UniTs - University of Trieste

Faculty of Scientific and Data Intensive Computing

Department of mathematics informatics and geosciences

High Performance Computing

Lecturer:
Prof. Stefano Cozzini

Author:
Andrea Spinelli

March 4, 2025

github.com/Spina02 andreaspinelli2002@gmail.com

Abstract

As a student of Scientific and Data Intensive Computing, I've created these notes while attending the **High Performance Computing** module of **High Performance and Cloude Computing** course. The course will introduce the fundamentals of High Performance Computing, exploring both its concepts and practical applications. The notes cover a wide range of topics, including:

- An overview of High Performance Computing and its importance in solving complex, real-world problems.
- The principles behind modern computer architectures and how they influence performance.
- Essential tools and techniques for parallel programming, alongside strategies to optimize code for advanced architectures.
- The evolution of computing facilities and how to effectively leverage them for large-scale computational challenges.
- Developing a proactive mindset, moving beyond the use of pre-packaged tools to a deeper understanding of the underlying systems.

While these notes were primarily created for my personal study, they may serve as a valuable resource for fellow students and professionals interested in High Performance Computing.

Contents

1 Introduction

1.1 Base Concepts

High Performance Computing (HPC), also known as ***supercomputing***, refers to computing systems with extremely high computatiional power that are able to solve hugely complex and demanding problems. [1]

Often, high precision and accuracy are required in scientific and engineering simulations, which can be achieved by increasing the computational power of the system. This is where HPC comes into play, as it allows for the execution of large-scale **simulations** of complex problems in a reasonable amount of time. Simulations have become the key method for researching and developing innovative solutions in both scientific and engineering fields. They are especially prominent in leading domains such as the aerospace industry and astrophysics, where they enable the investigation and resolution of highly complex problems. However, the increasing reliance on simulation also introduces significant **challenges related to complexity, scalability, and data management**, which in turn impact the supporting IT infrastructure.

As scientific inquiry progresses along what is known as the *Inference Spiral of System Science*, the complexity of models intensifies and the influx of new data enriches these systems with additional insights. Consequently, this dynamic evolution necessitates ever increasing computational power to efficiently handle the enhanced simulations and data management challenges.

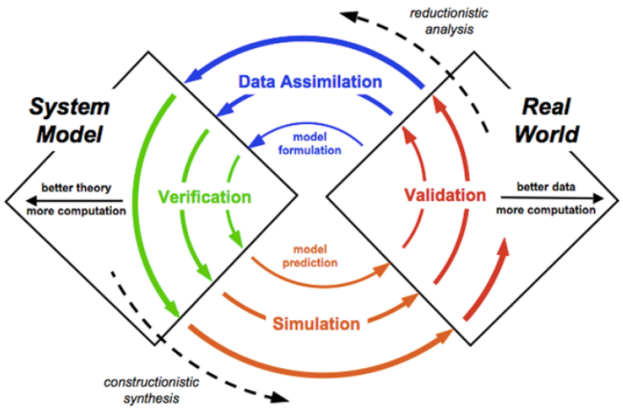

Figure 1.1: Research and Development

Prefix	Symbol	Value
Yotta	Y	10^{24}
Zetta	Z	10^{21}
Exa	E	10^{18}
Peta	P	10^{15}
Tera	T	10^{12}
Giga	G	10^{9}
Mega	M	10^{6}
Kilo	K	10^{3}

Table 1.1: Prefixes in HPC

Observation:

In today's world, larger and larger amounts of data are constantly being generated, from 33 zettabytes globally in 2018 to an expected 181 zettabytes in 2025. This exponential growth is driving a shift towards data-intensive applications, making HPC indispensable for processing and analyzing these vast datasets efficiently. Consequently, HPC is key to unlocking valuable insights that benefit citizens, businesses, researchers, and public administrations. [1]

1.1.1 What is High Performance Computing?

High Performance Computing (HPC) involves using powerful servers, clusters, and supercomputers, along with specialized software, tools, components, storage, and services, to solve computationally intensive scientific, engineering, or analytical tasks.

HPC is used by scientists and engineers both in research and in production across industry, government and academia.

