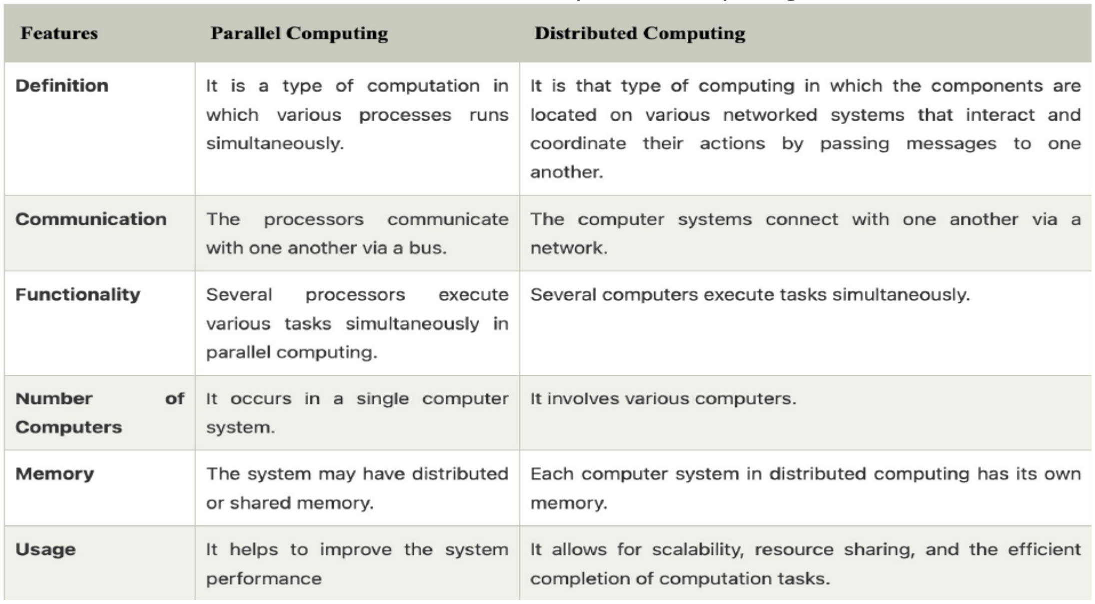
DPC NOTES

A. Whenever the socket is created, the server specifies the program—true or Fales. Justify.
True.
When a socket is created on the server, it is initiated by a server program that specifies the socket's parameters, including the domain address (IP address) and port number. This allows the program to establish communication channels for incoming connections. Thus, the statement is correct, as the server program indeed specifies both the socket and the domain address when it is created.

B. What are the limitations of distributed computing?

- High complexity in management and coordination.
- Difficulties in ensuring data consistency.
- Security vulnerabilities due to multiple networked systems.
- Increased latency in communication.

C. What is the difference between distributed and parallel computing?



Features	Parallel Computing	Distributed Computing
Definition	It is a type of computation in which various processes runs simultaneously.	It is that type of computing in which the components are located on various networked systems that interact and coordinate their actions by passing messages to one another.
Communication	The processors communicate with one another via a bus.	The computer systems connect with one another via a network.
Functionality	Several processors execute various tasks simultaneously in parallel computing.	Several computers execute tasks simultaneously.
Number of Computers	It occurs in a single computer system.	It involves various computers.
Memory	The system may have distributed or shared memory.	Each computer system in distributed computing has its own memory.
Usage	It helps to improve the system performance	It allows for scalability, resource sharing, and the efficient completion of computation tasks.

D. Amazon Web Services is an example of IAAS. Justify.
AWS is an example of **Infrastructure as a Service (IAAS)** because it provides basic computing infrastructure resources such as virtual machines, storage, and networking over the cloud. Users can rent and scale these resources on-demand without managing the underlying physical hardware. AWS offers services like **EC2 (Elastic Compute Cloud)** for virtual servers, **S3 (Simple Storage Service)** for storage, and **VPC (Virtual Private Cloud)** for networking. This allows businesses to avoid investing in physical infrastructure, paying instead for only what they use.

E. What are three widely used models of communications?

- **Remote Procedure Call (RPC),**
- **Message-Oriented Middleware (MOM) Communication,**
- **Streaming Oriented Communication**

F. What are the uses of Google Colab?
Google Colab (Colaboratory) is a free, cloud-based platform provided by Google that allows users to write, execute, and share Python code through Jupyter notebooks.

