

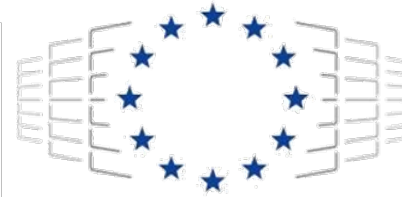
Introduction to High-Performance Computing

Dr Nikos Bakas nibas@grnet.gr

National Infrastructures for Research and Technology - GRNET



Co-funded by
the European Union



EuroHPC
Joint Undertaking



EuroCC@Greece

<https://eurocc-greece.gr/>

Contents

1	General Concepts of High Performance Computing (HPC)	5
2	Scaling	15
3	History of HPC	29
4	Programming Models in HPC	39
5	State of the art machines	47
6	EuroCC Services	53
7	Apply for Access at EuroHPC JU	59
8	Resources	81

Chapter 1

General Concepts of High Performance Computing (HPC)

1.1 Definition of HPC

High Performance Computing (HPC) refers to the practice of aggregating computing power in a way that delivers **much greater performance** than one could get out of a typical desktop computer or workstation in order to solve large problems in **science, engineering, and business**. HPC systems have the ability to process data and perform **complex calculations** at **high speeds**, which is essential for various research and industrial tasks.



1.2 Importance of HPC

The advancement and **breakthroughs in many scientific domains** are often attributed to the computational power available through HPC. These include:

- Weather forecasting and climate research
- Molecular modeling and simulations in pharmaceutical research
- Cryptography and cybersecurity
- Fluid dynamics calculations for designing vehicles, aircraft, etc.
- Energy exploration and seismic analysis
- Simulations with Finite Elements
- High-Performance Data Analytics
- Machine Learning and Artificial Intelligence

HPC is critical in these fields because it enables the simulation and analysis of complex systems and phenomena at a **scale or speed that is not possible with standard computing resources**.

1.3 Components of an HPC System

An HPC system typically involves:

- **Compute Nodes:** These are the individual servers that provide the processing power. Each node contains one or more CPUs (Central Processing Units) or GPUs (Graphics Processing Units), and often both.
- **Networking:** A high-speed network interconnects the compute nodes, allowing for rapid communication and data transfer between nodes.
- **Storage:** Fast, large-capacity storage systems are required for input/output (I/O) operations, storing the data that is generated and used by applications running on the HPC system.
- **Software:** This includes the operating systems, programming models, and applications and tools specific to HPC tasks.

1.4 Parallel Computing

1.4.1 Instructions

An instruction in the context of computing and processors refers to a **basic command** that **tells** the computer's **processor** to perform a **specific operation**.

These operations can include:

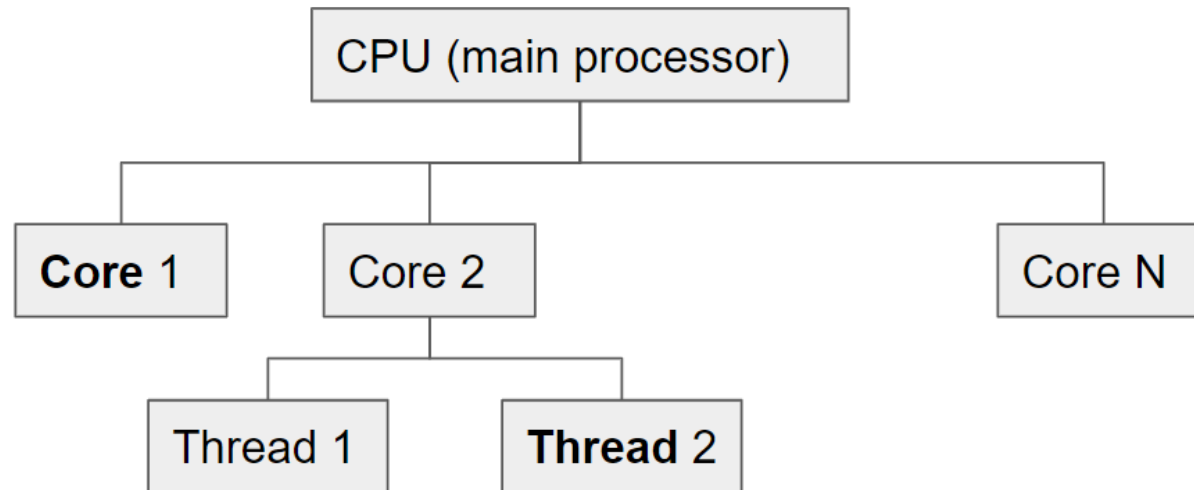
- **arithmetic** (like addition, subtraction, multiplication, and division),
- **logical operations** (like AND, OR, NOT, XOR), and
- **data movement** (such as loading data from memory into a register¹. or storing data from a register into memory),
- **control operations** (such as jumping to a different part of the program, conditional execution, and calling functions).

