Parallel Computing –
It is the use of multiple processing elements simultaneously for solving any problem. Problems are broken down into instructions and are solved concurrently as each resource which has been applied to work is working at the same time.
Advantages of Parallel Computing over Serial Computing are as follows:

1. It saves time and money as many resources working together will reduce the time and cut potential costs.
2. It can be impractical to solve larger problems on Serial Computing.
3. It can take advantage of non-local resources when the local resources are finite.
4. Serial Computing 'wastes' the potential computing power, thus Parallel Computing makes better work of hardware.

Types of Parallelism:

1. **Bit-level parallelism:** It is the form of parallel computing which is based on the increasing processor's size. It reduces the number of instructions that the system must execute in order to perform a task on large-sized data.
   Example: Consider a scenario where an 8-bit processor must compute the sum of two 16-bit integers. It must first sum up the 8 lower-order bits, then add the 8 higher-order bits, thus requiring two instructions to perform the operation. A 16-bit processor can perform the operation with just one instruction.
2. **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.
3. **Task Parallelism:** Task parallelism employs the decomposition of a task into subtasks and then allocating each of the subtasks for execution. The processors perform execution of sub tasks concurrently.

Why parallel computing?

- The whole real world runs in dynamic nature i.e. many things happen at a certain time but at different places concurrently. This data is extensively huge to manage.
- Real world data needs more dynamic simulation and modeling, and for achieving the same, parallel computing is the key.
- Parallel computing provides concurrency and saves time and money.
- Complex, large datasets, and their management can be organized only and only using parallel computing's approach.
- Ensures the effective utilization of the resources. The hardware is guaranteed to be used effectively whereas in serial computation only some part of hardware was used and the rest rendered idle.
- Also, it is impractical to implement real-time systems using serial computing.

Applications of Parallel Computing:

- Data bases and Data mining.
- Real time simulation of systems.
- Science and Engineering.
- Advanced graphics, augmented reality and virtual reality.

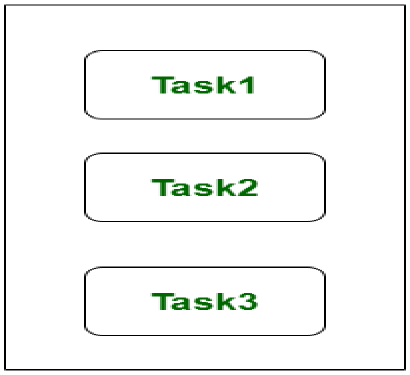
Limitations of Parallel Computing:

- 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 the parallel mechanism.
- The algorithms or program 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.

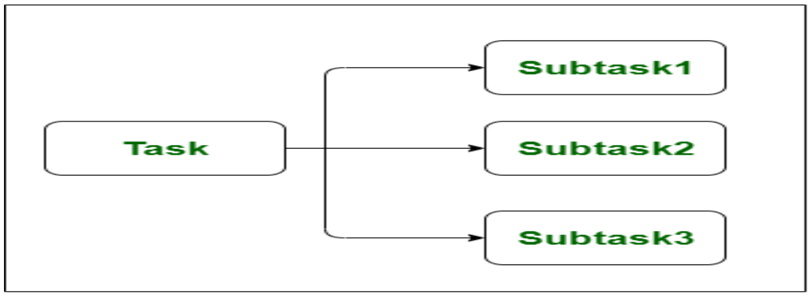
Future of Parallel Computing: The computational graph has undergone a great transition from serial computing to parallel computing. Tech giant such as Intel has already taken a step towards parallel computing by employing multicore processors. Parallel computation will revolutionize the way computers work in the future, for the better good. With all the world connecting to each other even more than before, Parallel Computing does a better role in helping us stay that way. With faster networks, distributed systems, and multi-processor computers, it becomes even more necessary.
Difference between Concurrency and Parallelism

Concurrency:
Concurrency relates to an application that is making progress more than one task at the same time. Concurrency is a approach that is used for decreasing the response time of the system by using the single processing unit. Concurrency is that the illusion of parallelism, however in actual the chunks of a task aren’t parallelly processed, but inside the application, there are more than one task is being processed at a time. It doesn’t fully end one task before it begins ensuing. Concurrency is achieved through the interleaving operation of processes on the central processing unit(CPU) or in other words by the context switching. that’s rationale it’s like parallel processing. It increases the amount of work finished at a time.