- **GPU and TPU Support**: Google Colab provides access to GPUs and TPUs, enabling efficient parallel processing for tasks like deep learning and matrix operations.
- **Multi-core Parallelism**: Users can leverage multiple CPU cores for parallel execution of algorithms using libraries like **Multiprocessing**.
- **Cloud-based Distributed Computing**: Colab allows users to connect multiple virtual machines, enabling distributed task execution across cloud resources.
- **Parallel Libraries**: Colab supports parallel and distributed libraries such as **Dask**, **Ray**, and **TensorFlow**, facilitating scalable parallel programming.
- **Integration with Google Cloud**: Colab integrates with **Google Cloud Storage** and services like Kubernetes, aiding in distributed data processing and scalable computing.
- **Collaborative Execution**: Multiple users can simultaneously execute code, supporting distributed system experiments and collaborative development.
- **Support for MPI**: Colab can run distributed applications using **MPI**, enabling inter-machine communication for distributed computing tasks.

G. Can GPU replace CPU? Justify.

No, GPUs cannot completely replace CPUs.
While GPUs (Graphics Processing Units) excel at parallel processing and are highly efficient for tasks such as deep learning, graphics rendering, and scientific simulations, they are not designed to handle general-purpose tasks, which are the domain of CPUs (Central Processing Units).
Key differences include:

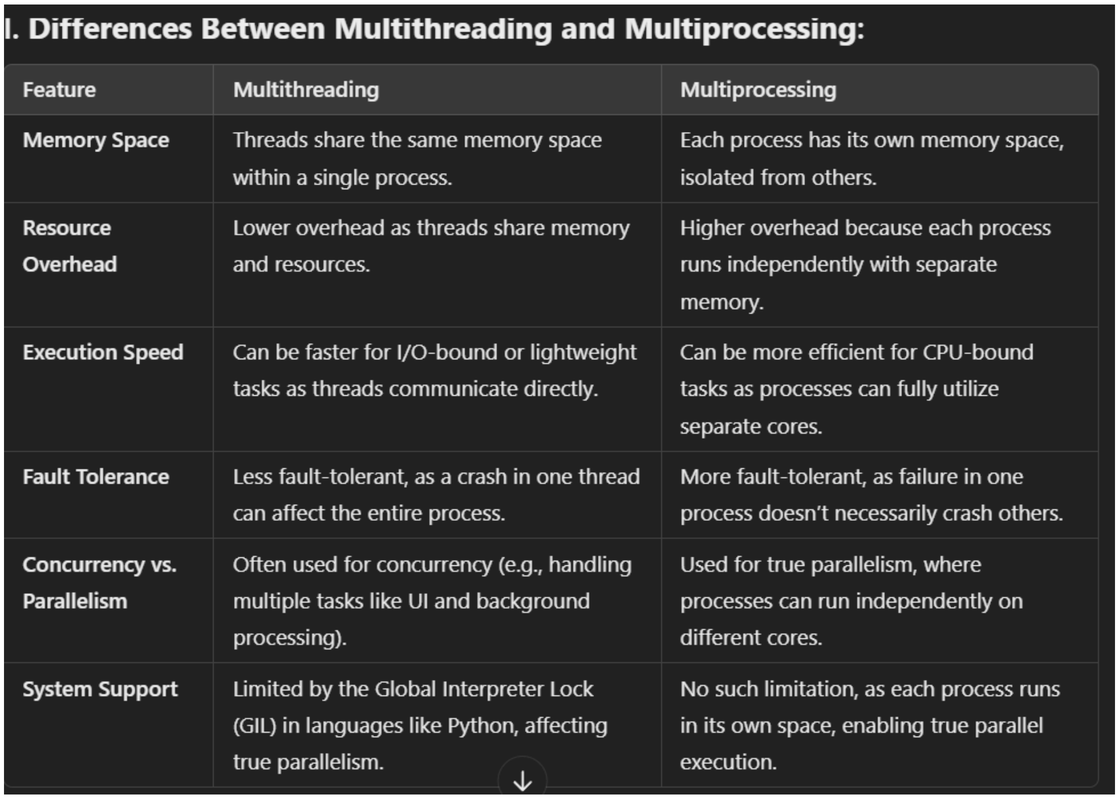
- **Task Specialization**: CPUs are optimized for general-purpose tasks (e.g., system management, running applications), while GPUs are designed for tasks involving massive parallelism (e.g., matrix operations, image processing).
- **Core Structure**: CPUs have a few powerful cores optimized for single-threaded performance, whereas GPUs have thousands of smaller, less powerful cores optimized for parallel tasks.
- **Latency and Control**: CPUs are better at handling tasks requiring low-latency responses and complex branching logic, such as running operating systems.
- **Memory Management**: CPUs manage the system's memory and coordinate with other system resources, which GPUs cannot do independently.

Thus, GPUs complement CPUs in tasks requiring parallelism but are not replacements for them in general computing tasks.