A program, whether it's a simple application or a complex operating system, is essentially **a collection of instructions** that the CPU executes.

In essence, instructions enable software to interact with hardware, providing a bridge between high-level programming languages and the binary operations that a CPU can directly execute. At the lowest level, the “language” the CPU truly understands is binary (1s and 0s). Each operation (like addition) is represented by a specific pattern of these binary digits. These sequences of bits directly trigger the electrical circuits within the CPU to carry out operations.

¹Registers are super-fast storage locations directly within the CPU. They are very limited in number and size (holding only a few bytes)

1.4.2 CPUs, Cores and Threads



- **Core:** a physical processing unit within a computer's central processing unit (CPU) that can independently execute instructions.
- **Thread:** a sequence of instructions that can be executed independently by a CPU. A CPU with multiple threads can work on multiple tasks in parallel, by using hyper-threading technology. This allows a single core to work on multiple threads concurrently.

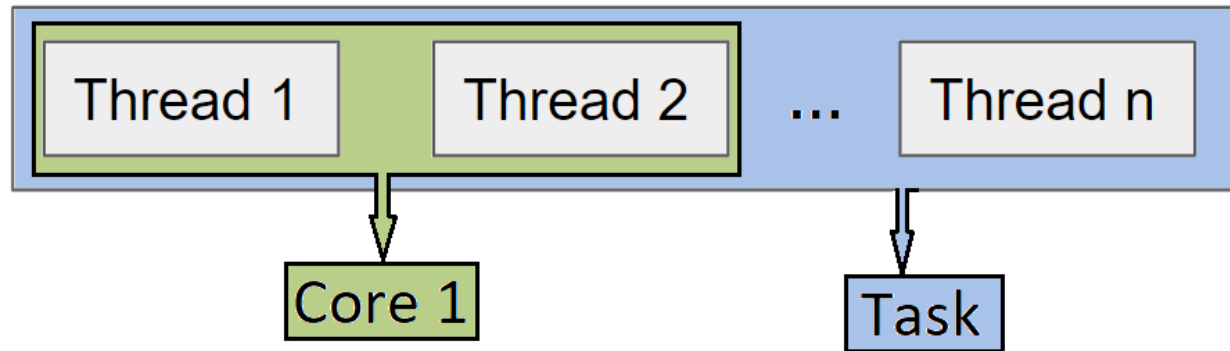
1.4.3 Threads: Software vs Hardware

Threads bridge the distance between software and hardware.

- **Hyper-threading:** Modern CPUs can support hyper-threading, where **one physical core** appears as multiple logical cores to the operating system. This creates the **illusion of parallel execution**.
- **Execution on Cores:** Threads are **ultimately** executed by the **physical cores** of a CPU.
- **Number of Threads per Core:** Traditionally, a single core could only execute one thread at a time. However, with hyper-threading technology, the number of threads supported by a core with hyper-threading typically ranges from **2 to 4**, depending on the CPU architecture.
- **Operating System Management:** The operating system's kernel creates, schedules, and manages **threads**. It views threads as **units of execution**.

Threads are fundamentally a **software abstraction** primarily managed by the **operating system**. designed to optimize **multitasking**. However, **they rely heavily on underlying hardware** features for efficient implementation.

