CSE3009 - Parallel and Distributed Computing

Course Type: LTP Credits: 4

Prepared by

Dr Komarasamy G

Associate Professor (Grade-2)

School of Computing Science and Engineering

VIT Bhopal University

Unit-2

Parallel Algorithm and Design - Preliminaries – Decomposition Techniques – Mapping Techniques for Load balancing.

Synchronous Parallel Processing – Introduction, Example-SIMD Architecture and Programming Principles.

Parallel Algorithm and Design

- Algorithm development is a critical component of **problem solving using computers.**
- A sequential algorithm is a **sequence of basic steps** for solving a given problem using a serial computer.
- A parallel algorithm has the added dimension of concurrency and the algorithm designer must specify sets of steps that can be executed simultaneously.
- This is essential for obtaining any performance benefit from the use of a parallel computer.

Parallel Algorithm and Design

- **Nontrivial parallel algorithm may include some of the following:**
  - ✔ **Identifying portions** of the work that can be performed concurrently.
  - ✔ Mapping the concurrent pieces of work onto multiple **processes running in parallel.**
  - ✔ **Distributing the input, output**, and intermediate data associated with the program.
  - ✔ Managing accesses to data **shared by multiple processors.**
  - ✔ **Synchronizing the processors** at various stages of the parallel program execution.

Parallel Algorithm and Design

- Parallel algorithms and their design are essential in **high-performance computing (HPC) and parallel computing.**
- They are used to efficiently solve problems by breaking them down into smaller tasks that can be executed simultaneously on multiple processing units.

Overview of parallel algorithms and their design considerations:

- **Parallelism Models**: There are various models for parallelism, including **task parallelism, data parallelism, pipeline parallelism**, and etc.
- **Granularity**: Granularity refers to the **size of the tasks being parallelized.** Fine-grained parallelism involves breaking tasks into small components, while coarse-grained parallelism involves larger tasks. Choosing the appropriate granularity is crucial for optimizing performance.

Parallel Algorithm and Design

- **Communication and Synchronization:** In parallel computing, communication and synchronization overhead can significantly impact performance. **Efficient algorithms minimize communication and synchronization**, often through techniques like message passing and shared-memory paradigms.
- **Load Balancing:** Ensuring that work is **evenly distributed across processing units** is essential for efficient parallel algorithms. Load balancing techniques dynamically distribute work to minimize idle time and maximize resource utilization.
- **Scalability:** Scalability refers to the ability of an algorithm to maintain or **improve performance as the problem size** or the number of processing units increases.
- Scalable algorithms avoid bottlenecks and contention points that hinder performance with larger problem sizes or more processors.

Parallel Algorithm and Design

- **Algorithm Design Patterns:** Various design patterns exist for parallel algorithms, such as **divide and conquer, parallel loops, map-reduce, and pattern computations**.
- Understanding these patterns helps in designing efficient parallel algorithms for different problem domains.
- **Parallel Data Structures:** Efficient data structures are crucial for parallel algorithms. Data structures should support concurrent **access and updates while minimizing contention and synchronization overhead**.
- **Parallelization Strategies:** Depending on the problem and the hardware architecture, different parallelization strategies may be appropriate, such as **task parallelism, data parallelism, or a combination of both**.

Parallel Algorithm and Design

- **Scalable Algorithms for Specific Problems:** Some problems have well-known parallel algorithms tailored to their characteristics. Examples include **parallel sorting algorithms, parallel graph algorithms, parallel linear algebra algorithms**, etc.
- **Performance Analysis and Optimization:** Profiling and analyzing the performance of parallel algorithms are essential for identifying bottlenecks and optimizing resource utilization.
- Techniques such as **parallel profiling and performance counters** help in understanding the behavior of parallel algorithms.
- In summary, designing efficient parallel algorithms requires a deep understanding of parallel computing principles, algorithmic techniques, communication and synchronization mechanisms, and performance optimization strategies.
- It's a complex but rewarding field that enables the efficient utilization of modern parallel computing architectures.