In the above figure, we can see that **there is multiple tasks making progress at the same time.** This figure shows the concurrency because concurrency is the technique that deals with the lot of things at a time.
Parallelism:
Parallelism related to an application in which the tasks are divided into smaller sub-tasks that are processed simultaneously or parallel. It is used for increasing the throughput and computational speed of the system by using the multiple processors. It is the technique that do lot of things simultaneously.
Parallelism leads to overlapping of central processing unit and input-output tasks in one process with the central processing unit and input-output tasks of another process. Whereas in

concurrency the speed is increased by overlapping the input-output activities of one process with CPU process of another process.



In the above figure, we can see that **the tasks are divided into smaller sub-tasks that are processing simultaneously or parallel.** This figure shows the parallelism because parallelism is the technique that do lot of things simultaneously.

Difference between Concurrency and Parallelism:-

S.NO	CONCURRENCY	PARALLELISM
1.	Concurrency is the task of running and managing the multiple computations at the same time.	While parallelism is the task of running multiple computations simultaneously.
2.	Concurrency is achieved through the interleaving operation of processes on the central processing unit(CPU) or in other words by the context switching.	While it is achieved by through multiple central processing units(CPUs).
3.	Concurrency can be done by using a single processing unit.	While this can't be done by using a single processing unit. it needs multiple processing units.
4.	Concurrency increases the amount of work finished at a time.	While it improves the throughput and computational speed of the system.
5.	Concurrency deals lot of things simultaneously.	While it do lot of things simultaneously.

6.	Concurrency is the non-deterministic control flow approach.	While it is deterministic control flow approach.
7.	In concurrency debugging is very hard.	While in this debugging is also hard but simple than concurrency.

Cache memory

Cache memory is sometimes called CPU (central processing unit) memory because it is typically integrated directly into the CPU chip or placed on a separate chip that has a separate bus interconnect with the CPU. Therefore, it is more accessible to the processor, and able to increase efficiency, because it's physically close to the processor.

In order to be close to the processor, cache memory needs to be much smaller than main memory. Consequently, it has less storage space. It is also more expensive than main memory, as it is a more complex chip that yields higher performance.

What it sacrifices in size and price, it makes up for in speed. Cache memory operates between 10 to 100 times faster than RAM, requiring only a few nanoseconds to respond to a CPU request.

The name of the actual hardware that is used for cache memory is high-speed static random access memory (SRAM). The name of the hardware that is used in a computer's main memory is dynamic random access memory (DRAM).

Cache memory is not to be confused with the broader term cache. Caches are temporary stores of data that can exist in both hardware and software. Cache memory refers to the specific hardware component that allows computers to create caches at various levels of the network.

Types of cache memory

Cache memory is fast and expensive. Traditionally, it is categorized as "levels" that describe its closeness and accessibility to the microprocessor. There are three general cache levels:

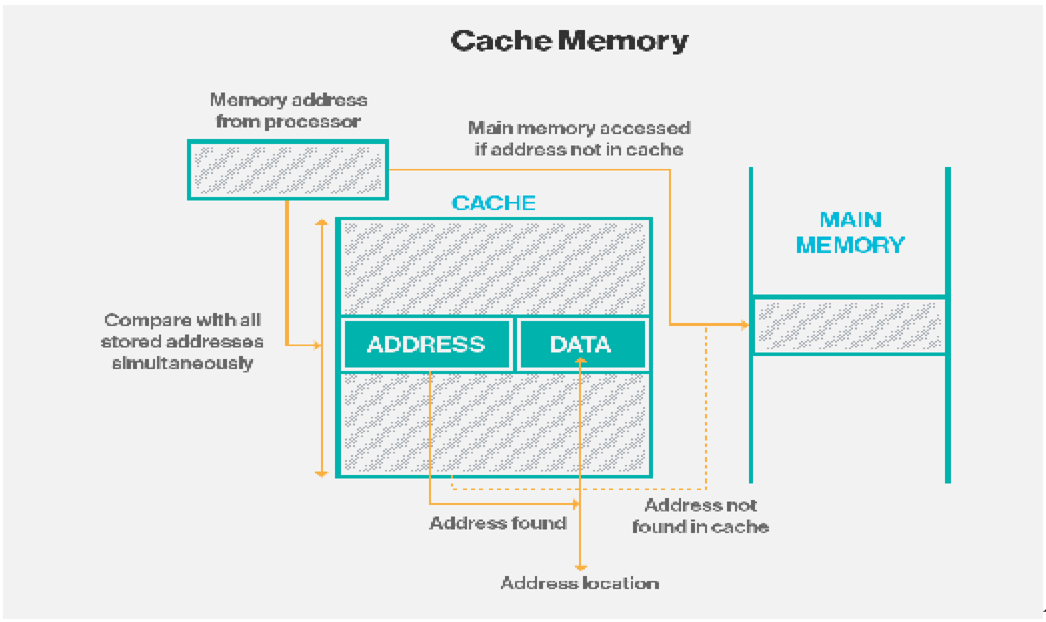
L1 cache, or primary cache, is extremely fast but relatively small, and is usually embedded in the processor chip as CPU cache.