1.4.4 Tasks, Threads, and Cores

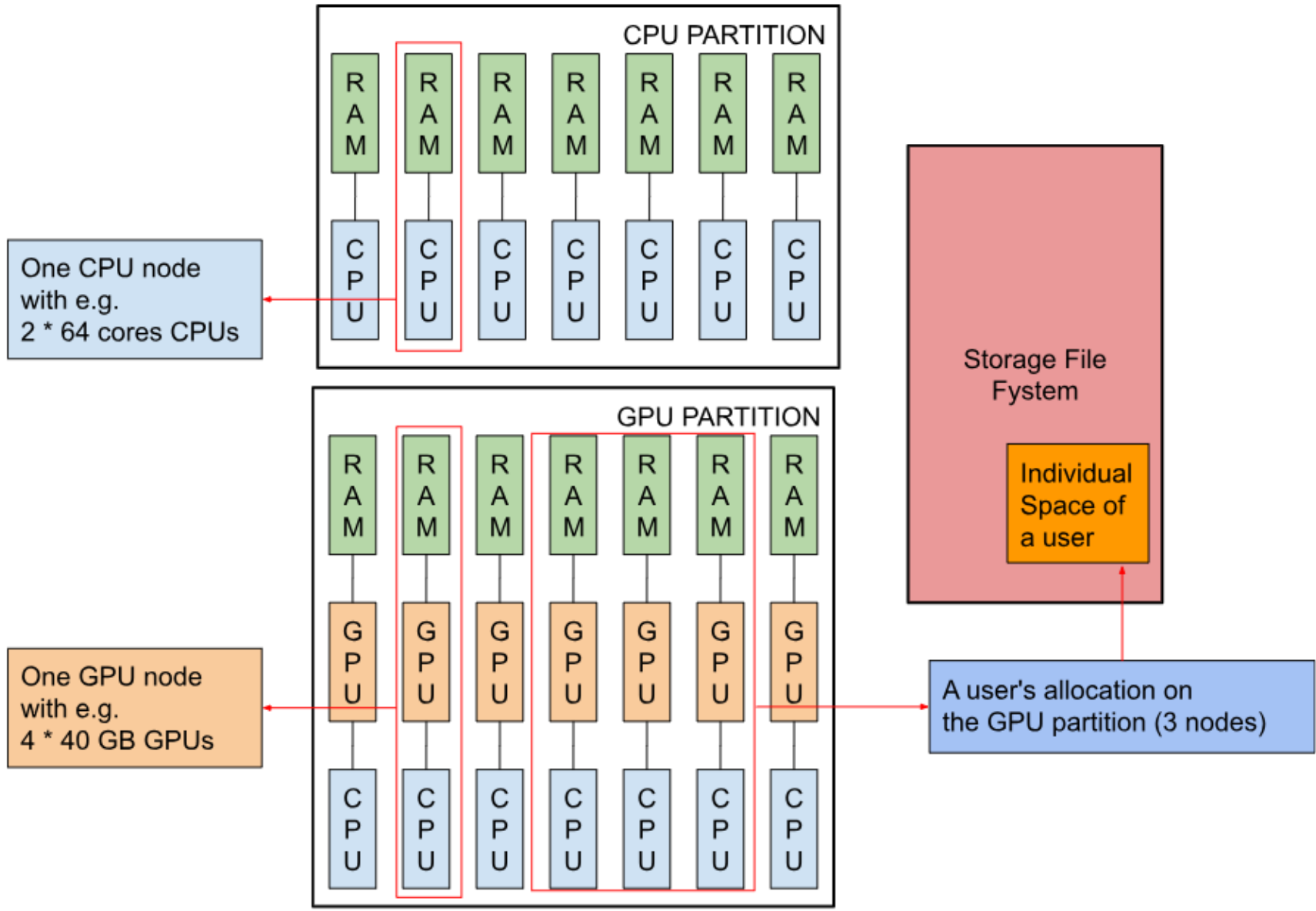


- A **task**, is a program instance submitted to a job scheduler like Slurm. It manages allocation of cores and memory for parallel execution.

A task can consist of multiple threads, which can be scheduled to execute concurrently on one or more cores depending on the system configuration and the operating system's thread scheduler.

1.5 CPUs, GPUs and Nodes

- A cluster divided into CPU and GPU partitions, each with dedicated RAM and CPUs/GPUs.
- Individual user space is allocated on the hard drive for storage.
- A CPU node highlighted - multiple CPU nodes exist within the CPU partition.
- User's allocation in the GPU partition is indicated, showing a combination of RAM and GPU resources assigned to a user.



