CS 3006 Parallel and Distributed Computer

Fall 2022

1. Learn about parallel and distributed computer architectures.(1)
2. Implement different parallel and distributed programming paradigms and algorithms using Message-Passing Interface (MPI) and OpenMP.(4)
3. Perform analytical modelling, dependence, and performance analysis of parallel algorithms and programs.(2)
4. Use Hadoop or MapReduce programming model to write bigdata applications.(5)

Week # 2 – Lecture # 3, 4, 5, 6

29th, 31st August 2022

1st , 3rd Safar ul Muzaffar, 1444

Dr. Nadeem Kafi Khan

Lecture # 3 – Topics (Lab # 2)

- **Lab Related Announcements**
- **Lab Homework will be posted on GLR later today**.

- Theory Topics
  - Overhead in Parallel and Distributed Computing
  - Speed-up and Amdahl Law
  - Flynn's Taxonomy

Lab

- **Announcements only (10 mins)**: Lab work is worth 14 weightage and will be assessed based on in-lab assignments, home-work, assignments, Project. (I am planning to group two assignments as project in my sections), etc. as per the will of instructor. (**no assignments in theory only quizzes**).
- One Lab per week. Total of 15 labs. No Lab midterm or Final Exam. Starting from lab # 2, assignment and home work given in previous lab will be evaluated.
- Linux VM-based setup for OpenMP and MPI in the lab and on student home laptop/PC
- Basic Linux command lines and scripting (see attachments)

Distributed Computing	Parallel Computing
In distributed computing, a number of unified computers work towards a common task while communicating with each other with the help of message passing	In parallel computing, a task is divided into multiple sub-task which are then allotted to different processors on the same computer system.
Number of Computer Systems Involved	
Multiple physical computer systems are present in the same computer system.	A single physical computer system hosts multiple processors.
Dependency Between Processes	
There might not be much dependency between the processes.	There is more dependency between the process. Output of one might be the input of another.
Scalability	
The systems are easily scalable as there is no limitation on how many systems can be added to a network.	The systems that implement parallel computing have limited scalability.
Resource Sharing	
Computers have their own memory and processors.	All the processors share the same memory.
Synchronization	
The computers in the network have to implement synchronization algorithms.	All processors use the same master clock for synchronization.
Usage	
Generally preferred in places requiring high scalability.	Generally preferred in places requiring faster speed and better performance.

Some General Parallel Terminology

- **Observed Speedup**
  - Observed speedup of a code which has been parallelized, defined as:

$$\frac{\text{wall-clock time of serial execution}}{\text{wall-clock time of parallel execution}}$$

  - One of the simplest and most widely used indicators for a parallel program's performance.

The Amdahl's Law formula is

$$overall\ speedup = \frac{1}{(1-P) + \frac{P}{S}}$$

- P is the time proportion of the algorithm that can be parallelized.
- S is the speedup factor for that portion of the algorithm due to parallelization.

For example, suppose that we use our strategy to search for primes using 4 processors, and that 90% of the running time is spent checking 2k-digit random numbers for primality (after an initial 10% of the running time computing a list of k-digit primes). Then P = .90 and S = 4 (for 4-fold speedup). According to Amdahl's Law,

$$overall\ speedup = \frac{1}{(1-0.90) + \frac{0.90}{4}} = \frac{1}{0.225} = 3.077$$

0.1 + 0.225 = 0.325

This estimates that we will obtain about 3-fold speedup by using 4-fold parallelism.