Key elements of the HPC ecosystem include:

- **Hardware:** High-performance servers, clusters, and supercomputers.
- **Software:** Specialized tools and applications designed to optimize complex computations.
- **Applications:** Scientific, engineering, and analytical tasks that leverage high computational power.

People in HPC

Human capital is by far the most important aspect in the HPC landscape. Two crucial roles include HPC providers, who plan, install, and manage the resources, and HPC users, who leverage these resources to their fullest potential. The mixing and interplaying of these roles not only enhances individual competence but also drives overall advancements in high-performance computing.

1.1.2 Performance and metrics

Performance in the realm of high-performance computing is a multifaceted concept that extends far beyond a mere measure of speed. While terms such as "how fast" something operates are often used to describe performance, they tend to be vague. Many factors contribute to the overall performance of a system, and the interpretation of these factors can vary depending on the specific context and objectives of the computational task. Performance, therefore, remains a complex and central issue in the field of HPC, as it involves more than just the raw computational speed.

The discussion often extends to the idea that the "P" in HPC might stand for more than just performance. A growing sentiment among professionals in the field suggests that high performance should be complemented by high productivity. This broader view recognizes that the true efficiency of a computing system is not only determined by its ability to perform tasks quickly but also by the ease and speed with which applications can be developed and maintained. In other words, while raw performance is critical, the overall productivity of a system—combining the system's speed with the programmer's effort—plays an equally important role.

To further clarify the distinction, consider that performance can be seen as a measure of how effectively a system executes tasks, whereas productivity is the outcome achieved relative to the effort invested in developing the application. For instance, if a code optimization leads to a system that runs twice as fast but requires an extensive period of development—say, six months of work—the benefits of the improvement must be weighed against the increased effort required. This example underlines the importance of balancing performance gains with the associated development costs.

Ultimately, the challenge lies in understanding and optimizing both aspects. A successful HPC system is one that not only achieves high computational throughput but also enhances the productivity of the developers who create and refine the applications. This balance is essential for advancing the capabilities of high-performance computing in both research and production environments.

Number Crunching on CPU

When evaluating the performance of a high-performance computing (HPC) system, one of the most fundamental metrics is the rate at which floating point operations are executed. This rate is typically expressed in millions (Mflops) or billions (Gflops) of operations per second. In essence, it quantifies how many calculations, such as additions and multiplications, the system is capable of performing every second.

To estimate this capability, we rely on the concept of theoretical peak performance. This value is computed by considering the system's clock rate, the number of floating point operations that can be executed in a single clock cycle, and the total number of processing cores available. Under ideal conditions, the theoretical peak performance can be expressed as follows:

$$\text{FLOPS} = \text{clock\_rate} \times \text{Number\_of\_FP\_operations} \times \text{Number\_of\_cores}$$

This formula provides an upper bound on the computational power of the system. However, it is important to note that this is a best-case scenario estimate and does not always reflect the performance achievable in real applications.

Sustained (Peak) Performance

While the theoretical peak performance offers insight into the maximum potential of an HPC system, the actual performance observed during real-world operations is better captured by the sustained (or peak) performance. In practice, several factors such as memory bandwidth limitations, communication latencies, and input/output overhead can prevent a system from reaching its theoretical maximum.

Sustained performance refers to the effective throughput that an HPC system attains when executing actual workloads. Since it is challenging to exactly measure the number of floating point operations performed by every application, standardized benchmarks are commonly used to assess this performance. One widely recognized benchmark is the HPL Linpack test, which forms the basis for the TOP500 list of supercomputers. This benchmark emphasizes the importance of sustained performance, as it reflects the system's efficiency and reliability under realistic operational conditions.

Understanding both the theoretical and sustained performance metrics is crucial. While the former provides an idealized estimate of a system's capabilities, the latter offers a more practical perspective, thereby guiding decisions on system improvements and resource allocation in high-performance computing environments.

…

1.1.3 Moore Law

Tipically, the Moore Law is stated as: "Performance doubles every 18 months". However, it is actually closer to "The number of transistors per unit cost doubles every 18 months".

The original Moore Law was formulated by Gordon Moore, co-founder of Intel, in 1965. He predicted that:

Definition:

"The complexity for minimum component costs has increased at a rate of roughly a factor of two per year. [...] Over the longer term, the rate of increase is a bit more uncertain, although there is no reason to believe it will not remain nearly constant for at least 10 years."