1.5.1 Sample Slurm Script

Sample slurm script to allocate 4 nodes, with 2 * 64 cores each, with hyperthreading (2 threads per core). Check also <https://slurm.schedmd.com/sbatch.html>.

```
#!/bin/bash

#SBATCH --job-name=python_job      # Descriptive job name
#SBATCH --nodes=4                  # Request 4 compute nodes
#SBATCH --ntasks-per-node=64       # 64 tasks per node
#SBATCH --cpus-per-task=2          # Allocate 2 CPUs per task (for hyperthreading)
#SBATCH --time=04:00:00            # Set a job time limit of 4 hours
#SBATCH --partition=highmem        # Submit to a partition suitable for your needs
#SBATCH --mem=512GB                # Example: Request 512GB memory per node

# Load necessary modules
module load python/3.9 # Example: Load Python 3.9 module
# Run your Python script
mpirun -np 256 python my_python_script.py
```

1.6 Types of Parallel Computing

HPC relies heavily on parallel computing, where multiple computational tasks are carried out simultaneously to increase the computational speed. Parallel computing, in the context of HPC, can be explained on different levels:

- **Bit-Level Parallelism:** Refers to the ability of a processor to perform operations on different bits or sets of bits simultaneously. For example, a 64-bit processor can operate on 64 bits at a time, thereby performing operations faster than a 32-bit processor, which can only handle 32 bits at once.
- **Instruction-Level Parallelism:** Refers to the ability of a CPU to execute multiple instructions simultaneously, such as adding numbers and loading files simultaneously. ILP is focused on executing multiple instructions of a single thread in parallel at the hardware level.
- **Data Parallelism:** Involves performing the same operation on each element of a larger data set in parallel. The same operation (like adding 2) is done at the same time across many pieces of data, often using vectorized computations. This is efficient for operations on large arrays or matrices.
- **Task Parallelism:** Different tasks are distributed across processors, which may then execute different instructions at the same time. In task parallelism, different computers or processors handle different jobs simultaneously, which can speed up complex projects with multiple distinct tasks.

Chapter 2

Scaling



2.1 Why Parallelization Matters

The following is an example code for the brightness increase of 1_000_000 Images:

```
using Base.Threads, Images

image_paths = ["image1.jpg", "image2.jpg", ..., "image1_000_000.jpg"]

brightness_increase = 50 # Define your brightness increase

@threads for path in image_paths
    image = load_image(path) # Load the image
    adjusted_image = clamp.(image .+ brightness_increase, 0, 255)
    ↪ # Adjust brightness
    save_image(adjusted_image, "adjusted_$path") # Save the adjust_
    ↪ ed image
end
```

The code comprises commands for:

- **Parallelism:** Distributes image processing across threads, making 1,000,000 images as fast to process as one, assuming sufficient resources.
- **Vectorized Computations:** Modern processors use vectorized instructions (SIMD) to operate on multiple data points simultaneously within a single instruction.

Computational Complexity: Turning an $O(n)$ operation into an effective $O(1)$ operation (assuming perfect parallelization zero overhead).

- **Perfect Parallelization is Impractical:** Real-world scenarios often have dependencies between loop iterations or require synchronization, making perfect parallelization difficult to achieve.
- **Overhead Always Exists (Almost):** Even the most optimized systems have some overhead associated with thread management. While it might be minimal, it's not truly zero.

2.2 Scalability in HPC systems

Scalability in **High Performance Computing (HPC)** systems refers to the ability of a system to effectively utilize increasing numbers of processors or compute resources to perform a task. A scalable HPC system is able to handle growing workloads by adding more resources, such as CPUs, GPUs, or memory, while maintaining efficiency and performance.

Two key aspects of scalability are:

- **Horizontal scalability:** This is the capability to increase performance by **adding more nodes** or machines to the system. It implies a distributed architecture where each node functions as a part of a larger coordinated system.
- **Vertical scalability:** This involves increasing the capacity of an existing machine, such as **adding more CPUs**, memory, or storage to handle larger workloads.

A scalable HPC system should display improved performance with minimal overhead as resources are added. The main measures for this are efficiency, speedup, and throughput.

Example:

Assume an HPC system can execute a task in **20 hours** using **5 nodes**. If scalability is perfect, when we increase the nodes from **5** to **10**, the time to complete the same task should be reduced ideally to **10 hours**. However, due to various overheads and inefficiencies, the time might only reduce to **12 hours**, which represents good but imperfect scalability.

2.3 Speedup

Speedup in the context of HPC is a metric used to quantify the performance improvement of a parallel system compared to a serial one. It is typically expressed as:

$$\text{Speedup} = \frac{\text{Execution Time on Single Processor}}{\text{Execution Time on Multiple Processors}}$$

An **ideal speedup** is linear, meaning the speedup is **equal to the number of processors**. This, however, is rarely the case due to **overheads** inherent in parallel systems.

