

Example 1: Video Streaming Platform (like YouTube)

One server handles user authentication (login/signup).

Another handles video storage and retrieval.

Another one analyzes user activity for recommendations.

Result: Different specialized tasks run in parallel → smooth performance.

Here's a detailed explanation of **Decomposition Techniques** in the context of **Parallel and Distributed Computing**, with examples:

Decomposition Techniques

Decomposition is the process of **breaking a large computational problem into smaller tasks** that can be executed **concurrently** on multiple processors or nodes. The goal is to **maximize parallelism and reduce execution time**.

Example 2: Google Search Indexing

There are **three main types of decomposition**:

Google divides billions of web pages among thousands of servers.

1. Data Decomposition

Each server analyzes and indexes its own set of pages.
 Result: Massive data processed quickly (same task on different data).

- **Definition:** The dataset is **divided into smaller parts**, and each processor or node works on a **different subset of the data**.
- **Key Idea:** Each processor performs the same operation on **different pieces of data**.
- **Parallel Computing Example:**
 - A matrix multiplication where a large matrix is split into blocks, and each core multiplies its assigned block.
- **Distributed Computing Example:**
 - In Hadoop, a 1 TB dataset is split into **chunks across nodes**, and each node processes its chunk independently (map tasks).

2. Task Decomposition (Functional Decomposition)

- **Definition:** The **problem is divided into different tasks or functions**, and each processor performs a **different task**.
- **Key Idea:** Processors work on **different computations** rather than the same operation on different data.
- **Parallel Computing Example:**
 - A video encoding pipeline where one core decodes frames, another applies filters, and a third compresses them.
- **Distributed Computing Example:**
 - In a web service, one server handles authentication, another handles database queries, and another handles analytics.

Example 2: Weather Simulation System

World is divided into regions (data decomposition).

Each region's processor calculates temperature, wind speed, and humidity (task decomposition)

Result: Global forecast generated faster and more accurately.

3. Hybrid Decomposition

- **Definition:** Combines **data and task decomposition** to exploit **both types of parallelism**.
 - **Parallel Computing Example:**
 - In a simulation, each thread processes a subset of particles (data decomposition) and applies different computations like force calculation and position update (task decomposition).
 - **Distributed Computing Example:**
 - In a distributed machine learning training system, the dataset is split across nodes (data decomposition), and each node runs **different phases of gradient computation** (task decomposition).
-

Summary Table

Decomposition Type	How it Works	Parallel Computing Example	Distributed Computing Example
Data	Split data among processors	Matrix multiplication, array processing	MapReduce: nodes process chunks of data
Task / Functional	Split tasks/functions among processors	Video processing pipeline: decode → filter → compress	Web service: auth server, DB server, analytics server
Hybrid	Combine data and task decomposition	Particle simulation: split particles & split computations	Distributed ML: split dataset + split gradient computations

Key Takeaways

- Decomposition is essential for **parallelism**.
 - **Data decomposition** works well when **same computation is applied to large data**.
 - **Task decomposition** works when **different computations are needed**.
 - **Hybrid decomposition** maximizes utilization of cores or nodes.
 - In **distributed systems**, decomposition also **minimizes communication** by keeping related data local.
-

Principle for Parallel Algorithm

- So how does one decompose a task into various subtasks?
- While there is no single recipe that works for all problems, we present a set of commonly used techniques that apply to broad classes of problems.
- These include:
 - Recursive decomposition
 - Data decomposition
 - Exploratory decomposition
 - Speculative decomposition

Here's a detailed explanation of **Principles for Parallel Algorithm Decomposition in Parallel and Distributed Computing**, along with examples:

Principle for Parallel Algorithm

Decomposing a task into subtasks is essential to **exploit parallelism**. There isn't a single method for all problems, but some commonly used **decomposition techniques** help design parallel algorithms efficiently.

The main techniques are: **Recursive decomposition, Data decomposition, Exploratory decomposition, Speculative decomposition**.