L2 cache, or secondary cache, is often more capacious than L1. L2 cache may be embedded on the CPU, or it can be on a separate chip or coprocessor and have a high-speed alternative system bus connecting the cache and CPU. That way it doesn't get slowed by traffic on the main system bus.

Level 3 (L3) cache is specialized memory developed to improve the performance of L1 and L2. L1 or L2 can be significantly faster than L3, though L3 is usually double the speed of DRAM. With multicore processors, each core can have dedicated L1 and L2 cache, but they can share an L3 cache. If an L3 cache references an instruction, it is usually elevated to a higher level of cache.

In the past, L1, L2 and L3 caches have been created using combined processor and motherboard components. Recently, the trend has been toward consolidating all three levels of memory caching on the CPU itself. That's why the primary means for increasing cache size has begun to shift from the acquisition of a specific motherboard with different chipsets and bus architectures to buying a CPU with the right amount of integrated L1, L2 and L3 cache.

Contrary to popular belief, implementing flash or more dynamic RAM (DRAM) on a system won't increase cache memory. This can be confusing since the terms *memory caching* (hard disk buffering) and *cache memory* are often used interchangeably. Memory caching, using DRAM or flash to buffer disk reads, is meant to improve storage I/O by caching data that is frequently referenced in a buffer ahead of slower magnetic disk or tape. Cache memory, on the other hand, provides read buffering for the CPU.


Cache Memory
Memory address from processor
Main memory accessed if address not in cache
CACHE
MAIN MEMORY
Compare with all stored addresses simultaneously
ADDRESS
DATA
Address found
Address not found in cache
Address location
A diagram of the architecture and data flow of a typical cache memory unit.

Cache memory mapping

Caching configurations continue to evolve, but cache memory traditionally works under three different configurations:

This CompTIA A+ video tutorial explains cache memory.

- **Direct mapped cache** has each block mapped to exactly one cache memory location. Conceptually, a direct mapped cache is like rows in a table with three columns: the cache block that contains the actual data fetched and stored, a tag with all or part of the address of the data that was fetched, and a flag bit that shows the presence in the row entry of a valid bit of data.
- **Fully associative cache mapping** is similar to direct mapping in structure but allows a memory block to be mapped to any cache location rather than to a prespecified cache memory location as is the case with direct mapping.
- **Set associative cache mapping** can be viewed as a compromise between direct mapping and fully associative mapping in which each block is mapped to a subset of cache locations. It is sometimes called *N-way set associative mapping*, which provides for a location in main memory to be cached to any of "N" locations in the L1 cache.

Data writing policies

Data can be written to memory using a variety of techniques, but the two main ones involving cache memory are:

- **Write-through.** Data is written to both the cache and main memory at the same time.
- **Write-back.** Data is only written to the cache initially. Data may then be written to main memory, but this does not need to happen and does not inhibit the interaction from taking place.

The way data is written to the cache impacts data consistency and efficiency. For example, when using write-through, more writing needs to happen, which causes latency upfront. When using write-back, operations may be more efficient, but data may not be consistent between the main and cache memories.

One way a computer determines data consistency is by examining the dirty bit in memory. The dirty bit is an extra bit included in memory blocks that indicates whether the information has been modified. If data reaches the processor's register file with an active dirty bit, it means that it is not up to date and there are more recent versions elsewhere. This scenario is more likely to happen in a write-back scenario, because the data is written to the two storage areas asynchronously.