H. What is the concept behind virtualization?
Virtualization involves creating virtual instances of resources (like hardware, storage, or networking) that allow multiple operating systems or applications to run on a single physical machine.
Other properties or features of virtualization includes operating in isolation letting parallel tasks to execute in isolated environment. Virtualization makes it easier to scale distributed systems by dynamically provisioning instances based on workload. By running multiple VMs on the same hardware, virtualization reduces costs in parallel and distributed environments, optimizing

hardware usage. cloud platforms, use virtualization to distribute workloads across multiple virtualized servers for efficient parallel processing.

I. Multithreading and Multiprocessing are both the same or different.



I. Differences Between Multithreading and Multiprocessing:

Feature	Multithreading	Multiprocessing
Memory Space	Threads share the same memory space within a single process.	Each process has its own memory space, isolated from others.
Resource Overhead	Lower overhead as threads share memory and resources.	Higher overhead because each process runs independently with separate memory.
Execution Speed	Can be faster for I/O-bound or lightweight tasks as threads communicate directly.	Can be more efficient for CPU-bound tasks as processes can fully utilize separate cores.
Fault Tolerance	Less fault-tolerant, as a crash in one thread can affect the entire process.	More fault-tolerant, as failure in one process doesn't necessarily crash others.
Concurrency vs. Parallelism	Often used for concurrency (e.g., handling multiple tasks like UI and background processing).	Used for true parallelism, where processes can run independently on different cores.
System Support	Limited by the Global Interpreter Lock (GIL) in languages like Python, affecting true parallelism.	No such limitation, as each process runs in its own space, enabling true parallel execution.

J. What do you mean by speed up in the context of parallel implementation of a serial program?

In the context of parallel computing, **speedup** refers to the performance gain achieved when a task that was originally executed serially is executed in parallel. The formula for speedup is:

Speedup=Time for serial execution\Time for parallel execution

- A speedup greater than 1 indicates that the parallel version is faster.
- **Amdahl's Law** governs the speedup limit, stating that the speedup is limited by the portion of the task that cannot be parallelized. If 90% of a task can be parallelized and 10% is serial, no matter how many processors are added, the speedup is limited by the serial portion.

Q. Explain the application of parallel computing. What are the limitations of parallel computing? Give examples of instruction level and task level parallelism.

Applications of Parallel Computing:
•Databases and Data mining.
•Real-time simulation of systems.

•Science and Engineering.
•Advanced graphics, augmented reality, and virtual reality.
Limitations of Parallel Computing:
•It addresses such as communication and synchronization between multiple sub-tasks and processes which is difficult to achieve.
•The algorithms must be managed in such a way that they can be handled in a parallel mechanism.
•The algorithms or programs must have low coupling and high cohesion. But it's difficult to create such programs.
•More technically skilled and expert programmers can code a parallelism-based program well.

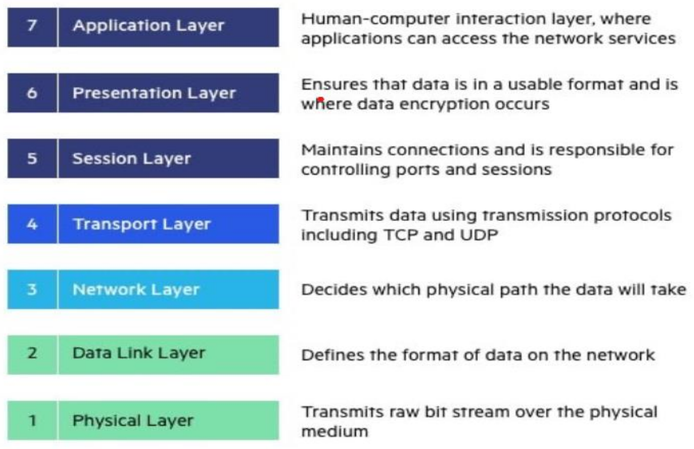
Instruction-level parallelism —
A processor can only address less than one instruction for each clock cycle phase. These instructions can be re-ordered and grouped which are later on executed concurrently without affecting the result of the program. This is called instruction-level parallelism.

- **Pipelining**: Modern CPUs use pipelining to execute multiple instructions simultaneously by breaking down instructions into stages (fetch, decode, execute, etc.) and executing different stages of different instructions in parallel.
- **Superscalar Processors**: CPUs can execute multiple instructions in the same clock cycle by dispatching instructions to different functional units (e.g., adding one number while multiplying another).

Task Parallelism -
Task parallelism employs the decomposition of a task into subtasks and then allocating each of the subtasks for execution. The processors perform the execution of sub-tasks concurrently.

