High Performance Computing (HPC)

What is HPC?

HPC is technology that uses clusters of powerful processors, working in parallel, to process massive multi-dimensional datasets (big data) and solve complex problems at extremely high speeds. HPC systems typically perform at speeds more than one million times faster than the fastest commodity desktop, laptop or server systems.

For decades the HPC system paradigm was the supercomputer, a purpose-built computer that embodies millions of processors or processor cores. Supercomputers are still with us; at this writing, the fastest supercomputer is the US-based Frontier (link resides outside ibm.com), with a processing speed of 1.102 *exaflops*, or quintillion floating point operations per second (flops). But today, more and more organizations are running HPC solutions on clusters of high-speed computers servers, hosted on premises or in the cloud.

HPC workloads uncover important new insights that advance human knowledge and create significant competitive advantage. For example, HPC is used to sequence DNA, automate stock trading, and run algorithms and simulations—like those enabling self-driving automobiles—that analyze terabytes of data streaming from IoT sensors, radar and GPS systems in real time to make split-second decisions.
How does HPC work?

A standard computing system solves problems primarily using serial computing—it divides the workload into a sequence of tasks, and then executes the tasks one after the other on the same processor.

In contrast, HPC leverages

- **Massively parallel computing**. Parallel computing runs multiple tasks simultaneously on multiple computer servers or processors. Massively parallel computing is parallel computing using tens of thousands to millions of processors or processor cores.

- **Computer clusters (also called HPC clusters)**. An HPC cluster consists of multiple high-speed computer servers networked together, with a centralized scheduler that manages the parallel computing workload. The computers, called nodes, use either high-performance multi-core CPUs or, more likely today, GPUs (graphical processing units), which are well suited for rigorous mathematical calculations, machine learning models and graphics-intensive tasks. A single HPC cluster can include 100,000 or more nodes.

- **High-performance components:** All the other computing resources in an HPC cluster—networking, memory, storage and file systems—are high-speed, high-throughput and low-latency components that can keep pace with the nodes and optimize the computing power and performance of the cluster.

HPC and cloud computing

As recently as a decade ago, the high cost of HPC—which involved owning or leasing a supercomputer or building and hosting an HPC cluster in an on-premises data center—put HPC out of reach for most organizations.

Today HPC in the cloud—sometimes called HPC as a service, or HPCaaS—offers a significantly faster, more scalable and more affordable way for companies to take advantage of HPC. HPCaaS typically includes access to HPC clusters and infrastructure hosted in a cloud service provider's data center, plus ecosystem capabilities (such as AI and data analytics) and HPC expertise.

Today HPC in the cloud is driven by three converging trends:

- **Surging demand.** Organizations across all industries are becoming increasingly dependent on the real-time insights and competitive advantage that results from solving the complex problems only HPC apps can solve. For example, credit card fraud detection—something virtually all of us rely on and most of us have experienced at one time or another—relies increasingly on HPC to identify fraud faster and reduce annoying false positives, even as fraud activity expands and fraudsters' tactics change constantly.
- **Prevalence of lower-latency, higher-throughput RDMA networking.** RDMA—remote direct memory access—enables one networked computer to access another networked computer's memory without involving either computer's operating system or interrupting either computer's processing. This helps minimize latency and maximize throughput. Emerging high-performance RDMA fabrics—including Infiniband, Virtual Interface Architecture, and RDMA over converged ethernet (RoCE)—are essentially making cloud-based HPC possible.
- **Widespread public-cloud and private-cloud HPCaaS availability.** Today every leading public cloud service provider offers HPC services. And while some organizations continue to run highly regulated or sensitive HPC workloads on-premises, many are adopting or migrating to private-cloud HPC solutions offered by hardware and solution vendors.

HPC use cases

HPC applications have become synonymous with AI apps in general, and with machine learning and deep learning apps in particular; today most HPC systems are created with these workloads in mind. These HPC applications are driving continuous innovation in:

Healthcare, genomics and life sciences. The first attempt to sequence a human genome took *13 years*; today, HPC systems can do the job in less than a day. Other HPC applications in healthcare and life sciences include drug discovery and design, rapid cancer diagnosis, and molecular modeling.

