.

Fundamentos

- As tarefas nas quais um problema é decomposto, são executadas em um processador físico.
- **Processo**: agente de processamento ou computação que realiza tarefas. Entidade abstrata que usa o código e os dados correspondentes a uma tarefa para produzir a saída para a tarefa em um intervalo limitado de tempo.
- Durante este tempo, além de realizar computações, o processo pode sincronizar ou se comunicar com outros processos.

Fundamentos

- **Mapeamento**: o mecanismo pelo qual as tarefas são atribuídas aos processos para serem executadas
- Um bom mapeamento depende dos grafos de interação e dependência de tarefas.
- Um bom mapeamento deve:
  - Maximizar o uso da concorrência, mapeando tarefas independentes sobre diferentes processos;
  - Minimizar o tempo de término, disponibilizando processos para executar tarefas no caminho crítico;
  - Minimizar interação entre processos mapeando tarefas com alto grau de interação mútua sobre os mesmos processos.

Fundamentos

Mapeamento

- Porém, na maioria dos algoritmos paralelos, estas metas são conflitantes.
- Encontrar um balanço que otimize a performance paralela global é um ponto chave para um algoritmo paralelo bem sucedido
- Portanto, o mapeamento de tarefa em processos é fundamental na determinação da eficiência de um algoritmo paralelo.
- Fig 3.7

Fundamentos

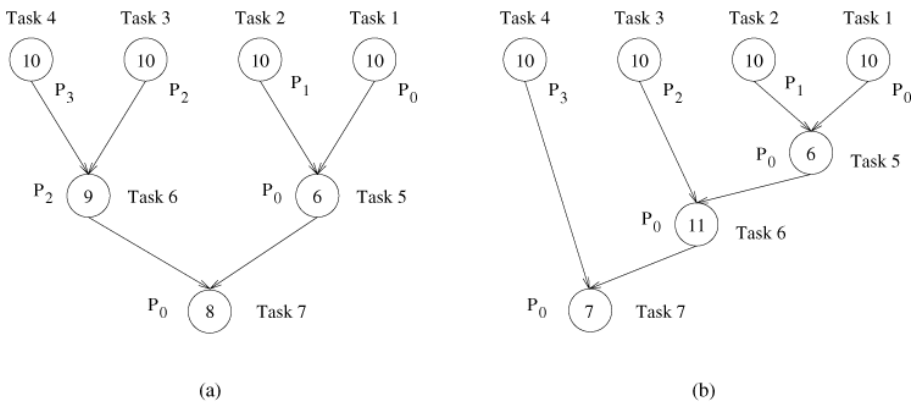


Figure 3.7 Mappings of the task graphs of Figure 3.5 onto four processes.

Fundamentos

Processos x Processadores

- Processos: agentes lógicos de computação que realizam tarefas;
- Processadores: unidades de hardware que realizam fisicamente a computação.
- Geralmente existe uma correspondência um para um