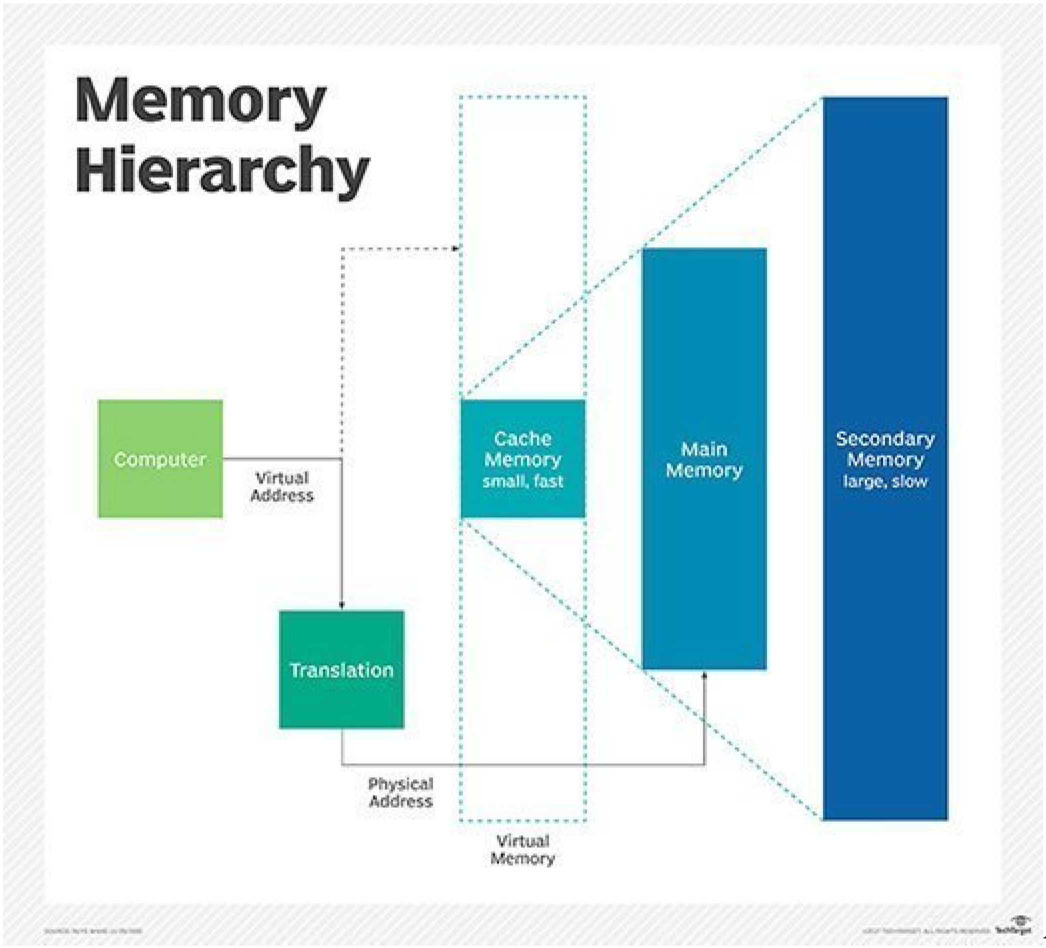
Specialization and functionality

In addition to instruction and data caches, other caches are designed to provide specialized system functions. According to some definitions, the L3 cache's shared design makes it a specialized cache. Other definitions keep the instruction cache and the data cache separate and refer to each as a specialized cache.

Translation lookaside buffers (TLBs) are also specialized memory caches whose function is to record virtual address to physical address translations.

Still other caches are not, technically speaking, memory caches at all. Disk caches, for instance, can use DRAM or flash memory to provide data caching similar to what memory caches do with CPU instructions. If data is frequently accessed from the disk, it is cached into DRAM or flash-based silicon storage technology for faster access time and response.

Specialized caches are also available for applications such as web browsers, databases, network address binding and client-side Network File System protocol support. These types of caches might be distributed across multiple networked hosts to provide greater scalability or performance to an application that uses them.



A depiction of the memory hierarchy and how it functions

Locality

The ability of cache memory to improve a computer's performance relies on the concept of locality of reference. Locality describes various situations that make a system more predictable.

Cache memory takes advantage of these situations to create a pattern of memory access that it can rely upon.