Financial services. In addition to automated trading and fraud detection (noted above), HPC powers applications in Monte Carlo simulation and other risk analysis methods.

Government and defense. Two growing HPC use cases in this area are weather forcasting and climate modeling, both of which involve processing vast amounts of historical meteorological data and millions of daily changes in climate-related data points. Other government and defense applications include energy research and intelligence work.

Energy. In some cases overlapping with government and defense, energy-related HPC applications include seismic data processing, reservoir simulation and modeling, geospatial analytics, wind simulation and terrain mapping.

What is HPC Domain?

The term "HPC domain" refers to the specific field or industry that extensively utilizes High-Performance Computing (HPC) technologies and methodologies to address computational challenges and achieve their objectives. The HPC domain encompasses various sectors and disciplines where large-scale data processing, simulation, modeling, and analysis are integral to advancing research, innovation, and problem-solving.

Some common domains that heavily rely on HPC include:

1. **Scientific Research**: HPC is widely used in scientific domains such as physics, chemistry, biology, astronomy, and geophysics to conduct complex simulations, analyze large datasets, and perform data-intensive computations.

2. **Engineering and Manufacturing**: HPC is employed in engineering simulations, computational fluid dynamics (CFD), finite element analysis (FEA), structural analysis, optimization, and other computationally demanding tasks to design and test new products, improve manufacturing processes, and enhance performance.

3. **Weather and Climate Modeling**: HPC systems are crucial for weather forecasting, climate modeling, and atmospheric simulations to predict weather patterns, study climate change, and understand the Earth's systems.

4. **Energy Exploration and Production**: HPC is utilized in the energy sector, including oil and gas exploration, reservoir modeling, seismic imaging, and simulating complex geological formations to optimize drilling techniques and enhance energy production.

5. **Financial Services**: HPC plays a significant role in financial modeling, risk analysis, algorithmic trading, portfolio optimization, and simulations used by banks, hedge funds, and financial institutions for decision-making, predicting market trends, and managing risk.

6. **Artificial Intelligence and Machine Learning**: HPC infrastructure is essential for training and deploying large-scale machine learning models, deep learning algorithms, and neural networks, enabling advanced AI applications such as image recognition, natural language processing, and data analytics.

