1. Parallelism in High Performance Computing

Modern processors have reached physical limits in increasing clock frequency. Instead of making a single processor infinitely faster, computer architects improve performance by **performing multiple operations simultaneously**, a concept known as **parallelism**.

Parallelism exists at several levels inside a computing system. Some forms of parallelism are implemented directly by the hardware, while others require support from compilers or programmers. Each level contributes to improving the overall performance of High Performance Computing (HPC) applications.

The major levels of parallelism are

- Bit-Level
- Instruction-Level
- Data Level
- Task Level

4.1. Bit-Level Parallelism (BLP)

Bit-Level Parallelism (BLP) refers to increasing processor performance by increasing the number of bits that the processor can handle in a single operation. Instead of processing small groups of bits, modern processors operate on larger word sizes, allowing more information to be processed per instruction.

4.2. Instruction-Level Parallelism (ILP)

Instruction-Level Parallelism (ILP) is the ability of a processor to execute multiple independent instructions simultaneously. Although a program appears sequential, many instructions do not depend on one another. Modern CPUs exploit this independence to increase performance.

Techniques used by modern CPUs:

- Pipelining: Overlaps fetch, decode, execute, write-back stages of different instructions
- Superscalar execution: Multiple execution units running in parallel
- Out-of-Order execution: Instructions executed in any order as long as data dependencies are respected

- Branch prediction: Speculatively execute instructions before knowing the branch outcome

4.3 Task Parallelism and Data Parallelism

Parallel computing can be achieved by dividing either the data or the tasks of an application. Based on this approach, parallelism is broadly classified into Data-Level Parallelism (DLP) and Task-Level Parallelism (TLP). These techniques enable modern computing systems to execute multiple operations simultaneously, thereby improving performance and reducing the execution time of computationally intensive applications.

4.4 Data-Level Parallelism (DLP)

The same operation applied to multiple data items simultaneously. It is also called SIMD (Single Instruction, Multiple Data). GPUs are the extreme example of DLP — thousands of CUDA cores executing the same instruction on different data elements in parallel.

Applications: Image processing, matrix operations, signal processing, deep learning inference.

Scalar: a [0] = b[0] + c[0]; (one add)

SIMD: a [0:7] = b[0:7] + c[0:7]; (eight adds simultaneously)

4.5 Task-Level Parallelism (TLP)

Task-Level Parallelism refers to executing **multiple independent tasks concurrently** on different processors or cores. Different tasks (potentially different code) executing simultaneously on different processors. Unlike DLP, each task may execute different instructions on different data.

- Process-level: Multiple OS processes (MPI ranks in HPC)
- Thread-level: Multiple threads within a process sharing memory (OpenMP)
- Function-level: Pipeline of functions, each on different data

Difference between Task Parallelism and Data parallelism

Task Parallelism	Data Parallelism
Different tasks run simultaneously	Same task runs on different data simultaneously
Tasks can be different operations	Operation is identical across all data
Heterogeneous work	Homogeneous work
More communication needed	Less communication needed
Hard to load balance	Easy to load balance
Less scalable	Highly scalable
Used in irregular workloads	Used in regular, structured workloads
Example — Web server (auth, logging, request handling)	Example — Matrix multiply, image filters
Model — MPI, task graphs	Model — CUDA, OpenMP parallel-for