Lec 1

- Main Types of HPC Systems:
  - The commodity HPC clusters:
    - ✓ Clusters of servers interconnected using a high speed connection.
    - ✓ Its sizes range from tens of servers to tens of thousands of servers.
    - ✓ Solving the problems by dividing them into smaller problems (Divide and Conquer).
    - ✓ The most popular form of HPC
  - Dedicated Supercomputers:
    - ✓ Providing high performance chips.
    - ✓ Providing processors that work faster on large amounts of data (Vectorization).
    - ✓ Providing more than a CPU on the same chip.
    - ✓ CPUs on the same chips share the address space (Memory).
    - ✓ Solving the problems faster.
- Proc. speed measured in Gigahertz (hertz = cycles per sec.)
- One operation may take several cycles.
- How many operation/sec done in 1 GHz Processor?
  - $>10^8$
- Parallel programming is explicitly indicating how different portions of the computation may be executed concurrently by different processors
- Parallel Architectures:
  - Multi-Processors Systems:
    - ✓ Multiple-CPU computer with shared memory.
    - ✓ The same address on different CPUs refers to the same memory location.
  - Multi-Computers Systems:
    - ✓ Has disjoint local address spaces (memory).
    - ✓ Each CPU has direct access to its local memory only.
    - ✓ The same address on different CPUs refers to different memory locations.
    - ✓ CPUs interact with each other by passing messages
- Centralized multiprocessor: all the primary memory is in one place
- Distributed multiprocessor: the primary memory is distributed among the processors.
- step process for designing parallel algorithms:
  - Decomposition -> partitioning the problem into tasks.
  - Communication -> connecting tasks to each other.
  - Agglomeration -> reducing the number of tasks to reduce communication overhead.
  - Mapping -> assigning tasks to processes.
- HPC Domain of Applications:
  - Finite element analysis.
  - Medical Physics and Medicine.
  - Chemistry and Biochemistry.

- Climate research.
- Biophysics and Bioinformatics
- Nanoscience

Lec 2

- Why Parallel Computer?
  - Tremendous advances in microprocessor technology
  - Processor is now capable of executing multiple instruction in the same cycle
  - The ability of memory system to feed data to processor at required rate increased
  - significant innovations in architecture and software have addressed the mitigation of bottlenecks posed by data-Path and memory
- the use of more transistors improves performance in two ways:
  - Parallelism -> multiple operations done at once (less processing time)
  - Locality -> data references performed close to the processor (less memory latency)
- High-performance computing (HPC) -> the use of parallel processing for running advanced application programs efficiently, reliably and quickly.
- Core -> a single computing unit with its own independent control
- Multicore -> is a processor having several cores that can access the same memory concurrently
- Tasks -> a computation is decomposed into several parts that can be computed in parallel
- Finding enough parallelism is (one of the) critical steps for high performance (Amdahl's law).
- Execution time -> The time elapsed between the beginning and the end of its execution.
- Speedup -> The ration between serial and parallel time (Speedup= T_s /T_p)
- Efficiency -> Ratio of speedup to the number of processors (Efficiency= Speedup/P)
- Amdahl's Law:
  - Used to predict maximum speedup using multiple processors.
  - f = fraction of work performed sequentially.
  - (1 - f) = fraction of work that is parallelizable.
  - On 1 CPU: T_1 = f + (1 – f) = 1
  - On P processors: T_p =(f + (1−f))/p
  - Speedup: T_1 / T_p = 1/(f+(1−f)/p) < 1 / f
  - Speedup limited by sequential part
- How is parallelism expressed in a program?

IMPLICITLY	EXPLICITLY
It is a characteristic of a programming language that allows a compiler or interpreter to automatically exploit the parallelism inherent to the computations expressed by some of the language's constructs.	it is the representation of concurrent computations by means of primitives in the form of special-purpose directives or function calls
Define tasks only, rest implied; or define tasks and work decomposition rest implied	tasks, work decomposition, data decomposition, communication, synchronization
OpenMP	MPI

A programmer does not need to worry about task division or process communication	A skilled parallel programmer takes advantage of explicit parallelism to produce very efficient code.
A programmer does not need to worry about task division or process communication	The absolute programmer control over the parallel execution.
focusing instead in the problem that his or her program is intended to solve	programming with explicit parallelism is often difficult, especially for non computing specialists

- Degree of Parallelism → How many people doing the work
- Initialization → What is needed to begin the work
- Work distribution →Who does what
- Data/IO access → Access to work part
- Communication → Whether they need info from each other to finish their own job
- Synchronization -> When are they all done
- Sources of overhead in parallel programs:
  - Inter process interaction
    - ✓ The time spent communicating data between processing elements is usually the most significant source of parallel processing overhead.
  - Idling:
    - ✓ Processes may become idle due to many reasons such as load imbalance, synchronization, and presence of serial components in a program.
  - Excess Computation:
    - ✓ The fastest known sequential algorithm for a problem may be difficult or impossible to parallelize, forcing us to use a parallel algorithm based on a poorer but easily parallelizable sequential algorithm.