1. Recursive Decomposition

- **Definition:** Breaks a problem into **smaller subproblems of the same type**, solves them recursively, and combines the results.
- **Parallel Computing Role:** Each subproblem can be solved **concurrently** on different cores or threads.
- **Example (Parallel):**
 - **Merge Sort:** An array is divided into two halves, each half is sorted in parallel, and then merged.

- **Example (Distributed):**
 - Distributed recursive computation, like **parallel quicksort**, where each node sorts its assigned segment recursively.
-

2. Data Decomposition

- **Definition:** Divides **input data** among multiple processors, each performing the **same operation** on its portion.
 - **Parallel Computing Role:** Cores operate **independently on different data blocks**.
 - **Example (Parallel):**
 - Matrix multiplication: a large matrix is divided into blocks, and each thread multiplies its block.
 - **Example (Distributed):**
 - **MapReduce:** A dataset is split across nodes, and each node processes its chunk in parallel.
-

3. Exploratory Decomposition

- **Definition:** Tasks are divided based on **exploring multiple possibilities or search paths simultaneously**.
 - **Parallel Computing Role:** Different processors explore **different branches** of a search space or perform different experiments concurrently.
 - **Example (Parallel):**
 - Parallel **branch-and-bound** algorithm in optimization problems.
 - **Example (Distributed):**
 - Distributed **genetic algorithms**, where each node explores different populations or solutions.
-

4. Speculative Decomposition

- **Definition:** Tasks are executed **before it is certain they are needed**, hoping that they will be useful later.
- **Parallel Computing Role:** Exploits idle cores to **speculatively compute possible future tasks**.
- **Example (Parallel):**
 - **CPU speculative execution:** predicting which branch of a conditional will be executed and computing it in advance.
- **Example (Distributed):**
 - Distributed database systems may **precompute query results** or cache potential data blocks before actual requests arrive.

Summary Table

Decomposition Type	How it Works	Parallel Computing Example	Distributed Computing Example
Recursive	Break problem into smaller subproblems	Parallel Merge Sort	Parallel Quicksort across nodes
Data	Split input data among processors	Block matrix multiplication	MapReduce: split dataset among nodes
Exploratory	Explore multiple paths or possibilities	Branch-and-bound optimization	Distributed genetic algorithm
Speculative	Compute tasks before needed	CPU branch prediction	Precompute query results in distributed DB

Key Takeaways

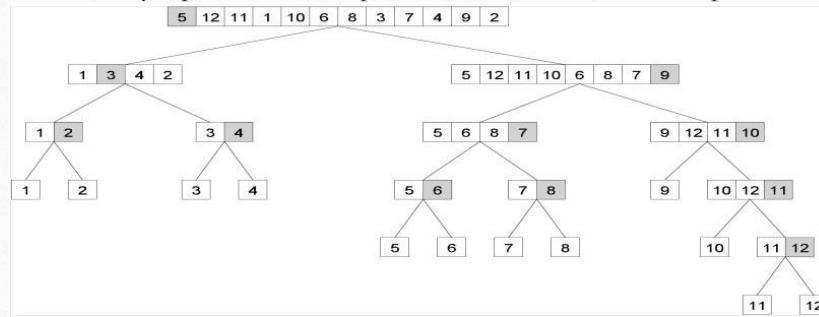
- Parallel algorithms rely on **decomposition to exploit concurrency**.
 - **Recursive and data decomposition** are most common for numerical problems.
 - **Exploratory and speculative decomposition** are useful for **search, prediction, and optimization** problems.
 - In **distributed computing**, decomposition also helps **reduce communication overhead** by localizing computations.
-

Recursive Decomposition

- Generally suited to problems that are solved using the divide and conquer strategy.
- A given problem is first decomposed into a set of subproblems.
- These sub-problems are recursively decomposed further until a desired granularity is reached.

Cont...