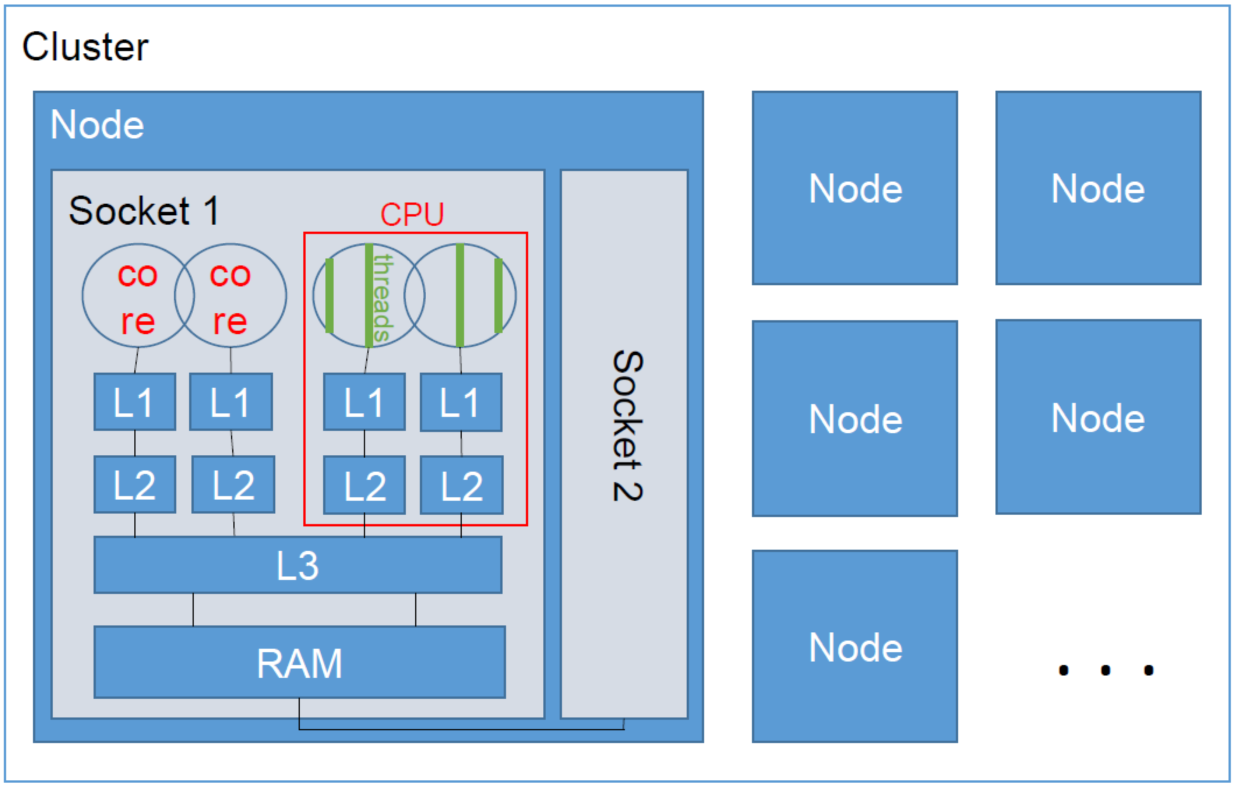
These are just a few examples of the domains where HPC is prominently utilized. However, the potential applications of HPC are broad and continue to expand as technology advances and computational needs evolve.

Cluster

A cluster in the HPC sense refers to a collection of multiple nodes, usually connected via a network providing high bandwidth and low latency communication. Accessing a cluster is possible by connecting to its specific login nodes.

Node

A node is an individual computer consisting of one or more sockets.



Backend Node

Backend nodes (compute nodes) are reserved for executing the actual (scientific) calculation computations (memory demanding and long running applications). These compute nodes are the most powerful, but also most power consuming part of a cluster as they make up around 98% of it. Since these nodes are not directly accessible by the user, a scheduler manages their access. In order to run on these nodes, a batch job needs to be submitted to the batch system via a scheduler specific command.

Copy Node

Copy nodes are reserved for transferring data to or from a cluster. They usually offer a better connection than other nodes and minimize the disturbance of other users on the system. Depending on the facility, software installed on these nodes may differ from other ones due to their restricted use case, though not every facility chooses to install a designated copy node at all. As an alternative login node may be used to move data between systems.

Login Node

Login nodes are reserved for connecting to the cluster of a facility. Most of the time they can also be used for testing and performing interactive tasks (e.g. the analysis of previously collected application profiles). These test runs should generally not exceed execution times of just a few minutes and may only be used to verify that your software is running correctly on the system and its environment before submitting batch jobs to the batch system.

Central Processing Unit(CPU)

The term "CPU" is widely used in the field of HPC, yet it lacks a precise definition. It is mostly used to describe the concrete hardware architecture of a node, but should generally be avoided due to possible misunderstandings and ambiguities - see below.

Socket

A socket is the physical package (processor) which contains multiple cores sharing the same memory.

Core

A core is the smallest unit of computing, having one or more hardware threads (if the hardware setting *Hyperthreading* is configured) and is responsible for executing instructions. In the "pre multicore" era, a processor had just one core.