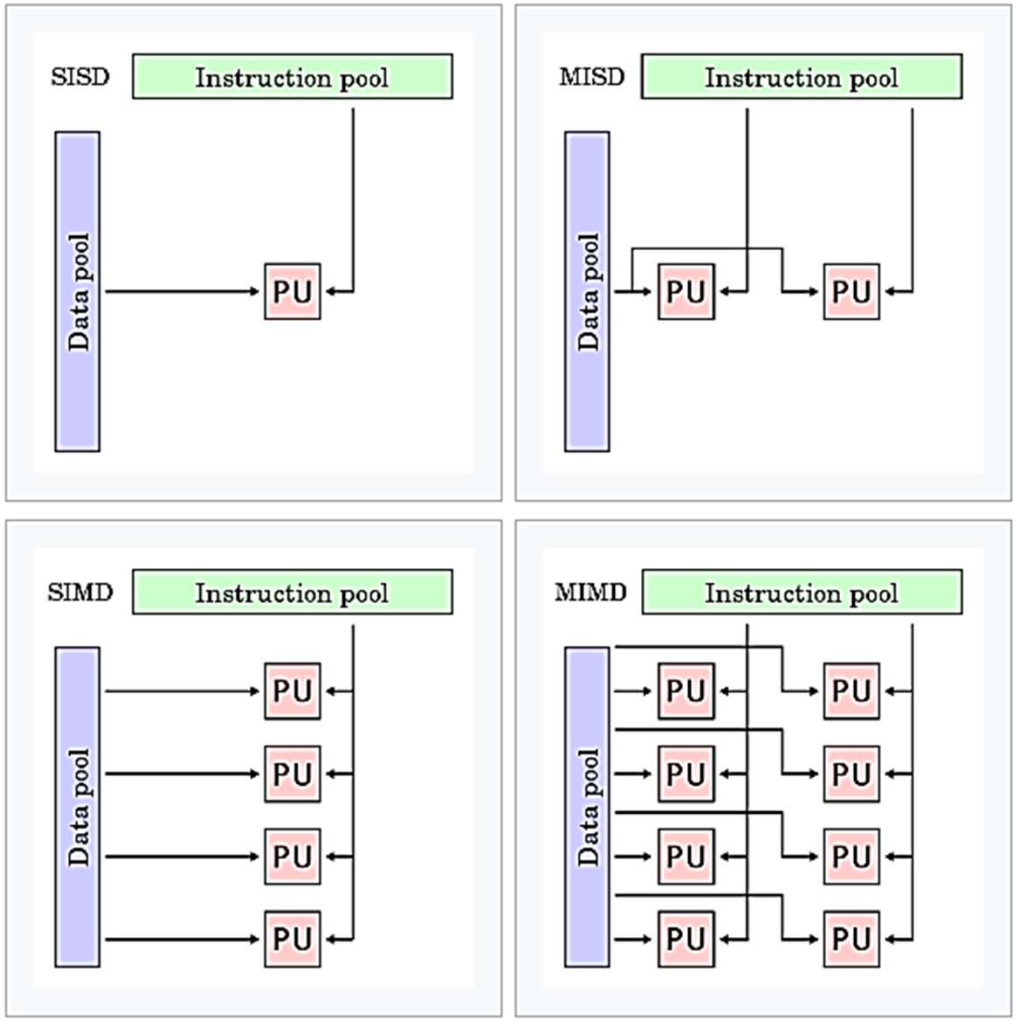
Some General Parallel Terminology

- **Parallel Overhead**
  - The amount of time required to coordinate parallel tasks, as opposed to doing useful work. Parallel overhead can include factors such as:
    - Task start-up time
    - Synchronizations
    - Data communications
    - Software overhead imposed by parallel compilers, libraries, tools, operating system, etc.
    - Task termination time
- **Massively Parallel**
  - Refers to the hardware that comprises a given parallel system - having many processors. **The meaning of many keeps increasing, but currently IBM Blue Gene/L pushes this number to 6 digits.**