- **Web Server**: A web server handling multiple client requests concurrently, where each request is treated as a separate task and handled by different threads or processes.
- **MapReduce**: In distributed systems, a MapReduce job splits a large data processing task into smaller tasks that are processed in parallel by multiple nodes, then aggregated in the reduce step.

Q. Describe the working of the OSI model with different layers. Describe the difference between OSI vs. TCP/IP Model.


7 Application Layer
Human-computer interaction layer, where applications can access the network services
6 Presentation Layer
Ensures that data is in a usable format and is where data encryption occurs
5 Session Layer
Maintains connections and is responsible for controlling ports and sessions
4 Transport Layer
Transmits data using transmission protocols including TCP and UDP
3 Network Layer
Decides which physical path the data will take
2 Data Link Layer
Defines the format of data on the network
1 Physical Layer
Transmits raw bit stream over the physical medium

Feature	**OSI Model**	**TCP/IP Model**
Number of Layers	7 layers: Application, Presentation, Session, Transport, Network, Data Link, Physical.	4 layers: Application, Transport, Internet, Network Access
Usage of layers	Simple applications do not use all the layers	Most applications use all the layers
Layer Functionality	Each layer has a distinct and well-defined function, leading to more granularity.	Combines some OSI layers (e.g., Presentation and Session merged into Application), making it less granular.
Protocol Dependency	Protocol-independent; it is a reference model and doesn't specify protocols at each layer.	Protocol-dependent; defines specific protocols such as TCP, IP, HTTP, FTP.
Usage and Implementation	Used as a reference model for understanding and designing networks, but not widely used in practice.	TCP/IP is widely used and forms the basis of the internet, making it the practical model for real-world networking.

Describe the working IPv4. What are the steps involved in Client-Server Communication?

IPv4 (Internet Protocol version 4) is a widely used protocol for identifying devices on a network through an addressing system. Each device is assigned a unique 32-bit IP address broken into 4 octets of 8bit fields and have a decimal notation (0-255).

For networks of different size,

the first one (for large networks) to three (for small networks) octets can be used to identify the network, while the rest of the octets can be used to identify the node on the network.

There are some procedures that we have to follow to establish client-server communication. These are as follows.

1. Socket: With the help of a socket, we can create a new communication.

2. Bind: With the help of this we can attach the local address with the socket.

3. Listen: With this help; we can accept the connection.

4. Accept: With this help; we can lock the incoming connection until the request arrives.

5. Connect: With this help; we can attempt to establish the connection.

6. Send: With the help of this; we can send the data over the network.

7. Receive: With this help; we can receive the data over the network.

8. Close: With the help of this, we can release the connection from the network.

Describe Uniform Memory Access (UMA) and Non-Uniform Memory Access (NUMA) with their advantages and disadvantages.

Uniform Memory Access (UMA)

- Most commonly represented today by Symmetric Multiprocessor (SMP) machine
- Identical processors
- Equal access and access times to memory
- Sometimes called CC-UMA - Cache Coherent UMA. Cache coherent means if one processor updates a location in shared memory, all the other processors know about the update. Hardware level.

Non-Uniform Memory Access (NUMA)

- Often made by physically linking two or more SMPs
- One SMP can directly access memory of another SMP
- Not all processors have equal access time to all memories
- Memory access across link is slower
- If cache coherency is maintained, then may also be called CC-NUMA - Cache Coherent NUMA

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 memo and CPUs. Adding more CPUs can geometrically increase 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 ensure "correct" access of global memory.

Q. Why is a GPU suitable for image processing? Give justification with an example.

- Shared Memory: Almost every modem GPU has shared memory', making it way better and exponentially faster than the CPU's cache. It works best with algorithms that have a high degree of locality.
- Managing Load: In contrast to a CPU, a GPU can considerably reduce the load on a subsystem by modifying the number of registers.
- Speed: The parallel processing aspect of GPU makes it much faster than a CPU because of the better memory' and processing power bandwidth. GPUs are almost 100X quicker in processing than a CPU.
- Embedded Applications: GPUs are a better alternative to embedded applications like ASICs and FPGAs as they offer more flexibility. Parallel execution of various tasks: The different hardware modules of a GPU allow entirely diverse tasks to be executed simultaneously—for instance, tensor kernels for neural networks.

Example:

Convolution in Image Filtering:
- One common image processing task is applying a convolution filter, such as a Gaussian blur, to an image. The convolution operation involves multiplying a kernel (filter) with the corresponding pixel values in the image and summing the results for each pixel.

Distributed Systems

Client-Server

In this client sends requests to the server then, the server responds to the client requests.

Clients contact the server for data, then format it and display it to the end-user. The end-user can also make changes from the client-side and commit them back to the server to make it permanent.