~Gordon Moore, 1965

Dennard Scaling: From Moore's Law to performance

Definition:

"Power density stays constant as transistors get smaller"

~Robert H. Dennard, 1974

The concept of Dennard scaling, named after Robert Dennard, an IBM researcher, is closely related to Moore's Law. Dennard scaling refers to the observation that as transistors shrink in size, their power density remains constant. This phenomenon allowed for the continuous increase in clock speeds and performance of microprocessors over the years.
However, Dennard scaling began to break down around the early 2000s, as power consumption and heat dissipation became significant challenges. Consequently, the industry shifted its focus from increasing clock speeds to improving parallelism and energy efficiency.

Intuitively:

- Smaller transistors $\rightarrow$ shorter propagation delay $\rightarrow$ faster frequency
- Smaller transistors $\rightarrow$ smaller capacitance $\rightarrow$ lower power consumption

$$Power \propto Capacitance \times Voltage^2 \times Frequency$$

End of Dennard Scaling: Power wall

The power wall is a fundamental limit on the amount of power that can be dissipated by a chip. This limit is determined by the chip's thermal design power (TDP), which is the maximum amount of heat that the cooling system can dissipate. As the number of transistors on a chip increases, the power consumed by the chip also increases, eventually reaching the TDP limit. When this limit is reached, the chip can no longer dissipate the heat generated by the transistors, leading to overheating and reduced performance.
However, the original Moore's Law is still valid, as the number of transistors per unit cost continues to double every 18 months, but no more on a single core. Instead, the industry has shifted towards multi-core processors and parallel computing to continue improving performance.

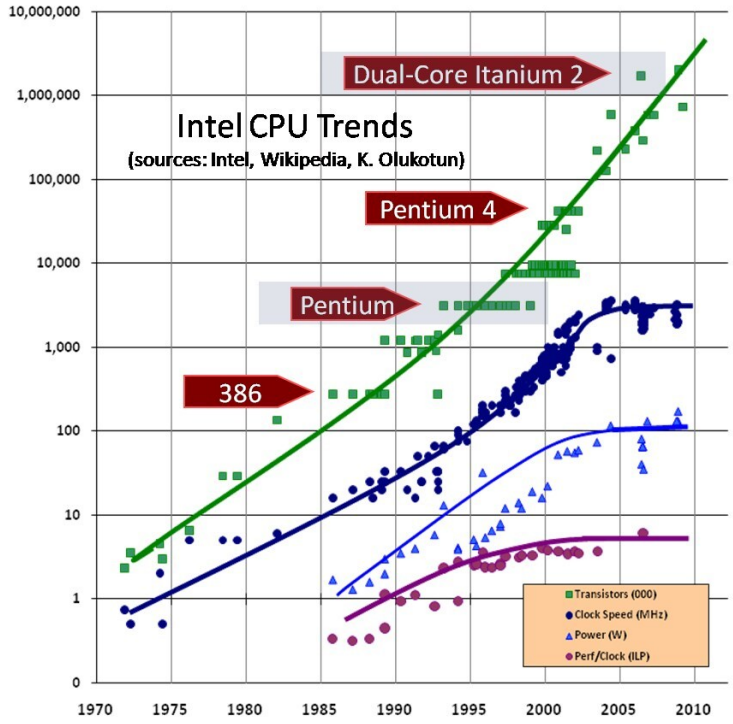

Figure 1.2: Moore's Law

This evolution marks what many computer scientists and engineers refer to as the end of the "free lunch" era, which began around 2006. Prior to this shift, software developers could rely on hardware improvements to automatically enhance their applications' performance without significant code optimization. Single-core performance scaling, which had been the primary driver of computational advances for decades, effectively plateaued as the industry encountered fundamental physical limitations.

The computing community has responded to this challenge through two complementary approaches:

The Software Solution: This approach emphasizes the critical importance of efficient software design and implementation. As hardware improvements no longer automatically translate to performance gains, developers must engage in deliberate "performance engineering"—applying sophisticated optimization techniques informed by deep understanding of hardware architecture. This involves careful algorithm selection, memory access pattern optimization, and exploitation of instruction-level parallelism to maximize the utilization of available hardware resources.