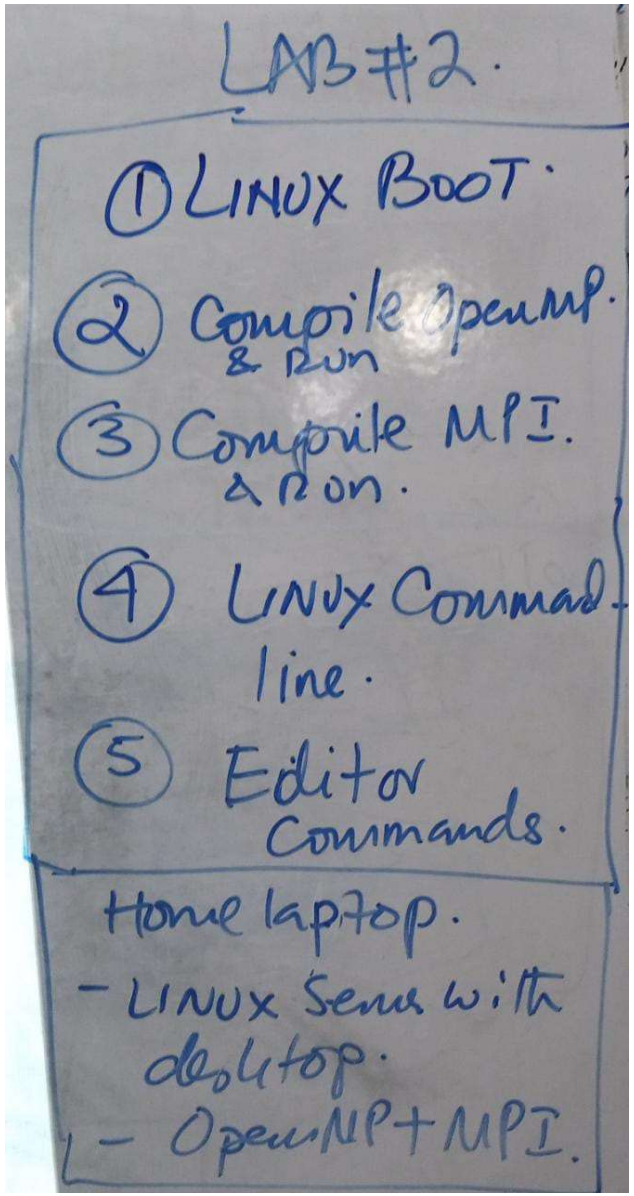
Flynn’s Taxonomy

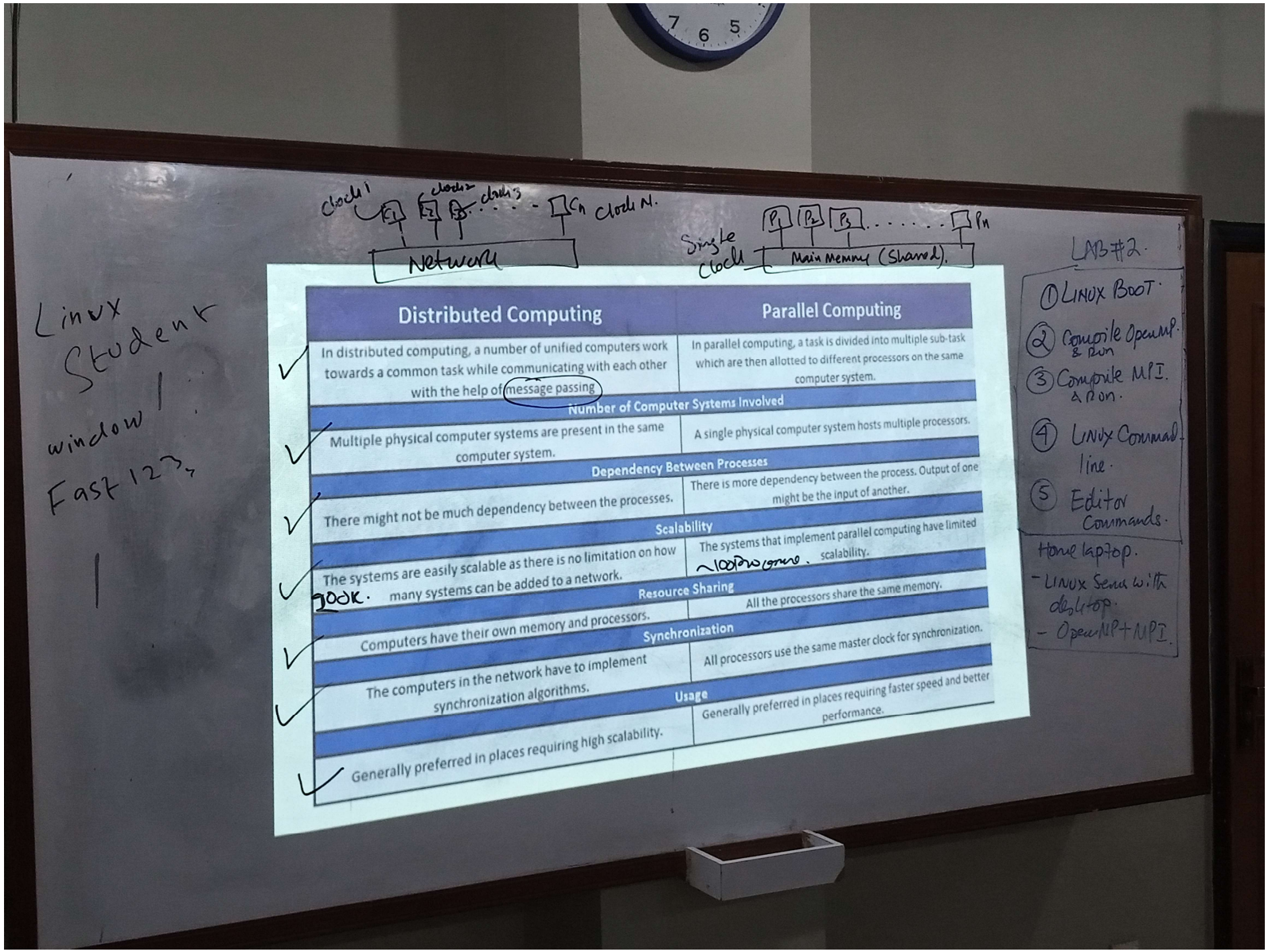
- Flynn's taxonomy is a classification of computer architectures, proposed by Michael J. Flynn in 1966 and extended in 1972.
- The classification system has stuck, and it has been used as a tool in design of modern processors and their functionalities.