Thread

- **Hardware Threads**

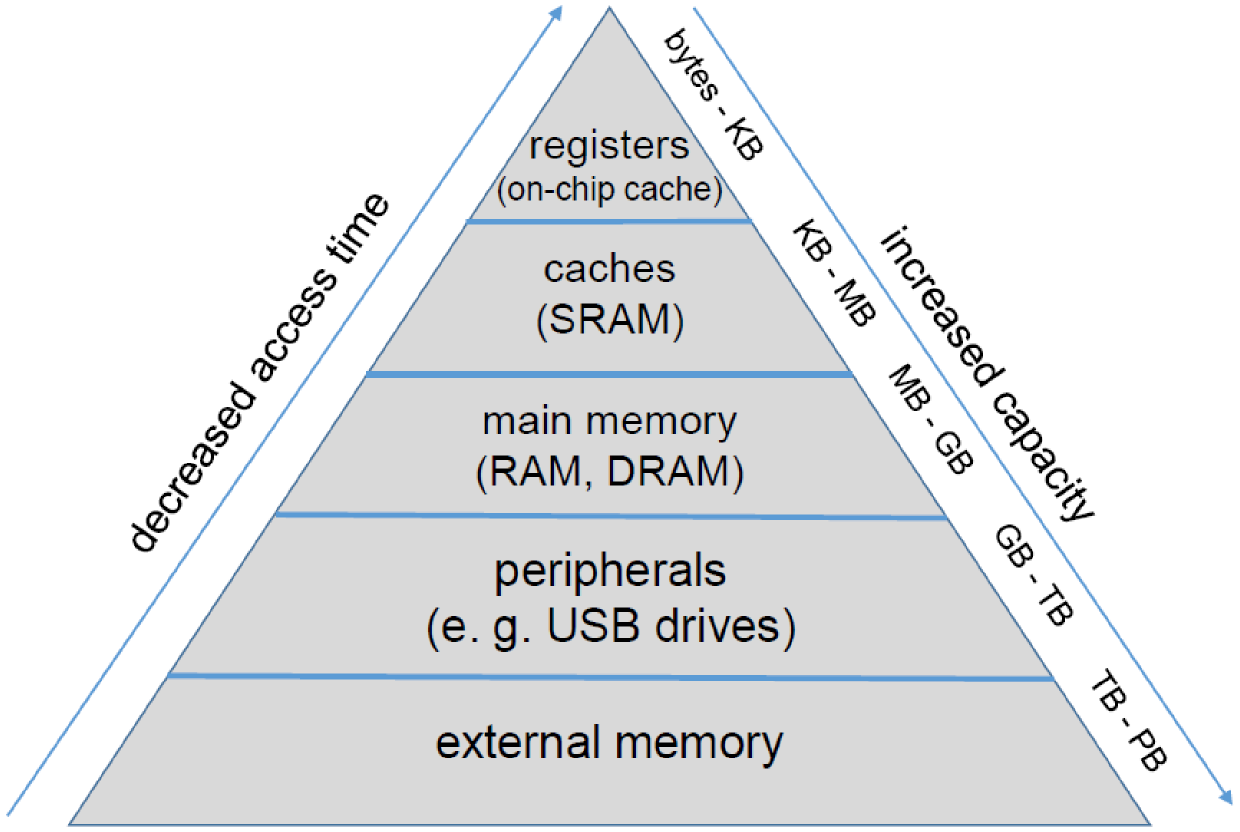
  If the same processor core can (and is configured to) execute more than one execution path in parallel (*Hyperthreading*, a hardware setting).

- **Software Threads**

  A process may have several threads, all sharing the same memory address space, but each thread having its own stack (*Multithreading*, eg. via OpenMP). Threads can be seen as "parallel execution pathes" *within the same program* (eg. to work through different and independant data arrays in parallel).

Software threads are not dependent on hardware threads -- you can run multithreaded applications on systems without Hyperthreading.

Random Access Memory(RAM)



RAM (a.k.a main memory) is used as working memory for the cores. RAM is *volatile* memory, losing all content when the power is lost or switched off. In general, RAM is shared between all sockets on a node (all sockets can access all RAM). In NUMA however, a socket might have access to certain RAM areas only be going through other sockets.

