Parallel Algorithm and Design – Preliminaries

Parallel algorithms are algorithms that can execute multiple operations simultaneously, leveraging parallel computing environments to solve problems faster and more efficiently.

Designing efficient parallel algorithms requires an understanding of key concepts, terminologies, and models.

1. Introduction to Parallel Algorithms

A **Parallel Algorithm** is an algorithm designed to perform computations simultaneously on multiple processors to reduce execution time and solve large-scale problems efficiently.

- **Parallelism:** The ability to perform multiple operations or tasks simultaneously.
- **Concurrency:** Execution of several instruction sequences at the same time.

2. Levels of Parallelism

1) **Bit-Level Parallelism:** Operations on multiple bits simultaneously (e.g., 64-bit processors).
2) **Instruction-Level Parallelism (ILP):** Multiple instructions are executed simultaneously in a single CPU cycle (pipelining).
3) **Data-Level Parallelism (DLP):** The same operation is performed on multiple data points (vectorization).
4) **Task-Level Parallelism (TLP):** Different tasks or threads run concurrently on different processors.

1)Bit-Level Parallelism: Operations on multiple bits simultaneously (e.g., 64-bit processors).

Example of **Bit-Level Parallelism**

Bit-level parallelism is a form of parallel computing where multiple bits are processed simultaneously within a single processor instruction.

It leverages the processor's ability to handle operations on large-bit registers (e.g., 32-bit, 64-bit) in parallel, increasing the speed of computations.

Example 1: 32-bit Addition Using Bit-Level Parallelism

Suppose we add two 32-bit binary numbers:

```
sql

  A = 00000000000000000000000000011001  (25 in decimal)
  B = 00000000000000000000000000010101  (21 in decimal)
-------------------------------------
Sum = 00000000000000000000000000101110  (46 in decimal)
```

- **Without parallelism:** Each bit would be added sequentially.
- **With bit-level parallelism:** The entire 32-bit addition occurs **simultaneously** in one operation.

2) Instruction-Level Parallelism (ILP) Works – Pipelining

In a traditional processor (without ILP), each instruction is completed one after the other. With **ILP**

and pipelining, different stages of multiple instructions overlap, similar to an assembly line.

Pipelining Stages:

1. **IF (Instruction Fetch):** Retrieve the instruction from memory.
2. **ID (Instruction Decode):** Decode the instruction to understand the operation.
3. **EX (Execute):** Perform the operation (e.g., addition or multiplication).
4. **MEM (Memory Access):** Read from or write to memory.
5. **WB (Write Back):** Store the result back into the register.

With Pipelining (ILP in Action):

- Multiple instructions are processed simultaneously in different pipeline stages.
- Total time = **8 cycles** for all instructions (instead of 20).

Cycle	I1 (ADD)	I2 (SUB)	I3 (MUL)	I4 (AND)
1	IF			
2	ID	IF		
3	EX	ID	IF	
4	MEM	EX	ID	IF
5	WB	MEM	EX	ID
6		WB	MEM	EX
7			WB	MEM
8				WB

Example of Data-Level Parallelism (DLP) – Vectorization

Data-Level Parallelism (DLP) is a form of parallel computing where the same operation is performed on multiple data points simultaneously. It is often achieved using techniques like vectorization, SIMD (Single Instruction, Multiple Data), and parallel loops.

Example 1: Vectorized Addition (SIMD)

Suppose we need to add two arrays of numbers:

Without Data-Level Parallelism (Sequential Processing):

```mathematica
A = [1, 2, 3, 4]
B = [5, 6, 7, 8]

Result:
C[0] = A[0] + B[0] = 1 + 5 = 6
C[1] = A[1] + B[1] = 2 + 6 = 8
C[2] = A[2] + B[2] = 3 + 7 = 10
C[3] = A[3] + B[3] = 4 + 8 = 12
```

- Total **4 operations** executed **sequentially**.

4) Task-Level Parallelism (TLP): Different Tasks Executing Concurrently

Task-Level Parallelism (TLP) is a type of parallel computing where multiple **different tasks or threads** execute **concurrently**, typically on **multiple processors or cores**. Each thread performs a distinct operation independently, and they work together to complete the overall process more quickly.

How Task-Level Parallelism (TLP) Works:

- **Focus:** Executes different tasks simultaneously rather than splitting the same task into multiple parts.
- **Threads:** Each task runs in its own **thread or process**.
- **Processors:** Tasks can run on multiple **CPU cores** or on **distributed systems**.
- **Concurrency:** TLP is **asynchronous**; tasks may finish at different times.

Parallel Algorithm Design Techniques

- **Divide and Conquer:** Break a problem into subproblems, solve them in parallel, and merge results.
- **Pipelining:** Organize computations in stages for continuous data processing.
- **Task Parallelism:** Divide the problem into tasks executed concurrently.
- **Data Parallelism:** Perform the same operation on different data simultaneously

1) **Parallel Algorithm Design Technique: Divide and Conquer**

Divide and Conquer is a fundamental algorithm design technique that breaks a problem into smaller subproblems, solves each subproblem **in parallel**, and then **combines (merges)** their results to form the solution to the original problem.

2) How Pipelining Works:

In pipelining, multiple tasks are divided into stages. Each stage processes a portion of the task and passes it to the next stage. New inputs enter the pipeline before the previous outputs are complete, maximizing throughput.

Stages of Pipelining:

1. **Fetch:** Retrieve the input or instruction.
2. **Decode:** Interpret the input or instruction.
3. **Execute:** Perform the operation.
4. **Memory:** Access or store data.
5. **Write-Back:** Save the result.

Example 1: Pipelining in a CPU (Instruction Pipeline)

In modern processors, pipelining allows multiple instructions to be executed simultaneously:

Cycle	Instruction 1 (ADD)	Instruction 2 (SUB)	Instruction 3 (MUL)
1	Fetch		
2	Decode	Fetch	
3	Execute	Decode	Fetch
4	Memory	Execute	Decode
5	Write-Back	Memory	Execute

- Each instruction moves through the pipeline stages simultaneously.
- **Throughput** increases as multiple instructions are in different stages at the same time.

3) **Task Parallelism:** Divide the problem into tasks executed concurrently.

Parallel Pipelining in Image Processing

In an image processing pipeline:

1. **Stage 1:** Read Image (Input)
2. **Stage 2:** Apply Filter (Processing)
3. **Stage 3:** Compress Image (Processing)
4. **Stage 4:** Save Image (Output)

Data Parallelism: Perform the same operation on different data simultaneously