- A classic example of a divide-and-conquer algorithm on which we can apply recursive decomposition is Quicksort.
- In this example, a task represents the work of partitioning a (sub)array. Note that each subarray represents an independent subtask. This can be repeated recursively



Here's a clear explanation of **Recursive Decomposition** in the context of **Parallel and Distributed Computing**, with examples:

Recursive Decomposition

- **Definition:**
Recursive decomposition is a technique where a problem is **repeatedly broken down into smaller subproblems** until the subproblems are simple enough to solve directly. It is closely associated with **divide-and-conquer strategies**.
 - **How it works:**
 1. Take the original problem.
 2. Split it into **smaller subproblems**.
 3. Solve each subproblem **recursively**, either in parallel or sequentially.
 4. Combine the results to get the final solution.
-

Parallel Computing Perspective

- Each subproblem can be assigned to a **different core or thread**, allowing **concurrent execution**.
- Recursive decomposition is effective when subproblems are **independent**, minimizing communication between cores.

Example (Parallel Computing):

- **Quicksort on multi-core CPU:**
 1. Partition the array into two subarrays around a pivot.
 2. Assign each subarray to a separate core to sort concurrently.
 3. Each subarray may be further partitioned recursively, with tasks executed in parallel until small arrays are reached.
 - Benefits:
 - Reduces execution time significantly for large datasets.
 - Exploits **both coarse-grained and fine-grained parallelism**.
-

Distributed Computing Perspective

- Recursive decomposition can be extended across **nodes in a distributed system**.
- Each node processes a **subproblem independently**, and results are combined at higher levels.

Example (Distributed Computing):

- **Parallel Quicksort across cluster nodes:**
 1. Initial array is split into subarrays.
 2. Each subarray is sent to a different node to sort.
 3. Nodes may further split their subarrays locally and sort recursively.
 4. Sorted subarrays are merged to produce the final sorted array.
- Benefits:

- Allows handling of **very large datasets** that don't fit into a single machine's memory.
 - Reduces network bottlenecks by **processing subarrays locally**.
-

Key Points

- Recursive decomposition is best for **divide-and-conquer problems**.
 - Subproblems must be **independent** to exploit parallelism effectively.
 - In **parallel computing**, it reduces execution time by using multiple cores.
 - In **distributed computing**, it enables scaling across multiple nodes for large datasets.
-

◆ Video Streaming Platforms

Video files are split into segments and processed (encoded/compressed) in parallel by different

Exploratory Decomposition

- In many cases, the decomposition of the problem goes hand in-hand with its execution.
- These problems typically involve the exploration (search) of a state space of solutions.
- Problems in this class include a variety of discrete optimization problems (0/1 integer programming, QAP, etc.), theorem proving, game playing, etc.

Cont...

- A simple application of exploratory decomposition is in the solution to a 15 puzzle (a tile puzzle). We show a sequence of three moves that transform a given initial state (a) to desired final state (d).

1	2	3	4
5	6	8	
9	10	7	11
13	14	15	12

(a)

1	2	3	4
5	6	7	8
9	10	11	→
13	14	15	12

(b)

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

(c)

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

(d)

Cont...

- The state space can be explored by generating various successor states of the current state and viewing them as independent tasks.
- In many instances of parallel exploratory decomposition, unfinished tasks can be terminated when the first solution is found

Here's a clear explanation of **Exploratory Decomposition** in the context of **Parallel and Distributed Computing**, with examples:

Exploratory Decomposition

- **Definition:**
Exploratory decomposition is a technique where a problem is decomposed **dynamically during execution by exploring multiple possible solutions or paths simultaneously**.
- **Key Idea:**
 - Each subtask represents **a possible path, state, or solution**.
 - Tasks are **independent**, so they can be explored concurrently.
 - Often used in **search, optimization, and decision-making problems**.