Data Streams \ Instruction Streams	one	many
one	SISD traditional von Neumann single CPU computer	MISD May be pipelined Computers
many	SIMD Vector processors fine grained data Parallel computers	MIMD Multi computers Multiprocessors



PU = Processing Unit


LAB #2.
① LINUX BOOT.
② Compile OpenMP. & Run
③ Compile MPI. & Run.
④ LINUX Command line.
⑤ Editor Commands.
Home laptop.
- LINUX Server with desktop.
- OpenMP + MPI.



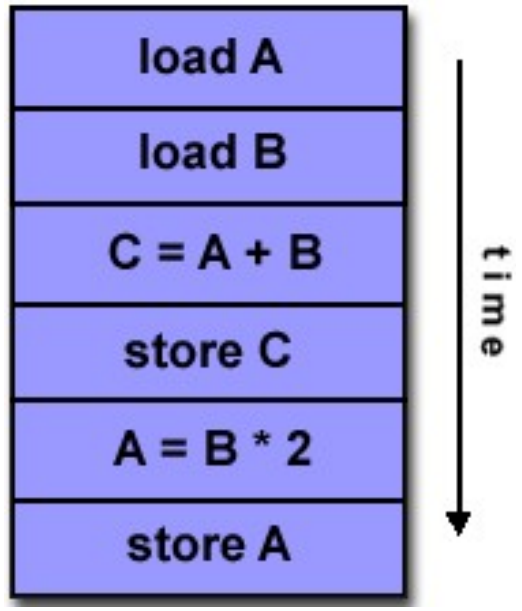
Distributed Computing	Parallel Computing
In distributed computing, a number of unified computers work towards a common task while communicating with each other with the help of message passing	In parallel computing, a task is divided into multiple sub-task which are then allotted to different processors on the same computer system.
Number of Computer Systems Involved	
Multiple physical computer systems are present in the same computer system.	A single physical computer system hosts multiple processors.
Dependency Between Processes	
There might not be much dependency between the processes.	There is more dependency between the process. Output of one might be the input of another.
Scalability	
The systems are easily scalable as there is no limitation on how many systems can be added to a network.	The systems that implement parallel computing have limited scalability.
Resource Sharing	
Computers have their own memory and processors.	All the processors share the same memory.
Synchronization	
The computers in the network have to implement synchronization algorithms.	All processors use the same master clock for synchronization.
Usage	
Generally preferred in places requiring high scalability.	Generally preferred in places requiring faster speed and better performance.