There are three main methods: **Sockets, remote procedure calls, Pipes.**

Most web applications are **three tiers**: **Presentation, Business, Data Tier**

Peer to peer network: **Torrent**

Components: Devices/systems, Network, Resource Management

Characteristics: Multiple devices/system; peer to peer architecture, shared resources, horizontal scaling

Finance and Commerce: Online Banking, E-Commerce websites.

Information Society: Search Engines, Wikipedia, Social Networking, Cloud Computing.

Cloud Technologies: AWS, Salesforce, Microsoft Azure, SAP.

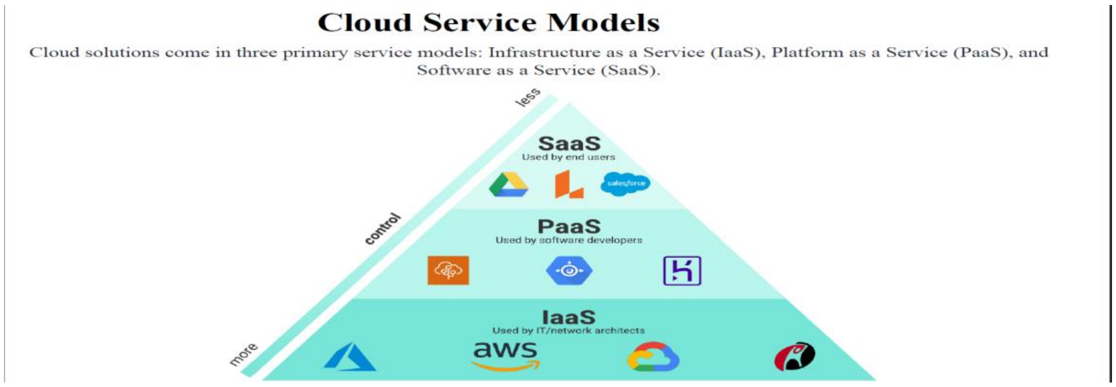
Entertainment: Online Gaming, Music, YouTube.

Healthcare: Online patient records, Health Informatics. Education: E-learning.

Transport and logistics: GPS, Google Maps. Environment Management: Sensor technologies.

Advantages: scalability, readability, flexibility

Disadvantages: Complexity, security, Performance.


Cloud Service Models
Cloud solutions come in three primary service models: Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS).
less
control
more
SaaS
Used by end users
salesforce
PaaS
Used by software developers
IaaS
Used by IT/network architects
aws

Cloud Computing benefits:

Less expense, security, flexibility, collaboration, automatic updates

There are two types of computations: **parallel computing** and **distributed computing**.

Parallel computing allows several processors to accomplish their tasks at the same time.

In contrast, **distributed computing splits a single task among numerous systems to achieve a common goal.**

Parallelism Types:

Bit level, Instruction Level, Task level, Data Level.

Inter-process communication is at the **heart** of **all distributed systems**.

All communication in **distributed system** is based on **message passing**.

A protocol is a **set of rules** and conventions that describe how **information** is to be **exchanged** between **two entities**.

A distinction is made between **two** general **types of protocols**.

connection-oriented protocols:

Before exchanging data, the sender and receiver first explicitly establish a connection, and possibly negotiate the protocol they will use. When they are done, they must release (terminate) the connection.

Example: Telephone

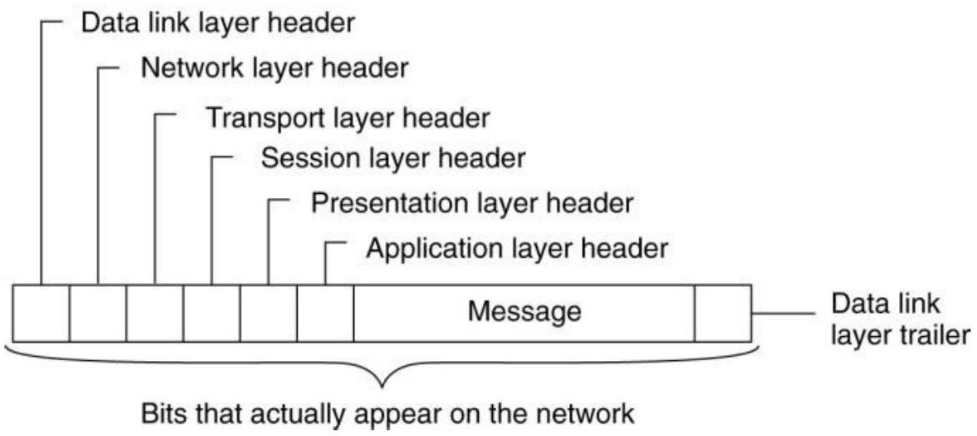
Connectionless protocols:

no setup in advance is needed. The sender just transmits the first message when it is ready.