Parallel Computing Perspective

- Multiple threads explore **different parts of the state space simultaneously**.
- Useful when the **first solution found can terminate other tasks** (early stopping).
- **Example (Parallel Computing): 15-Puzzle**
 - Problem: Move tiles from an initial configuration to a goal configuration.
 - Each thread explores a different **sequence of moves** (successor states).
 - Threads continue exploring **in parallel** until the puzzle is solved.
 - Once a solution is found, other threads may terminate, saving computation.
- Other examples:

- Parallel **game tree search** (e.g., chess AI).
 - Parallel **branch-and-bound** for optimization problems.
-

Distributed Computing Perspective

- Nodes in a distributed system explore **different parts of the solution space independently**.
 - Reduces **overall execution time**, especially for large search spaces.
 - **Example (Distributed Computing): Solving 0/1 Integer Programming**
 - Each node is assigned a subset of possible variable assignments.
 - Nodes evaluate their assignments **independently**, pruning invalid solutions.
 - When one node finds a valid solution, it may notify others to **stop unnecessary computation**.
 - Other examples:
 - Distributed **theorem proving**: nodes explore different proof paths.
 - Distributed **genetic algorithms**: nodes evolve different populations in parallel.
-

Steps in Exploratory Decomposition

1. Identify the **search space** or set of possible solutions.
 2. Generate **independent tasks** from the current state or node.
 3. Assign tasks to **threads (parallel)** or **nodes (distributed)**.
 4. Explore tasks **simultaneously**.
 5. Optionally, **terminate remaining tasks** when the first solution is found.
-

Key Points

- Best suited for **search and optimization problems**.
 - Tasks are often **dynamic and irregular**.
 - In **parallel computing**, threads can explore state space concurrently for **speedup**.
 - In **distributed computing**, nodes explore different parts of the space, reducing overall **computation time and communication overhead**.
-

AI Game Playing (Chess Engines, like AlphaZero)

The system explores thousands of possible moves simultaneously to find the best one.

Each processor explores a different game path.

Speculatory Decomposition

- In some applications, dependencies between tasks are not known a-priori.
- For such applications, it is impossible to identify independent tasks.
- There are generally two approaches to dealing with such applications: conservative approaches, which identify independent tasks only when they are guaranteed to not have dependencies, and, optimistic approaches, which schedule tasks even when they may potentially be erroneous.
- Conservative approaches may yield little concurrency and optimistic approaches may require roll-back mechanism in the case of an error.

Here's a clear explanation of **Speculative (Speculatory) Decomposition** in the context of **Parallel and Distributed Computing**, with examples:

Speculative Decomposition

- **Definition:**

Speculative decomposition is used when **dependencies between tasks are not known in advance.**

- It allows tasks to be executed **before it is certain they are needed**, hoping the execution will be useful.
- It is particularly useful for **dynamic or unpredictable computations**.

- **Key Idea:**

- **Conservative approach:** Execute only tasks that are guaranteed independent → less parallelism.
- **Optimistic approach:** Execute tasks speculatively → may require **rollback** if dependencies are violated.

Parallel Computing Perspective

- Tasks are **executed speculatively on different cores**.
- If a speculative task later conflicts with dependencies, the results may be **discarded or rolled back**.

- **Example (Parallel Computing): CPU Speculative Execution**
 - A CPU predicts the branch of a conditional statement and executes instructions ahead of time.
 - If the prediction is correct → saves time.
 - If the prediction is wrong → rollback occurs, and correct instructions are executed.
 - Other examples:
 - Parallel **out-of-order execution** in multi-core processors.
 - Speculative parallel algorithms for dynamic graphs where future edges or nodes may or may not exist.
-

Distributed Computing Perspective

- Nodes execute tasks **before it is confirmed whether they are needed**, potentially reducing idle time.
 - **Rollback mechanisms** may be required if speculative computations are invalid.
 - **Example (Distributed Computing): Speculative MapReduce**
 - In Hadoop or Spark, tasks are speculatively run on multiple nodes to **handle straggler tasks** (slow tasks).
 - The first completed task result is used; other speculative tasks are terminated.
 - Other examples:
 - Distributed databases: speculative query execution to reduce latency.
 - Predictive prefetching in distributed file systems: nodes fetch data speculatively before it's requested.
-