The Specialized Architectural Solution: The second approach acknowledges a fundamental shift in design constraints: while chip space has become relatively inexpensive, power consumption has emerged as the primary limiting factor. Rather than continuing to develop increasingly complex general-purpose processing cores, this approach advocates for heterogeneous computing systems. Such systems incorporate specialized accelerators (such as GPUs, TPUs, and FPGAs) that are optimized for specific computational patterns. This architectural diversification allows for significant performance improvements in targeted application domains while maintaining reasonable power consumption profiles.

These complementary strategies represent the computing industry's response to the physical limitations that have constrained traditional performance scaling. By combining software optimization with hardware specialization, the field continues to advance computational capabilities even as the straightforward scaling of single-core performance has reached its practical limits.

1.1.4 The Shift to Multicore Architecture

Modern CPUs have evolved into multicore processors due to physical constraints in power consumption and heat dissipation, with manufacturers reducing clock frequencies while increasing core count to deliver greater computational throughput within manageable thermal profiles. These independent cores can execute separate instruction streams simultaneously but share critical resources including memory hierarchies, controllers, and peripheral subsystems, creating a complex environment where cores must cooperate and compete for resources. This architectural shift effectively circumvents the limitations of traditional single-core scaling but presents new challenges for software developers, who must now explicitly design for parallelism to fully leverage available computational capabilities.
[Hardware accelerators image]

1.1.5 Parallel Compuers

Parallel computing is a type of computation in which many calculations or processes are carried out simultaneously. Flynn Taxonomy is a classification of parallel computer architectures, proposed by Michael J. Flynn in 1966. It categorizes computer systems based on the number of instruction streams and data streams that can be processed concurrently. The four categories are shown in Table 1.2.

	HW level	**SW level**
SISD	A Von Neumann CPU	no parallelism at all
MISD	On a superscalar CPU, different ports executing different *read* on the same data	ILP on same data; multiple tasks or threads operating on the same data
SIMD	Any vector-capable hardware (vector registers on a core, a GPU, a vector processor, an FPGA, ...)	data parallelism through vector instructions and operations
MIMD	Every multi-core processor; on a superscalar CPU, different ports executing different ops on different data	ILP on different data; multiple tasks or threads with different data on each core

Table 1.2: Comparison of SISD, MISD, SIMD, and MIMD at HW and SW levels

While Flynn's taxonomy provided a foundational classification system in 1966, its utility for categorizing modern HPC infrastructure has diminished significantly. The dramatic evolution of CPUs and computing architectures over the past six decades has produced systems with hybrid designs that transcend these simple classifications. Nevertheless, the fundamental concepts of SIMD and MIMD remain relevant principles that continue to influence the design and implementation of contemporary HPC hardware solutions.

1.1.6 Essential Components of a HPC Cluster

- Several computers (nodes)
  Often in special cases (1U) for easy mounting in racks
- One or more networks (interconnects) to hook the nodes together
- Some kind of storage
- A login/access node

[Cluster image]

1.2 Single CPU topology

Modern CPUs are multy- (or many-) core processors.

Definition:

A **core** is the smallest unit of computing, having one or more (hardware/software) threads and is responsible for executing instructions.

A CPU uses a **Cache hierarchy** to store data and instructions. The cache hierarchy consists of several levels of cache, each with different sizes and access times. The cache hierarchy is designed to minimize the time it takes to access data and instructions, which can significantly improve the performance of the CPU.
[CPU layout image]
[Node topology image]
[Overall topology image]
…

1.2.1 memory

on a supercomputer there is a hybrid approach as for the memory placement:

- **Shared memory:** the memory on a single nodes can be accessed directly by all the cores on that node, meaning that memory access is a "read/write" instructions irrespectively of what exact memory bank it refers to.
- **distributed memory:** when you use many nodes at a time, a process can not directly access the memory on a different node. It need to issue a request for that, not a read/write instruction.

These are hardware concepts, i.e. they describe how the memory is physically accessible. However, they do also refer to programming paradigms, as we'll see in a while.

Tip: ***Notation***

- **Multiprocessor:** server with more than 1 CPU
- **Multicore:** CPU with more than 1 core
- **Processor** = CPU = Socket

Note that sometimes the term "processor" is used to refer to the CPU, sometimes to the core.