Example: Dropping a letter in a mailbox

computers, **both connection-oriented and connectionless communication are common**.

Connection: TCP Connection less: UDP (User Datagram Protocol)



Socket Programming

A socket is a type of medium that provides a connection between two devices.

Helps different applications to attach to the local network with different ports.

Every time when the socket is created, the server specifies the program, and that program specifies the socket and the domain address.

Socket Programming: method to connect two nodes over a network to establish a means of communication between those two nodes (device with net connection).

A socket is the endpoint used for connecting to a node.

TCP is used for services with a large data capacity, and a persistent connection

UDP is more commonly used for quick lookups, and single use query-reply actions.

Berkley Socket

It is an abstraction through which an application may send and receive data

Provide generic access to inter-process communication services

e.g., IPX/SPX, AppleTalk, TCP/IP

Uniquely identified by

- internet addresses
- an end-to-end protocol (e.g., TCP or UDP)
- a port number

Two types of (TCP/IP) sockets:

Stream sockets (e.g., uses TCP): provide reliable byte-stream service

Datagram sockets (e.g., uses UDP): provide best-effort datagram service (max 65.5K bytes).

Remote Procedure Call (RPC)

Communication technology that is used by one program to **make a request to another program** for **utilizing its service on a network without even knowing the network's details**. A **function call** or a **subroutine call** are **other terms** for a **procedure call**.

Multiple RPCs can be executed concurrently by utilizing lightweight processes on threads that share the same address space.

Remote Procedure Call program as often as possible utilizes the Interface Definition Language (IDL), a determination language for describing a computer program component's (API).

Types of RPC:

Callback RPC: In a Callback RPC, a 2P (Peer to Peer) paradigm opts between participating processes. In this way, a process provides both client and server functions which are quite helpful. Callback RPC's features include:

•The problems handled remotely
•It provides a server for clients to use.
•Due to the callback mechanism, the client process is delayed.
•Deadlocks need to be managed in callbacks.

RPC for Broadcast: A client's request that is broadcast all through the network and handled by all servers that possess the method for handling that request is known as a broadcast RPC.

Broadcast RPC's features include:

•option of selecting whether or not the client's request message ought to be broadcast.
•option of declaring broadcast ports
•helps in diminishing physical network load.

Batch-mode RPC: Batch-mode RPC enables the client to line and separate RP inquiries in a transmission buffer before sending them to the server in a single batch over the network.
Batch-mode RPC's features include:
•diminishes the overhead of requesting the server by sending them all at once using the network.
•It is used for applications that require low call rates.
•It necessitates the use of a reliable transmission protocol.

Local Procedure Call:
- Same address space
- Less prone to failures as no such comm network required.
- Takes comparatively less time

Remote Procedure Call:
- Disjoint/different address space
- More prone to failures
- Takes comparatively more time

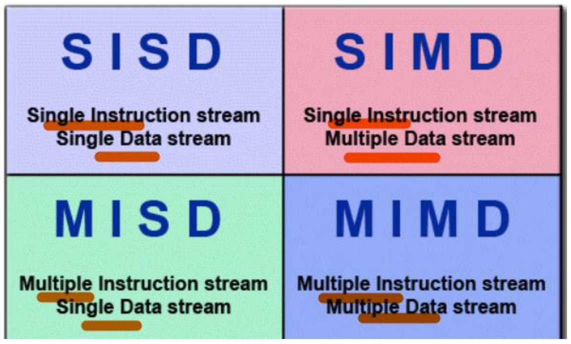
Parallel Computing

In the simplest sense, parallel computing is the simultaneous use of multiple compute resource to solve a computational problem
- A problem is broken into discrete parts that can be solved concurrently
- Each part is further broken down to a series of instructions.
- Instructions from each part execute simultaneously on different processors.

- An overall control/coordination mechanism is employed.

Flynn's Classical Taxonomy

There are a number of different ways to classify parallel computers.
One of the more widely used classifications, in use since 1966, is called Flynn's Taxonomy
Distinguishes according to the two independent dimensions of **Instruction Stream** and **Data Stream**.
Each dimension can have only one of two possible states: **Single or Multiple**.
The matrix below defines the **4 possible classifications** according to Flynn:


SISD
Single Instruction stream
Single Data stream
SIMD
Single Instruction stream
Multiple Data stream
MISD
Multiple Instruction stream
Single Data stream
MIMD
Multiple Instruction stream
Multiple Data stream

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.

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
Application: Image processing (GPU follows SIMD)
Two varieties: Processor Arrays and Vector Pipeline

Multiple Instruction, Single Data (MISD)
A type of parallel computer
Multiple Instruction: Each processing unit operates on the data independently via separate instruction streams.
Single Data: A single data stream is fed into multiple processing units.