Preliminaries

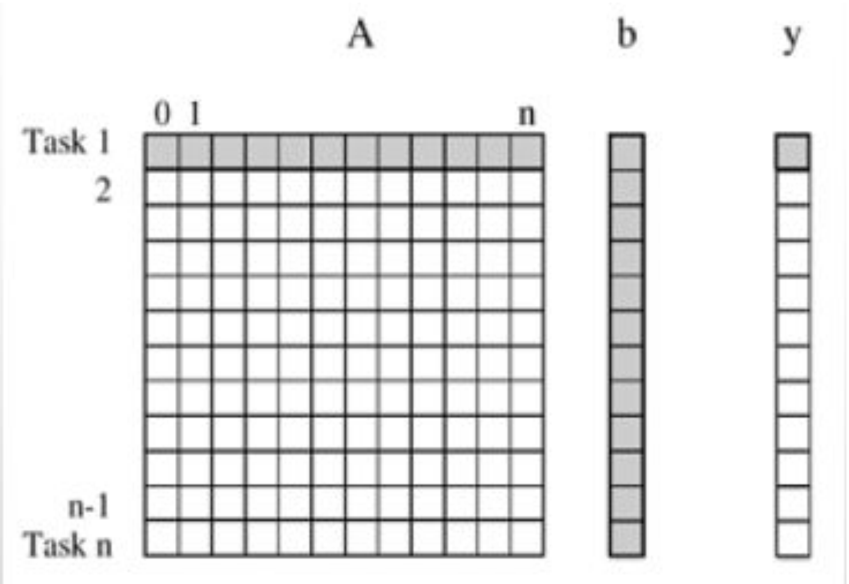
- Dividing a computation into **smaller computations and assigning them to different processors for parallel execution** are the two key steps in the design of parallel algorithms.
- Basic terminology and introducing these **two key steps** in parallel algorithm design using
  1. **Matrix-vector multiplication**
  2. **Database query processing**

Decomposition, Tasks, and Dependency Graphs

- The process of dividing a computation into smaller parts, some or all of which may potentially be executed in parallel, is called **decomposition.**
- Programmer-defined units of computation into which the main computation is **subdivided by means of decomposition**, is called **Task.**
- **Simultaneous execution of multiple tasks** is the key to reducing the time required to solve the entire problem.
- Some tasks may use data produced by other tasks and thus may need to wait for these tasks to finish execution.

- **Decomposition, Tasks, and Dependency Graphs**

Figure, 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.

An abstraction used to **express dependencies among tasks and their relative order** of execution is known as a **task dependency graph.**

Preliminaries

- **Granularity, Concurrency, and Task-Interaction**
- The number and size of tasks into which a problem is decomposed determines the **granularity of the decomposition.**
- Decomposition into a large number of small tasks is called **fine-grained.**
- Decomposition into a small number of large tasks is called **coarse-grained.**
- A concept related to granularity is that of **degree of concurrency.**
- The maximum number of tasks that can be executed simultaneously in a parallel program at any given time is known as its **maximum degree of concurrency.**
- A feature of a task-dependency graph that determines the average degree of concurrency for a given granularity is its **critical path.**

Preliminaries

- **The longest directed path between any** pair of start and finish nodes is known as the **critical path.**
- The **sum of the weights of nodes along this path** is known as the **critical path length,** where the weight of a node is the size or amount of work associated with corresponding task.
- The ratio of the total amount of work to the critical path length is the average degree of concurrency.

Preliminaries

- Figure 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.

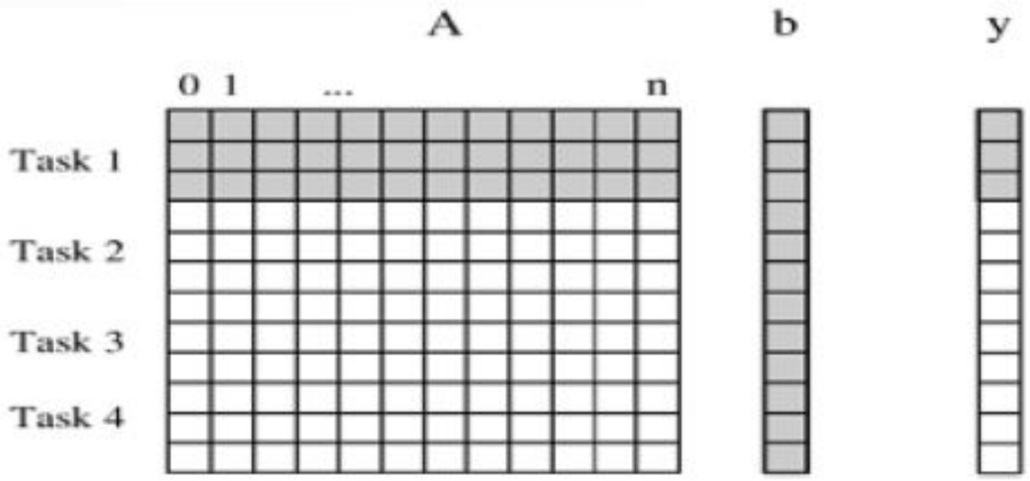


Figure , Abstractions of the task graphs