Lecture # 4 - Topics

- Flynn's Taxonomy (Contd.)
- Task Parallelism
- Data Parallelism
- Granularity – The ratio between computation to communication
- Memory Models
  - Shared Memory
  - Distributed Memory
  - Hybrid: Distributed Shared Memory

Single Instruction, Single Data (SISD)

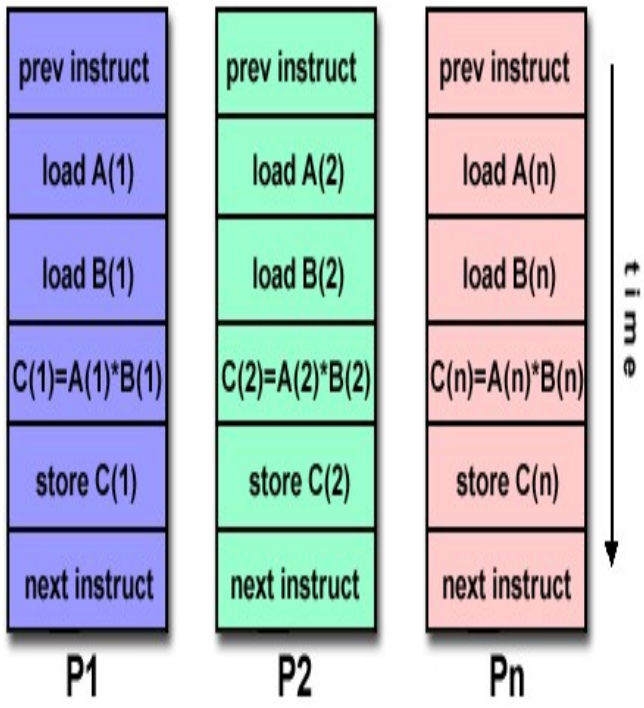
- A serial (non-parallel) computer
- Single instruction: only one instruction stream is being acted on by the CPU during any one clock cycle
- Single data: only one data stream is being used as input during any one clock cycle
- Deterministic execution
- This is the oldest and until recently, the most prevalent form of computer
- Examples: most PCs, single CPU workstations and mainframes



Single Instruction, Multiple Data (SIMD)

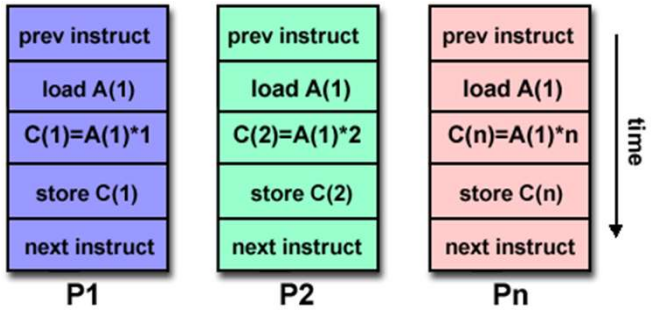
- A type of parallel computer
- **Single instruction**: All processing units execute the same instruction at any given clock cycle
- **Multiple data:** Each processing unit can operate on a different data element
- This type of machine typically has an instruction dispatcher, a very high-bandwidth internal network, and a very large array of very small-capacity instruction units.
- Best suited for specialized problems characterized by a high degree of regularity, such as image processing.
- Synchronous (lockstep) and deterministic execution
- Two varieties: Processor Arrays and Vector Pipelines

- Examples: Vectorization is a prime example of SIMD in which the same instruction is performed across multiple data. A variant of SIMD is single instruction, multi-thread (SIMT), which is commonly used to describe GPU workgroups.