The operating system separates all processes' address space, and also takes care of removing/deleting the content after a process ends.

Cache

A cache is a relatively small amount of very fast memory (compared to RAM), on the processor chip (die). It is used to fetch and hold data from the main memory near to the cores working on them, to allow faster access than RAM.

A modern processor has three cache levels: L1 and L2 are local to each core, while L3 (or *Last Level Cache* (LLC)) is shared among all cores of a CPU.

What is L1 and L2 cache?

L1 cache (Level 1 cache) and L2 cache (Level 2 cache) are types of cache memory that are located closer to the processor in a computer system. Both L1 and L2 caches are designed to provide fast access to frequently used data and instructions, reducing the time it takes for the processor to fetch information from the main memory or external storage.

1. L1 Cache:

L1 cache is the first level of cache memory in the memory hierarchy and is directly integrated into the processor itself. It consists of separate instruction and data caches:

- L1 Instruction Cache (L1i): L1i cache stores instructions that the processor fetches from the main memory. It holds the instructions needed for the processor's execution pipeline, enabling quick access to frequently used instructions and reducing the latency involved in fetching instructions from main memory.
- L1 Data Cache (L1d): L1d cache stores data accessed by the processor. It holds recently accessed data, operands, and variables, improving the overall performance by reducing the time it takes to retrieve data from main memory. The data cache is often split into separate caches for read and write operations.

L1 caches are typically small in size but have very low latency, allowing for extremely fast access to data and instructions. They operate at the same frequency as the processor, which further enhances their performance.

2. L2 Cache:

L2 cache is the second level of cache memory and is located between the L1 cache and the main memory. It serves as a larger and slower cache compared to L1 cache but still provides faster access to data than the main memory. L2 cache is often shared among multiple processor cores in multi-core systems.

The purpose of the L2 cache is to hold a larger portion of frequently accessed data and instructions than the L1 cache. It helps to bridge the speed gap between the faster L1 cache and the slower main memory, reducing the memory latency and improving the overall system performance.

L2 cache sizes can vary depending on the processor architecture and design, ranging from a few megabytes (MB) to several tens of megabytes. The latency of L2 cache is higher than L1 cache, but it is still significantly faster than accessing data from the main memory.

By utilizing multiple levels of cache, including L1 and L2 caches, computer systems can reduce memory latency and improve the efficiency of data and instruction access, resulting in faster processing speeds and improved overall performance.

Why L1 and L2 cache use in multicore cluster processors?

L1 and L2 caches are used in multicore cluster processors to enhance the performance and efficiency of the overall system. Here's why they are employed in this context:

1**. Cache Coherency**: In a multicore cluster processor, each core has its own L1 cache. Having separate L1 caches allows each core to store frequently accessed data and instructions locally, reducing the need to access shared memory or caches of other cores. This improves cache coherency and reduces the need for inter-core communication, leading to better performance and reduced latency.

2**. Core-level Performance:** L1 caches provide extremely fast access to data and instructions directly integrated into each core. This reduces the latency associated with fetching data from main memory, as the core can quickly retrieve frequently used data from its own L1 cache. It helps improve the core-level performance and enables efficient execution of individual threads or tasks running on each core.

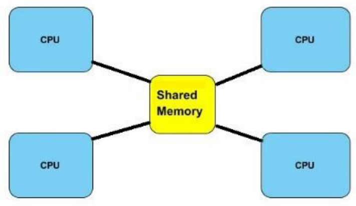
3. **Sharing of Common Data:** While each core has its own L1 cache, they often share a larger L2 cache. The shared L2 cache allows different cores to share commonly accessed data, improving data sharing and communication between cores. When one core modifies data in its L1 cache, the shared L2 cache helps ensure the other cores can access the updated data without needing to fetch it from main memory, reducing latency and enhancing performance.

4. **Reduced Memory Access:** The L1 and L2 caches act as intermediate levels of memory between the processor cores and the main memory. By storing frequently accessed data and instructions in these caches, the need to access the slower main memory is minimized. This reduces memory latency and improves overall system performance.