Steps in Speculative Decomposition

1. Identify tasks whose dependencies are **uncertain or dynamic**.
 2. Decide between **conservative** or **optimistic** approach.
 3. Execute tasks speculatively on cores or nodes.
 4. Monitor for **conflicts or invalid results**.
 5. Rollback or discard tasks if necessary.
-

Key Points

- Useful for **dynamic, unpredictable computations** where dependencies are not known in advance.
- **Conservative approach:** safe but limited parallelism.
- **Optimistic approach:** higher parallelism but may need rollback.
- **Parallel systems:** improves CPU or core utilization via speculative execution.

Search Engines (Google)

When you start typing a query, Google starts fetching results for multiple possible completions whichever you choose, it's ready instantly.

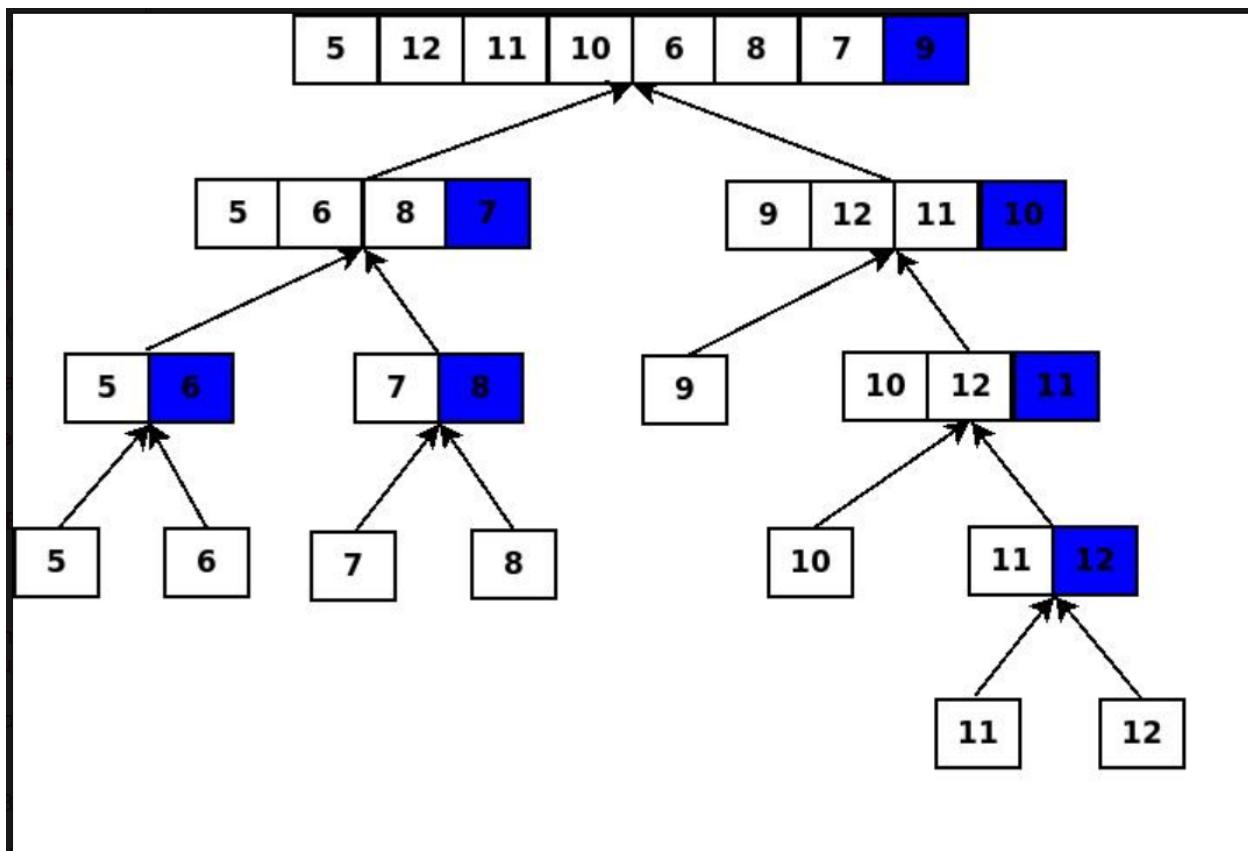
- **Distributed systems:** helps reduce latency by precomputing possible tasks and mitigating stragglers.