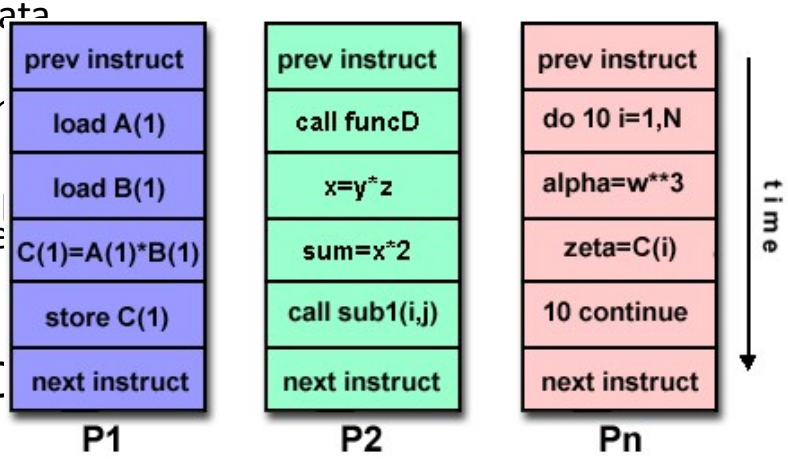
Multiple Instruction, Single Data (MISD)

- A single data stream is fed into multiple processing units.
- Each processing unit operates on the data independently via independent instruction streams. This is not a common architecture.
- Few actual examples of this class of parallel computer have ever existed. One is the expérimental Carnegie-Mellon C.mmp computer (1971).
- Some conceivable uses might be:
  - multiple frequency filters operating on a single signal stream
  - multiple cryptography algorithms attempting to crack a single coded message.
  - **Redundant computation on the same data. This is used in highly fault-tolerant approaches such as spacecraft controllers. Because spacecraft are in high radiation environments, these often run two copies of each calculation and compare the output of the two.**



Multiple Instruction, Multiple Data (MIMD)

- Currently, the most common type of parallel computer. Most modern computers fall into this category.
- Multiple Instruction: every processor may be executing a different instruction stream
- Multiple Data: every processor may be working with a different data stream
- Execution can be synchronous or asynchronous, deterministic or n deterministic
- Examples: most current supercomputers, networked parallel com "grids" and multi-processor SMP computers - including some type PCs.
- The final category has parallelization in both instructions and data and is referred to as MIMD. This category describes multi-core parallel architectures that comprise the majority of large parallel systems.



Definitions of Data and Task Parallelism

- Data parallel computation:
  - Perform the same operation to different items of data at the same time; the parallelism grows with the size of the data.
- Task parallel computation:
  - Perform distinct computations -- or tasks -- at the same time; with the number of tasks fixed, the parallelism is not scalable.

Task Parallel Model

The task parallel model can be characterized as follows:

- The program is split into a number of tasks
- Each task is assigned to a specific processor
- The data necessary for each task is sent to the appropriate processor

Data Parallel Model

The data parallel model can be characterized as follows:

- Data is distributed over the processors
- Each processor works on a different part of the same data structure

Granularity

Granularity is the ratio of computation to communication.

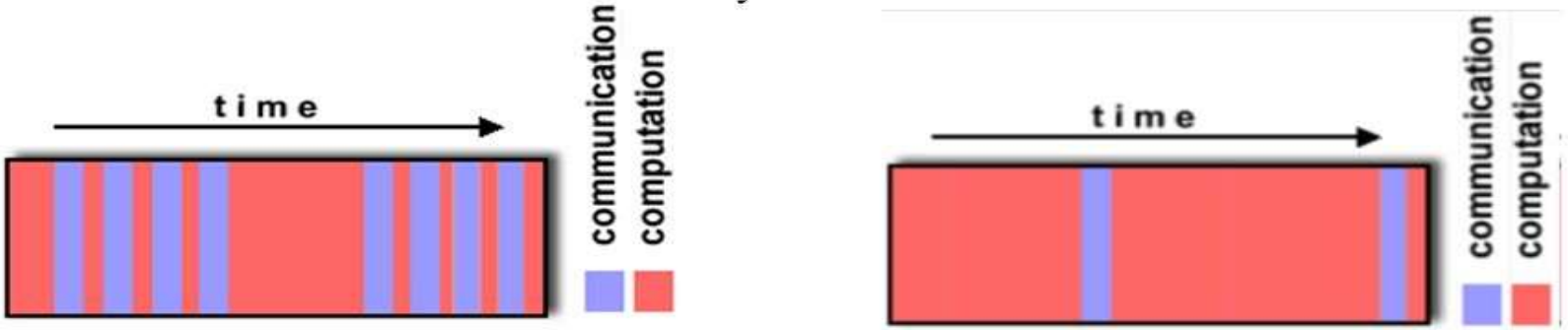
Periods of computation are typically separated from periods of communication by synchronization events.