5. **Scalability:** Multicore cluster processors are designed for scalability, allowing for the integration of multiple cores in a single processor package. The use of L1 and L2 caches in each core enables the efficient utilization of resources, enhances performance, and allows for the scaling of performance as more cores are added to the cluster.

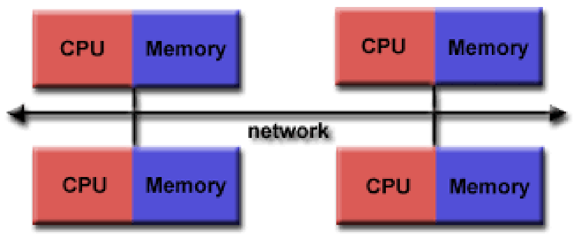
In summary, L1 and L2 caches in multicore cluster processors contribute to improved cache coherency, core-level performance, data sharing, reduced memory access, and scalability. These caches play a crucial role in enhancing the overall performance and efficiency of multicore systems by reducing memory latency, improving data access speeds, and enabling effective parallel computing within the cluster.

Shared Memory :



OpenMP

Distributed Memory :



Message Passing Interface

Concurrency and Parallelism

Concurrency : A condition of a system in which multiple task are logically active at one time.

Parallelism : A condition of a system in which multiple tasks are actually active at one time.

Scalability

Scalability describes how well an application can use an increasing amount of hardware resources. Good scalability in general means reduced runtimes when more and more cores are used to solve the same or larger problems.

Typically, applications will hit an upper limit of cores beyond they don't scale further, ie. more cores don't yield lower runtimes (or even increase it again). This is mainly due to the increasing amount of arbitration and communication necessary to coordinate the working cores.

However, good scalability can also imply that the execution time remains the same, when the problem size grows similarly to the hardware resources.

- **Strong Scalability**

By involving more and more processor cores to solve the **same** problem (size), the application still achieves reduced runtimes.

- **Weak Scalability**

By involving more and more processor cores, the application can tackle **larger** problems (size), though you can no longer achieve reduced runtimes on the **same** problem (size).

Scheduler

The scheduler is the administrative heart of any HPC cluster. Like a dispatcher, it receives batch jobs submitted by the users, arbitrating and coordinating their placement and execution on various compute nodes.

The batch script to be submitted can contain directives (pragmas) to control job placement (eg. "put this job on any node with at least 16 cores and 32 GByte main memory" or "put this job on any node with a graphics card (GPU)".

Why CentOS use in HPC cluster?

CentOS (Community Enterprise Operating System) is a popular Linux distribution that is often used in High-Performance Computing (HPC) clusters for several reasons:

1. **Stability and Long-Term Support**: CentOS is known for its stability and reliability. It is built from the source code of Red Hat Enterprise Linux (RHEL), which is widely used in enterprise environments. CentOS provides long-term support, including security updates and bug fixes, which is crucial for HPC systems that require continuous operation and minimal downtime.

2. **Compatibility with RHEL**: CentOS is binary compatible with RHEL, meaning software and applications developed and tested on RHEL can run seamlessly on CentOS. Many commercial software packages and libraries used in the HPC domain are specifically designed and validated for RHEL-based distributions. By using CentOS, HPC clusters can ensure compatibility with a wide range of software tools and libraries.

3. **Package Availability**: CentOS offers a vast repository of pre-compiled software packages, including HPC-related software stacks, scientific libraries, compilers, and development tools. These packages are frequently updated and maintained by the CentOS community, making it convenient for HPC users to install and manage the required software components.

4. **Community Support**: CentOS has a large and active user community, which means there is a wealth of knowledge, documentation, and online resources available for troubleshooting, configuration, and optimization. Users can benefit from community-driven forums, mailing lists, and online tutorials to seek help and share experiences related to HPC deployments.