Example:

Let's say a computational task runs for **80 hours** on a single processor. Running the same task on a parallel system with **4 processors** takes **22 hours**. Thus, the speedup is:

$$\text{Speedup} = \frac{80}{22} \approx 3.64 < 4$$

This represents a **sub-linear speedup**, indicating that there are likely some inefficiencies in the parallel processing.

2.4 Parallelization Efficiency

Parallelization efficiency, also known as **parallel efficiency**, is a measure of how efficiently a computational task runs in parallel compared to serially. It is defined as the ratio of the speedup achieved to the number of processors used:

$$\text{Parallelization Efficiency} = \frac{\text{Speedup}}{\text{Number of Processors}}$$

Values of parallelization efficiency range from **0** (no benefit from parallelization) to **1** (perfect linear speedup). In reality, efficiencies are less than **1** due to **overheads like communication between processors, synchronization, and non-uniform memory access delays**.

Example:

If a task takes 100 hours to complete on a single processor and 15 hours to complete on 8 processors, the speedup is:

$$\text{Speedup} = \frac{100}{15} \approx 6.67$$

The parallelization efficiency would be:

$$\text{Parallelization Efficiency} = \frac{6.67}{8} \approx 0.83$$

Here, an efficiency of **0.83** means the system is utilizing the parallel processors fairly efficiently, but there is still some room for improvement.

2.5 Scaling Tests

Scaling tests in HPC are experiments designed to assess the **scalability of a system**. Two common types of scaling tests are:

- **Strong Scaling Test:** Measures the system's ability to reduce the time to solve a **fixed-size problem as more resources are added**.
- **Weak Scaling Test:** Measures how the solution time changes when **both the problem size and the system size increase proportionally**.

These tests are crucial for understanding the performance characteristics of an HPC system and predict its behavior with different workloads and configurations.

2.6 Strong Scaling

Strong scaling measures the system's capacity to **speed up a fixed-size problem by applying more computational resources**. The aim is to reduce the total execution time. It is defined as:

$$\text{Strong Scaling} = \frac{\text{Execution Time with Single Processor}}{\text{Execution Time with Multiple Processors}}$$

Strong scaling is often limited by the portion of the program that cannot be parallelized, which affects the overall speedup.

Example: If the time to process a dataset is **120** minutes on one core, and it's **40** minutes on **4** cores, then the strong scaling achieved is:

$$\text{Strong Scaling} = \frac{120}{40} = 3$$

This indicates a threefold speedup, which is less than the ideal factor of **4**, and suggests the presence of some non-parallelizable parts or overheads.

Strong scaling and speedup are both related to performance improvement in parallel computing, but they have distinct focuses:

- **Speedup:** Speedup is a more **general term** that refers to how much **faster a parallel program executes** compared to its sequential counterpart.
- **Strong Scaling:** Strong scaling specifically examines how well a system can reduce the execution time for a **fixed-size problem** by adding more processing power.

2.7 Amdahl's Law

Amdahl's Law provides a **theoretical maximum speedup** that can be achieved using parallel processing. It takes into account the **fraction of a program that is serial and cannot be parallelized**. The law is expressed as:

$$\text{Speedup} = \frac{1}{(1 - P) + \frac{P}{N}}$$

where P is the **parallelizable** portion of the task, and N is the number of **processors**.

Example: If 90% of a program can be parallelized ($P = 0.9$), and it's run on **10 processors**, the theoretical maximum speedup according to Amdahl's law would be:

$$\text{Speedup} = \frac{1}{(1 - 0.9) + \frac{0.9}{10}} = \frac{1}{0.1 + 0.09} = \frac{1}{0.19} \approx 5.26$$

According to Amdahl's Law, as N approaches infinity, the speedup approaches $\frac{1}{1-P}$, meaning the maximum speedup is governed by the serial fraction of the task. This can be mathematically represented as:

$$\lim_{N \rightarrow \infty} \text{Speedup} = \lim_{N \rightarrow \infty} \frac{1}{(1 - P) + \frac{P}{N}} = \frac{1}{1 - P}$$

where P is the parallelizable portion of the task.

This law is a critical consideration in the design and **optimization of algorithms for parallel processing**, suggesting that efforts to **reduce the serial portion of programs** can significantly enhance performance gains from parallelization.