Fine-grain Parallelism: Relatively small amounts of computational work are done between communication events.

Facilitates load balancing and Implies high communication overhead and less opportunity for performance enhancement

Coarse-grain Parallelism: Relatively large amounts of computational work are done between communication/synchronization events.

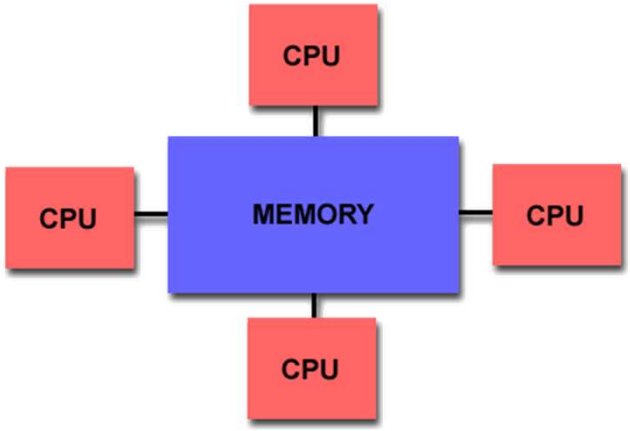
Harder to load balance efficiently



Memory architectures

- Shared Memory
- Distributed Memory
- Hybrid Distributed-Shared Memory

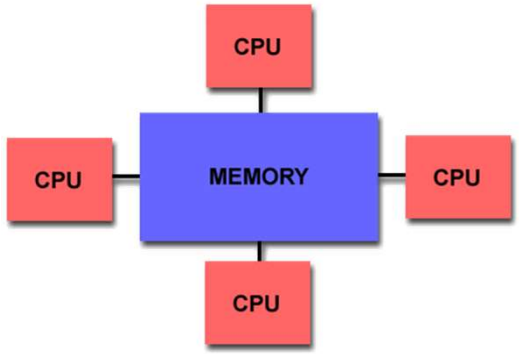
Shared Memory



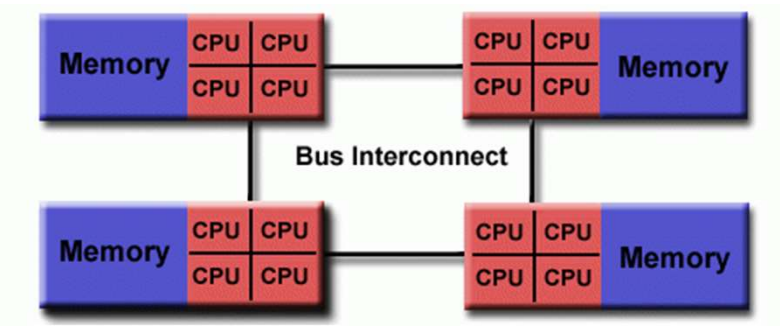
- Shared memory parallel computers vary widely, but generally have in common the ability for all processors to access all memory as global address space.
- Multiple processors can operate independently but share the same memory resources.
- Changes in a memory location effected by one processor are visible to all other processors.

Shared Memory : UMA vs. NUMA

- **Uniform Memory Access (UMA):**
  - Most commonly represented today by Symmetric Multiprocessor (SMP) machines having identical processors.
  - Equal access and access times to memory



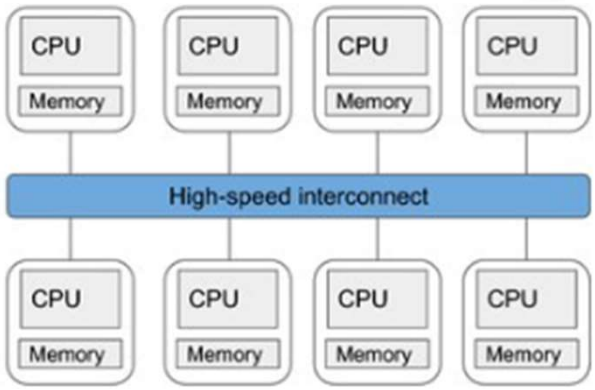
- **Non-Uniform Memory Access (NUMA):**
  - Two or more SMPs. Physically linked using a network.
  - One SMP can directly access memory of another SMP
  - Access time of own memory is less than Memory access across link.
  - Network access introduces variable delays.