5**. Cost-Effectiveness**: CentOS is an open-source distribution and freely available, making it an attractive choice for organizations looking for cost-effective solutions for their HPC clusters. It provides enterprise-grade features and performance without the need for expensive licensing fees associated with some commercial operating systems.

Overall, the use of CentOS in HPC clusters offers stability, compatibility, a rich software ecosystem, community support, and cost-effectiveness, making it a favored choice among many HPC practitioners and system administrators.

Why InfiniBand use in HPC cluster?

In general, HPC clusters utilize high-speed interconnect technologies to enable efficient communication and data transfer between computing nodes. Examples of commonly used interconnect technologies in HPC clusters include InfiniBand, Ethernet, and Omni-Path.

InfiniBand is a popular interconnect technology widely used in HPC clusters due to its high bandwidth, low latency, and scalability. It offers high-speed data transfers and low communication overhead, allowing for efficient parallel computing and communication between nodes.

However, if you are referring to a specific technology called "Infinity Band" that I'm not aware of, I apologize for the misunderstanding. If you can provide more details or clarify the term, I'll do my best to assist you further.

Which is various type of interconnect technology in HPC cluster?

HPC clusters employ various interconnect technologies to facilitate fast and efficient communication between the computing nodes. Here are some commonly used interconnect technologies in HPC clusters:

1. **InfiniBand**: InfiniBand is a high-speed interconnect technology that offers low latency, high bandwidth, and high message rate capabilities. It is widely used in HPC clusters due to its performance and scalability. InfiniBand provides both point-to-point and switched fabric configurations, allowing for high-speed communication between nodes.

2. **Ethernet**: Ethernet, specifically enhanced Ethernet technologies such as 10 Gigabit Ethernet (10GbE), 25 Gigabit Ethernet (25GbE), and 100 Gigabit Ethernet (100GbE), is commonly used in HPC clusters. While Ethernet traditionally had higher latency compared

to other interconnects, advancements in technology and the emergence of RDMA (Remote Direct Memory Access) over Ethernet have made it a viable option for HPC clusters.

3. **Omni-Path Architecture (OPA)**: Developed by Intel, Omni-Path Architecture is a high-performance interconnect technology designed specifically for HPC systems. OPA offers low latency, high bandwidth, and high scalability, enabling efficient parallel computing and data communication in large-scale clusters.

4. **Cray's Aries**: Cray's Aries interconnect is utilized in Cray supercomputers and HPC systems. It provides low-latency communication and high-bandwidth performance, optimizing data movement between nodes and enabling high-speed parallel computing.

5. **Myrinet**: Myrinet is a high-speed, low-latency interconnect technology that has been used in HPC clusters. It offers high bandwidth and is particularly suitable for message-passing parallel computing models.

6. **High-Speed Interconnect Fabrics**: Other high-speed interconnect fabrics like Fiber Channel (FC), RapidIO, and proprietary interconnects designed by specific vendors may be used in certain HPC clusters based on specific requirements and system configurations.

The choice of interconnect technology depends on factors such as performance requirements, scalability, latency tolerance, cost, and the specific needs of the HPC workload. Each interconnect technology has its own strengths and trade-offs, and the selection depends on the specific requirements of the HPC cluster and the applications it supports.

Which architecture use in HPC cluster?

HPC clusters typically utilize a distributed computing architecture. This architecture involves the interconnected deployment of multiple computing nodes, where each node functions as an independent computer with its own processor(s), memory, and storage. These nodes work together in parallel to perform computational tasks and collectively provide high-performance computing capabilities.

The distributed computing architecture of an HPC cluster can take different forms, including:

1. **Symmetric Multiprocessing (SMP)**: SMP architecture involves multiple processors connected to a shared memory system. Each processor can access the shared memory, allowing for easy sharing and communication of data among the processors. SMP architectures are suitable for parallel computing within a single node.

2. **Message Passing Interface (MPI)**: MPI is a communication protocol and programming model commonly used in HPC clusters. It allows for efficient communication and data exchange between nodes in a distributed memory system. MPI enables parallel computing across multiple nodes, where each node operates independently and communicates through message passing.