Shared Memory: UMA

Uniform memory access (UMA): Each processor has uniform access to memory. Also known as symmetric multiprocessing (SMP).

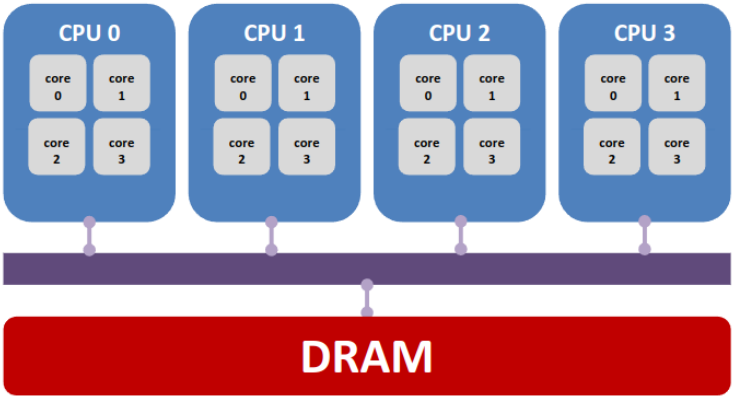

Figure 1.3: Uniform Memory Access (UMA)

Shared memory: NUMA

Non-uniform memory access (NUMA): Time for memory access depends on location of data. Local access is faster then non-local access.

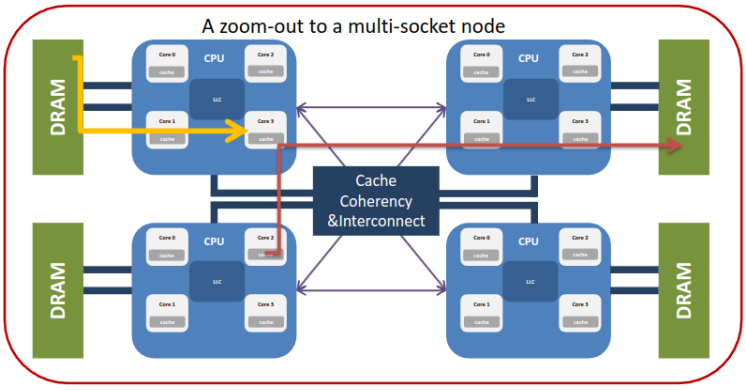

Figure 1.4: Non-Uniform Memory Access (NUMA)

Parallelism within a HPC node

A single node can have multiple cores, each with multiple hardware threads. This introduces several levels of parallelism:

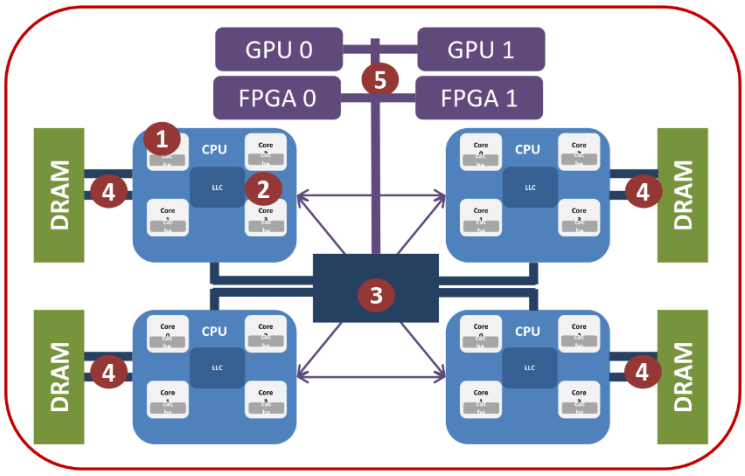

Figure 1.5: Levels of parallelism

1. The first level parallelism is in a single core of a CPU
2. The second level of parallelism is between cores of a single CPU
3. The third level of parallelism is introduced by inner cache levels
4. The fourth level of parallelism is between CPUs of a single node.
5. A node can also have accelerators, like GPUs or FPGAs which introduce another level of parallelism.

Bibliography

[1] *High Performance Computing — digital-strategy.ec.europa.eu*. https://digital-strategy.ec.europa.eu/en/policies/high-performance-computing. [Accessed 03-03-2025].