Shared Memory: Pro and Con

- **Advantages**
  - Global address space provides a user-friendly programming perspective to memory
  - Data sharing between tasks is both fast and uniform due to the proximity of memory to CPUs
- **Disadvantages**:
  - Primary disadvantage is the lack of scalability between memory and CPUs. Adding more CPUs can geometrically increases traffic on the shared memory-CPU path, and for cache coherent systems, geometrically increase traffic associated with cache/memory management.
  - Programmer responsibility for synchronization constructs that insure "correct" access of global memory.
  - Expense: it becomes increasingly difficult and expensive to design and produce shared memory machines with ever increasing numbers of processors.

Distributed Memory

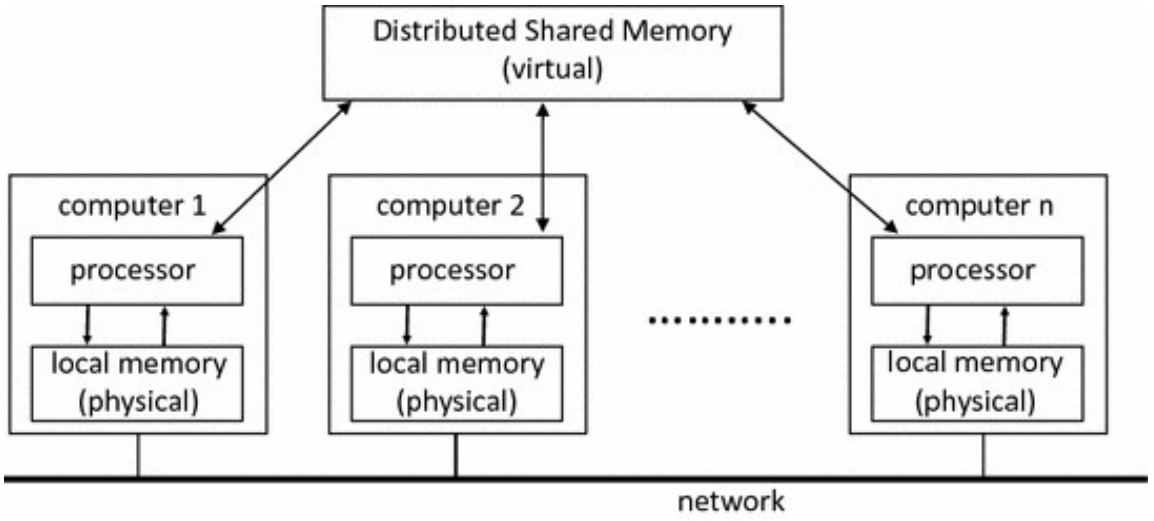


- Distributed memory systems require a communication network to connect inter-processor memory.
- Processors have their own local memory. Memory addresses in one processor do not map to another processor, so there is no concept of global address space across all processors.
- Each processor has its own local memory, it operates independently. Changes it makes to its local memory have no effect on the memory of other processors.
- When a processor needs access to data in another processor, it is usually the task of the programmer to explicitly define how and when data is communicated, including synchronization between tasks.

Distributed Memory: Pro and Con

- Advantages
  - Memory is scalable with number of processors. Increase the number of processors and the size of memory increases proportionately.
  - Each processor can rapidly access its own memory without interference and without the overhead.
  - Cost effectiveness: can use commodity, off-the-shelf processors and networking.
- Disadvantages
  - The programmer is responsible for many of the details associated with data communication between processors.
  - It may be difficult to map existing data structures, based on global memory, to this memory organization. Also, Non-uniform memory access (NUMA) times.

Hybrid Distributed-Shared Memory



Distributed shared memory (DSM) is a form of memory architecture where physically separated memories can be addressed as a single shared address space—i.e., the same physical address on two processors refers to the same location in memory.