Multiple Instruction, [Multiple Data (MIMD)

•A type of parallel computer

•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 non-deterministic

Most computers are of this type.

Parallel Programming Models

- **Shared Memory (without threads)**
- **Threads**
- **Distributed Memory (Message Passing)**
- **Data Parallel**
- **Hybrid**
- **Single Program Multiple Data (SPMD)**
- **Multiple Program Multiple Data (MPMD)**

Shared Memory Model (without threads)

In this programming model, processes/tasks share a common address space, which they read and write to asynchronously.

Shared memory classified as UMA and NUMA.

Threads Model

In the threads model of parallel programming, a single "heavy weight" process can have multiple "light

Weight", concurrent execution paths.

Ex: POSIX Threads and OpenMP

Distributed Memory / Message Passing

Set of tasks that use their own local memory" during computation

Multiple tasks can reside on the same physical machine and/or across an arbitrary number of machines.

Tasks exchange data through communications by sending and receiving messages.

Data Parallel Model

May also be referred to as the Partitioned Global Address Space (PGAS) model.

On shared memory architectures, all tasks may have access to the data structure through global memory.

On distributed memory architectures, the global data structure can be split up logically and/or physically across tasks.

Single Program Multiple Data (SPMD)

SPMD is actually a "high level" programming model that can be built upon any combination of the previously mentioned parallel programming models.

SINGLE PROGRAM: All tasks execute their copy of the same program simultaneously. This program can be threads, message passing, data parallel or hybrid.

MULTIPLE DATA: All tasks may use different data

SPMD using message passing

Multiple Program Multiple Data (MPMD)

a "high level" programming model that can be built upon any combination of the previously mentioned parallel programming models.

MULTIPLE PROGRAM: Tasks may execute different programs simultaneously. The programs can be threads, message passing, data parallel or hybrid.

MULTIPLE DATA: All tasks may use different data.

CPU	GPU
Have large memory capacity	Relatively low memory capacity
Low memory bandwidth	High bandwidth memory
Latency optimized via large caches	Latency tolerant via parallelism
Very fast clock speed	Have more compute resource
Small no of thread run quickly	Low per thread performance
Low performance/watt	High performance/watt
	High throughput

A CPU works together with a GPU to increase the throughput of data and the number of concurrent calculations within an application.

A CPU can never be fully replaced by a GPU: a GPU complements CPU architecture by allowing repetitive calculations within an application to be run in parallel while the main program continues to run on the CPU.

Parallel Program Design

parallelizing compiler generally works in two different ways:

Fully Automatic

- The compiler analyzes the source code and identifies opportunities for parallelism.
- The analysis includes identifying inhibitors to parallelism and possibly a cost weighting on whether or not the parallelism would actually improve performance.
- Loops (do, for) are the most frequent target for automatic parallelization.

Programmer Directed

- Using "compiler directives" or possibly compiler flags, the programmer explicitly tells the compiler how to parallelize the code.
- The most common compiler generated parallelization is done using on-node shared memory and threads (such as OpenMP).

JAVA

Java is a high-level, general-purpose, object-oriented, and secure programming language.

Features of Java

- **Simple:** Java is a simple language because its syntax is simple, clean, and easy to understand. For example, pointer and operator overloading are not used in Java.
- **Object-Oriented**: In Java, everything is in the form of the object. A program must have at least one class and object.
- **Robust**: Java makes an effort to check error at run time and compile time. It uses a strong memory management system called garbage collector. Exception handling and garbage collection features make it strong.
- **Secure**: Java is a secure programming language because it has no explicit pointer and programs runs in the virtual machine. Java contains a security manager that defines the access of Java classes.
- **Platform-Independent**: Java provides a guarantee that code writes once and run anywhere. This byte code is platform independent and can be run on any machine.
- **Portable**: Java Byte code can be carried to any platform.
- **High Performance**: Java is an interpreted language.
- **Distributed**: Java also has networking facilities. It is designed for the distributed environment of the internet because it supports TCP/IP protocol. EJB and RMI are used to create a distributed system.
- **Multi-threaded**: Java also supports multi-threading. It means to handle more than one job a time.

Types of Variables

Local Variable

- A variable declared inside the body of the method is called a local variable.
- You can use this variable only within that method.
- A local variable cannot be defined with a "static" keyword.

Instance Variable

- A variable declared inside the class but outside the body of the method, is called an instance variable.
- It is not declared as static.
- It is called an instance variable because its value is instance-specific and is not shared among instances.

Static Variable

- A variable that is declared as static is called a static variable. It cannot be local.