1. Recursive Decomposition

It is based on recursion.

e.g. Quick

Sort



2. Data Decomposition

$$\begin{array}{|c|c|} \hline a_1 & a_2 \\ \hline \hline a_3 & a_4 \\ \hline \end{array} *
 \begin{array}{|c|c|} \hline b_1 & b_2 \\ \hline \hline b_3 & b_4 \\ \hline \end{array} =
 \begin{array}{|c|c|} \hline c_1 & c_2 \\ \hline \hline c_3 & c_4 \\ \hline \end{array}$$

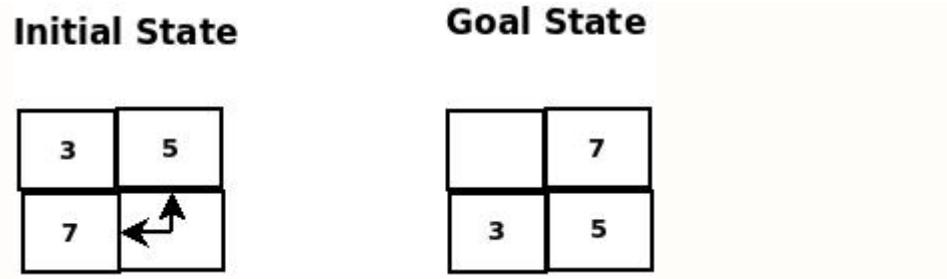
$$c_1 = a_1 * b_1 + a_2 * b_3$$

$$c_2 = a_1 * b_2 + a_2 * b_4$$

$$c_3 = a_3 * b_1 + a_4 * b_3$$

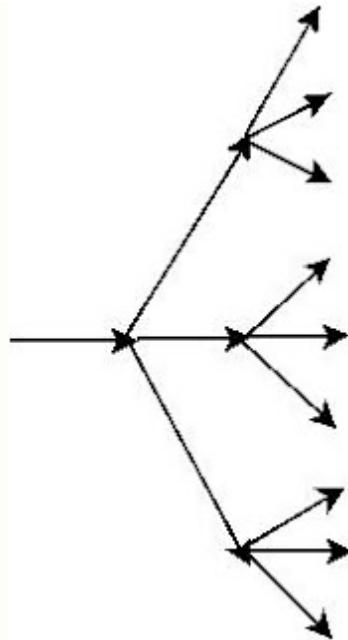
$$c_4 = a_3 * b_2 + a_4 * b_4$$

3. Exploratory Decomposition



Speculative Decomposition

e.g. Switch case



Certainly! Here are visual diagrams illustrating the four decomposition techniques in parallel and distributed computing:

1. Recursive Decomposition

This technique is commonly used in divide-and-conquer algorithms, where a problem is recursively divided into smaller subproblems.

Example: Quicksort

Source: [Computer Revolution](#)

2. Data Decomposition

In data decomposition, large datasets are partitioned into smaller chunks, and each chunk is processed independently.

Example: Matrix Multiplication

Source: [Computer Revolution](#)

3. Exploratory Decomposition

This approach is used when the problem involves exploring a state space, such as in game playing or optimization problems.

Example: 15-Puzzle

Source: [Computer Revolution](#)

4. Speculative Decomposition

In speculative decomposition, tasks are executed before it is certain they are needed, to improve performance.

Example: Switch Case Execution