2.8 Weak Scaling

Weak scaling is a technique employed in High-Performance Computing (HPC) to utilize additional processing power for handling proportionally larger problems. It maintains a constant workload per processor while **simultaneously increasing both the number of processors (P) and the problem size**. This approach is particularly beneficial **for applications that exhibit**:

- **Memory-boundedness**: These applications require more memory than a single node can provide. Weak scaling allows distribution across multiple nodes, effectively increasing available memory.
- **Inherent parallelism**: These applications are well-suited for breaking down into independent tasks that can be executed concurrently on multiple processors.

While there isn't a single, universally accepted formula for weak scaling, one common metric stems from **Gustafson's Law**.

The available memory and its bandwidth play a critical role in determining the actual scalability and performance. **RAM is not included in Gustafson's; however, it is important to consider** it alongside computational resources when planning for scaling, as memory constraints can become **bottlenecks** even if the computational resources would allow for further scaling.

2.9 Gustafson's Law

Gustafson's Law addresses a key limitation of Amdahl's Law by considering the **scaling of problem size with the number of processors** in a parallel computing environment. It suggests that as we increase the number of processors, we can also increase the problem size to fully utilize the additional computing power, making it a more realistic model for many practical applications of parallel computing.

The law is formalized by the formula:

$$S = (1 - \alpha) + \alpha P \quad (2.1)$$

where:

- S is the theoretical speedup of the execution of the whole task;
- α ($0 \leq \alpha \leq 1$) is the **proportion** of the task that can benefit from **parallelization** (i.e., the parallel portion);
- P is the number of **processors**;
- $(1 - \alpha)$ represents the proportion of the task that must be executed **sequentially** and does not benefit from parallelization.

In essence, Amdahl's Law sets a theoretical limit for a fixed task, while Gustafson's Law explores how we can push that limit by adapting the workload for parallel processing. **Amdahl's Law concerns strong scaling, while Gustafson's Law regards weak.**

Consider a scenario where a task can be divided into a part that benefits from parallelization and a part that does not. Let's say **70% of the task can be parallelized, while the remaining 30% must be executed sequentially**. If we use **8 processors**, we can apply Gustafson's Law to calculate the theoretical speedup.

Given:

- $\alpha = 0.7$ (70% of the task is parallelizable),
- $P = 8$ processors.

Substituting the given values into Gustafson's Law:

$$S = (1 - 0.7) + 0.7 \times 8 \quad (2.2)$$

$$S = 0.3 + 5.6 = 5.9 \quad (2.3)$$

Thus, with 8 processors and 70% of the task being parallelizable, the theoretical speedup of the execution of the whole task is **5.9 times faster** than executing the task on a single processor.

This law provides a theoretical model for understanding how computational work can be scaled with the addition of processors, under the assumption that **increasing the number of processors allows for a proportional increase in the problem size** that can be handled efficiently.

2.10 Performance Metrics in HPC

There are several metrics to evaluate HPC performance:

FLOPS (Floating Point Operations Per Second): A measure of a computer's performance, especially in fields of scientific computations that require floating-point calculations.

- **Latency:** The time taken for a message to travel from one node to another. E.g.: 5 milliseconds between nodes.
- **Bandwidth:** The rate at which data can be transferred over the network. E.g.: 100 Gbps in a high-speed network.

- **Scalability:** The ability of the HPC system to maintain performance as more computational resources are added.

Operations	Name	Abbreviation
1	FLOPS	FLOPS
10^3	Kilo FLOPS	KFLOPS
10^6	Mega FLOPS	MFLOPS
10^9	Giga FLOPS	GFLOPS
10^{12}	Tera FLOPS	TFLOPS
10^{15}	Peta FLOPS	PFLOPS
10^{18}	Exa FLOPS	EFLOPS

Table 2.1: Magnitudes of FLOPS (Floating Point Operations Per Second)

FLOPS are widely used to assess state-of-the-art Supercomputers.

Chapter 3

History of HPC

The evolution of HPC is characterized by significant milestones that mark the technological advancements over the years.

3.1 The Early Days: 1960s

The concept of HPC can be traced back to the **supercomputers** of the 1960s. The first supercomputer is considered the **CDC 6600**, designed by **Seymour Cray** at Control Data Corporation. It was able to perform three millions of floating-point operations per second (**3 MFLOPS**), setting the standard for performance at the time.