Lec 3

- A computing platform includes a hardware architecture and a software framework (including application frameworks)
- Typical platforms include a (computer architecture, operating system, programming languages, related user interface)
- architectural concepts relate to parallel processing :
  - Implicit parallel platforms.
  - Explicit parallel platforms.
- Explicit Parallelism
  - Elements of a Parallel Computer Hardware
    - ✓ Hardware -> ▪ Multiple Processors ▪ Multiple Memories ▪ Interconnection Network System Software
    - ✓ System Software -> ▪ Parallel Operating System ▪ Programming Constructs to Express/Orchestrate Concurrency Application Software
    - ✓ Application Software -> Parallel Algorithms
  - Goal -> Utilize the Hardware, System, & Application Software to either:
    - ✓ Achieve Speedup.
    - ✓ Solve problems requiring a large amount of memory.

- Implicit Parallelism:
  - Concerning Memory- processor data path bottlenecks, microprocessor designer invent (trend) alternate routs to cost effective performance.
  - Execution of multiple instruction in a single clock cycle
  - the mechanism used by various processors to support Execution of multiple instruction in a single clock cycle:
    - ✓ Pipelining and superscalar execution
    - ✓ Very long instruction word processors.
- Limitation: / Scheduling of instructions is determined by number of factors:
  - True Data Dependency -> The result of one operation is an input to the next.
  - Resource Dependency -> Two operations require the same resource.
  - Branch Dependency -> This requires very accurate branch prediction.
- Pipelining and superscalar execution
  - the speed of a single pipeline is limited by largest atomic task in pipeline
  - Pipeline rate limited by slowest pipeline stage
  - Multiple pipeline operate simultaneously using different resources
  - Potential speedup = number pipe stages
  - Unbalanced length of pipe stages reduce speedup
  - Time to fill pipeline and time to drain reduce speedup
  - Superscalar execution -> the ability of a processor to issue multiple instructions in the same cycle
- Superscalar execution
  - The scheduler, a piece of hardware looks at large number of instructions in an instruction queue and selects appropriate number of instructions to execute concurrently based on these factors.
- Issue Mechanisms:
  - in-order issue -> if the second instruction cannot be issued because it has a data dependency with the first, only one instruction is issued in the cycle
  - dynamic issue -> if the second instruction has data dependencies with the first, but the third instruction does not, the first and third instructions can be co-scheduled.
- Performance of in-order issue is generally limited.
- The parallelism extracted by superscalar processor is often limited by the instruction look ahead.
- Very long instruction word processors (VLIW)
  - processors relies on the compiler to resolve dependencies & resource availability at compile time where:
    - ✓ Instruction that can be executed concurrently are packed into groups given to processor as a single long instruction word to be executed on multiple functional units at the same time.
    - ✓ Very sensitive to compilers ability to detect data & resource dependencies
- Memory System Performance is mainly captured by two parameters, **latency and bandwidth**.
- The effective performance of a program on computer relies -> (Speed of processor, Ability of memory to feed data to processor)

- Latency -> is the time from the issue of a memory request to the time the data is available at the processor.
- Bandwidth->is the rate at which data can be pumped to the processor by the memory system.
- To handle mismatch between processor and DRAM:
  - Caches are small and fast memory elements between the processor and DRAM.
  - This memory acts as a low-latency, high-bandwidth storage.
  - If a piece of data is repeatedly used, the effective latency of this memory system can be reduced by the cache.
- Parallel Computing Platform:
  - Logical organization (programmer view's of platform)
  - Physical organization (the actual hardware architecture)
- A dichotomy -> is any splitting of a whole into **exactly two non-overlapping parts**, meaning it is a procedure in which a whole is divided into two parts.
- An explicitly parallel program must specify accurately the interaction between concurrent subtasks.
- Logical organization:
  - Control structure -> ways to express parallel tasks
    - ✓ Every instruction inside a program can viewed as parallel task
    - ✓ Every program of a group of programs can be task
  - Communication models -> mechanism for specifying interaction between tasks
- Task -> A logically discrete section of computational work. A task is typically a program or program-like set of instructions that is executed by a processor.
- Parallel Task -> A task that can be executed by multiple processors safely
- Parallelism can be expressed at various levels of granularity (is a qualitative measure of the ratio of computation to communication)
- Periods of computation are typically separated from periods of communication by synchronization events.

Fine-grain Parallelism	Coarse-grain Parallelism
Relatively small amounts of computational work are done between communication events	Relatively large amounts of computational work are done between communication/synchronization events
Low computation to communication ratio	high computation to communication ratio
Implies high communication overhead and less opportunity for performance enhancement	Implies more opportunity for performance increase
Facilitates load balancing	Harder to load balance efficiently

- In Fine-grain Parallelism, If granularity is too fine it is possible that the overhead required for communications and synchronization between tasks takes longer than the computation.
- The most efficient granularity is dependent on the algorithm and the hardware environment in which it runs.
- Fine-grain parallelism can help reduce overheads due to load imbalance
- In most cases the overhead associated with communications and synchronization is high relative to execution speed so it is advantageous to have coarse granularity.