- You can create a single copy of the static variable and share it among all the instances of the class. Memory allocation for static variables happens only once when the class is loaded in the memory.

Multithreading in Java

Multithreading in Java is a process of executing multiple threads simultaneously.

A **thread** is a lightweight sub-process, the smallest unit of processing. Multiprocessing and multithreading, both are used to achieve multitasking.

Advantages:

It doesn't block the user because threads are independent and you can perform multiple operations at the same time.

You can perform many operations together, so it saves time.

Threads are independent, so it doesn't affect other threads if an exception occurs in a single thread.

Java Multithreading is mostly used in games, animation, etc.

Multitasking

Multitasking is a process of executing multiple tasks simultaneously. We use multitasking to utilize

the CPU. Multitasking can be achieved in two ways:

Process-based Multitasking (Multiprocessing)

Thread-based Multitasking (Multithreading)

1) **Process-based Multitasking (Multiprocessing)**

- Each process has an address in memory. In other words, each process allocates a separate memory area.
- A process is heavyweight.
- Cost of communication between the process is high.
- Switching from one process to another requires some time for saving and loading registers, memory maps, updating lists, etc.

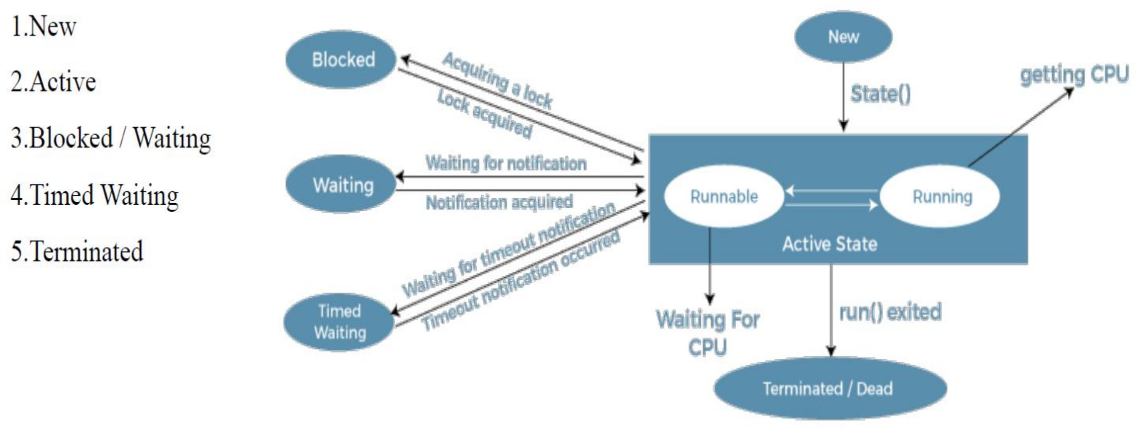
2) **Thread-based Multitasking (Multithreading)**

- Threads share the same address space.
- A thread is lightweight.
- Cost of communication between the thread is low.

Thread

A thread is a lightweight subprocess, the smallest unit of processing. It is a separate path of execution. Threads are independent. If there occurs exception in one thread, it doesn't affect other threads. It uses a shared memory area.

Java provides Thread class to achieve thread programming. Thread class provides constructors and methods to create and perform operations on a thread. Thread class extends Object class and implements Runnable interface.



Life Cycle of a Thread

Java RMI: Remote Method Invocation

• It is a mechanism that allows an object residing in one system (JVM) to access invoke an object running on another JVM.
• RMI is used to build distributed applications; it provides remote communication between Java programs.

Architecture of an RMI Application

- In an RMI application, we write two programs, a server program (resides on the server) and a client program (resides on the client).
- Inside the server program, a remote object is created and reference of that object is made available for the client (using the registry).
- The client program requests the remote objects on the server and tries to invoke its methods.

Working of an RMI Application

- When the client makes a call to the remote object, it is received by the stub which eventually passes this request to the RRL.
- When the client-side RRL receives the request, it invokes a method called invoke () of the object remoteRef. It passes the request to the RRL on the server side.

- The RRL on the server side passes the request to the Skeleton (proxy on the server) which finally invokes the required object on the server.
- The result is passed all the way back to the client.

Marshalling

Whenever a client invokes a method that accepts parameters on a remote object, the parameters are bundled into a message before being sent over the network. In case of primitive type, the parameters are put together and a header is attached to it. In case the parameters are objects, then they are serialized. This process is known as marshalling.

Unmarshalling

At the server side, the packed parameters are unbundled and then the required method is invoked. This process is known as unmarshalling.

RMI Registry

RMI registry is a namespace on which all server objects are placed. Each time the server creates an object, it registers this object with the RMIregistry (using bind () or rebind () methods). These are registered using a unique name known as bind name.