3. **Distributed Shared Memory (DSM)**: DSM is an architectural approach that provides the illusion of shared memory across distributed nodes. It allows multiple nodes to access and share memory as if it were a single shared memory space, even though physically it is distributed across nodes. DSM architectures require specialized software and coherence protocols to ensure consistency and efficient memory access across nodes.

4. **Hybrid Architectures**: Some HPC clusters employ a combination of different architectures, such as a mix of SMP and MPI or a combination of distributed memory and shared memory models. These hybrid architectures leverage the strengths of each architecture to optimize performance and efficiency for specific types of applications or workloads.

The specific choice of architecture depends on factors such as the nature of the applications being run, the scalability requirements, memory access patterns, communication needs, and the available hardware resources. HPC system designers and administrators carefully consider these factors to determine the most suitable architecture for their cluster to achieve high-performance computing capabilities.

Why use MIMD in HPC cluster?

Multiple Instruction Multiple Data (MIMD) is a parallel computing model that allows for the concurrent execution of multiple instructions on multiple sets of data. MIMD is commonly used in HPC (High-Performance Computing) clusters for several reasons:

1. **Parallelism:** HPC clusters are designed to tackle computationally intensive tasks that can be divided into smaller subtasks. MIMD allows these subtasks to be executed simultaneously across multiple processing units (nodes) in the cluster. By leveraging parallelism, the overall computation time can be significantly reduced, leading to faster results.

2. **Performance Scaling:** With MIMD, HPC clusters can scale their computational resources effectively. As the number of nodes in the cluster increases, more instructions can be executed in parallel, resulting in increased computational power. This scalability is essential for handling large-scale simulations, data analysis, and complex scientific computations.

3. **Task Independence**: In HPC applications, there are often many independent tasks that can be executed concurrently. MIMD architecture enables the execution of different instructions on different sets of data, allowing independent tasks to be processed simultaneously. This maximizes the utilization of cluster resources and accelerates overall computation.

4. **Data Parallelism:** MIMD supports data parallelism, which is crucial in many HPC workloads. Data parallelism involves dividing a large dataset into smaller subsets and distributing them across multiple nodes. Each node operates on its assigned data subset in

parallel, allowing for efficient processing of large-scale data analysis, simulations, and computations.

5. **Flexibility:** HPC clusters are utilized for a wide range of scientific, engineering, and data-intensive applications. The MIMD model offers flexibility to execute different instructions and algorithms simultaneously on various datasets, accommodating diverse workloads and applications.

6. **Efficient Resource Utilization**: MIMD enables load balancing across the cluster by distributing instructions and data among different nodes. This ensures that the computational workload is evenly distributed, preventing resource bottlenecks and maximizing the utilization of cluster resources.

In summary, MIMD is employed in HPC clusters to harness parallelism, achieve high-performance scaling, handle task independence, exploit data parallelism, provide flexibility for diverse workloads, and optimize resource utilization. By leveraging the capabilities of MIMD, HPC clusters can achieve faster computations and handle complex scientific and computational tasks efficiently.

Need of Parallel Programming

- The exponential growth of processing and networks speeds means that parallel architecture isn't just a good idea it's necessary. Big data and the IoT will soon force us to crunch trillions of data points at once.
- Dual-core, quad-core, 8-core and even 56-core chips are all example of parallel computing. So, while parallel computers aren't new, here's the rub; new technologies are cranking out even-faster networks, and compute performance has grown 250,000 times in 20 years.
- For instance, in just the healthcare sector, AI tools will be rifling through the heart rates of a hundred millions patients, looking for the telltale sign of A-fib or V-tach and saving lives. They won't be able to make it work if they have to plod along performing one operation at a time.

Why Parallel Programming?

- Parallel computing models the REAL WORLD.
- Saves Time
- Saves Money
- Solve more Complex or Linear Problems
- Make better use of underlying Parallel Hardware