There are several types of locality. Two key ones for cache are:

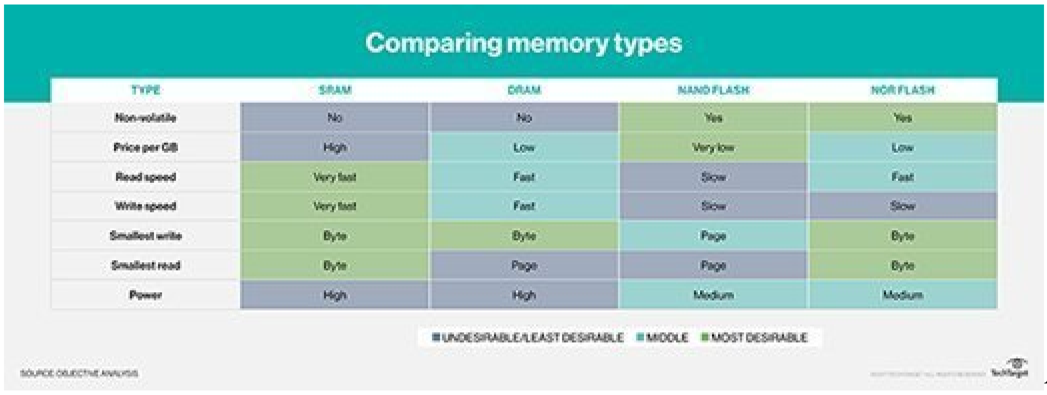
- **Temporal locality.** This is when the same resources are accessed repeatedly in a short amount of time.
- **Spatial locality.** This refers to accessing various data or resources that are near each other.

Performance

Cache memory is important because it improves the efficiency of data retrieval. It stores program instructions and data that are used repeatedly in the operation of programs or information that the CPU is likely to need next. The computer processor can access this information more quickly from the cache than from the main memory. Fast access to these instructions increases the overall speed of the program.

Aside from its main function of improving performance, cache memory is a valuable resource for *evaluating* a computer's overall performance. Users can do this by looking at cache's hit-to-miss ratio. Cache hits are instances in which the system successfully retrieves data from the cache. A cache miss is when the system looks for the data in the cache, can't find it, and looks somewhere else instead. In some cases, users can improve the hit-miss ratio by adjusting the cache memory block size -- the size of data units stored.

Improved performance and ability to monitor performance are not just about improving general convenience for the user. As technology advances and is increasingly relied upon in mission-critical scenarios, having speed and reliability becomes crucial. Even a few milliseconds of latency could potentially lead to enormous expenses, depending on the situation.



Comparing memory types

TYPE	SRAM	DRAM	NAND FLASH	NOR FLASH
Non-volatile	No	No	Yes	Yes
Price per GB	High	Low	Very low	Low
Read speed	Very fast	Fast	Slow	Fast
Write speed	Very fast	Fast	Slow	Slow
Smallest write	Byte	Byte	Page	Byte
Smallest read	Byte	Page	Page	Byte
Power	High	High	Medium	Medium

■ UNDESIRABLE/LEAST DESIRABLE ■ MIDDLE ■ MOST DESIRABLE

SOURCE: OBJECTIVE ANALYSIS

A chart comparing cache memory to other memory types

Cache vs. main memory

DRAM serves as a computer's main memory, performing calculations on data retrieved from storage. Both DRAM and cache memory are volatile memories that lose their contents when the power is turned off. DRAM is installed on the motherboard, and the CPU accesses it through a bus connection.

DRAM is usually about half as fast as L1, L2 or L3 cache memory, and much less expensive. It provides faster data access than flash storage, hard disk drives (HDD) and tape storage. It came into use in the last few decades to provide a place to store frequently accessed disk data to improve I/O performance.

DRAM must be refreshed every few milliseconds. Cache memory, which also is a type of random access memory, does not need to be refreshed. It is built directly into the CPU to give the processor the fastest possible access to memory locations and provides nanosecond speed access time to frequently referenced instructions and data. SRAM is faster than DRAM, but because it's a more complex chip, it's also more expensive to make.

An example of dynamic RAM.

Cache vs. virtual memory

A computer has a limited amount of DRAM and even less cache memory. When a large program or multiple programs are running, it's possible for memory to be fully used. To compensate for a shortage of physical memory, the computer's operating system (OS) can create virtual memory.

To do this, the OS temporarily transfers inactive data from DRAM to disk storage. This approach increases virtual address space by using active memory in DRAM and inactive memory in HDDs to form contiguous addresses that hold both an application and its data. Virtual memory lets a computer run larger programs or multiple programs simultaneously, and each program operates as though it has unlimited memory.

In order to copy virtual memory into physical memory, the OS divides memory into page files or swap files that contain a certain number of addresses. Those pages are stored on a disk and when they're needed, the OS copies them from the disk to main memory and translates the virtual memory address into a physical one. These translations are handled by a memory management unit (MMU).