3.2 Vector Processors and the Rise of Cray: 1970s-1980s

In the 1970s and 1980s, HPC progressed with the introduction of **vector processors**, which could perform a single operation on a large set of data simultaneously (**SIMD**). Seymour Cray continued to be a pioneer in this area, developing the **Cray-1 (160 MFLOPS)**, **Cray X-MP (200 MFLOPS)** and **Cray-2 systems (1.9 GFLOPS)**, with Cray-2 becoming the fastest machine in the world in 1985.

3.3 Parallel Processing: 1990s

With the rise of **parallel processing** in the 1990s, HPC systems started to be built using multiple processors working in parallel to solve a problem. This period saw the development of the **Massively Parallel Processors (MPPs)** such as the **Thinking Machines CM-5**, which featured hundreds or even thousands of processors interconnected in various topologies to achieve high computational speed. A CM-5 with **1024 processors** could achieve a peak performance of approximately **131 GFLOPS (1993 TOP500 list)**.

MPPs relied on expensive, **custom-designed hardware** and specialized interconnects for maximum performance, while **Beowulf clusters** used **commodity PCs** and standard **networking** for cost-efficiency and scalability. The **Beowulf concept** has been tremendously influential in the development of **modern supercomputers**.

3.4 The Advent of Clustering: Late 1990s and 2000s

The late 1990s and 2000s witnessed the increased use of **computer clusters** for HPC. Clusters are groups of loosely or tightly connected computers that work together so that, in many respects, they can be viewed as a single system. The Beowulf Project is an example of a **Beowulf cluster**, which uses **inexpensive, commodity hardware** for high-performance computations, therefore democratizing access to HPC by **reducing costs**. The first Beowulf cluster (**1994**) comprised **16 i486 DX4 processors** and reached **500 MFLOPS**.

The performance of a Beowulf cluster can vary widely because Beowulf clusters are rather **a concept for building high-performance parallel computing systems** using standard, off-the-shelf components. Hence, gradually, their performance started increasing to **TFLOPS** and beyond.

Taking an Intel Core i9 processor with 10 cores, a conservative clock speed of 3.0 GHz, and assuming it can perform 16 FLOPS per cycle per core, the theoretical peak performance would be:

$$10 \text{ cores} \times 3.0 \text{ GHz} \times 16 \frac{\text{FLOP}}{\text{cycle core}} = 480 \text{ GFLOPS}$$

3.5 Petaflops and Beyond: 2010s

By the 2010s, the HPC field had advanced into the **petaflop** era, where systems could perform 10^{15} floating-point operations per second. An example is the **Cray's XT5 "Jaguar"** system at the Oak Ridge National Laboratory in the United States, which in **2009** was the fastest in the world (**1.75 petaflops**). In this era, accelerators such as General-Purpose Graphics Processing Units (**GPGPUs**) and co-processors, like those from **Intel's Xeon Phi** line, became integral to achieving high performance.

3.6 Exascale Computing: The Next Frontier

Currently, the HPC community is on the brink of reaching **exascale** computing, with systems capable of performing at least 10^{18} operations per second. This new frontier promises **to facilitate innovations** across multiple disciplines, from climate modeling and precision medicine to artificial intelligence and the analysis of massive datasets for scientific research. The race to build the first exascale computer is a testament to the continual evolution of HPC and its key role in science and technology.

These milestones not only reflect increases in raw computational power but **also** signify **advancements in algorithms, data storage, network communication, and parallel software development**. HPC has evolved from a tool for a select few scientific applications to **a fundamental resource for a broad range of disciplines**, influencing both academic research and industrial innovation. **This aligns with the scope of the EuroCC project.**

3.7 Moore's Law

A **transistor** is a semiconductor device used to amplify or switch electronic signals and electrical power. It is one of the basic building blocks of modern electronic devices. In essence, transistors can be understood **as a type of switch** that controls the flow of electricity in a circuit.

An **integrated circuit** (IC), sometimes called a **microchip**, is a semiconductor wafer on which thousands or millions of tiny resistors, capacitors, and transistors are fabricated. An integrated circuit can function as an amplifier, oscillator, timer, microprocessor, or even computer memory.

Gordon E. Moore, a co-founder of the semiconductor company **Intel**, is the person behind the formulation of Moore's Law.

In 1965, Moore observed that the number of transistors on integrated circuits had doubled every year since their invention and predicted that this trend would continue into the foreseeable future.

Moore's initial observation was made during a time when the integrated circuit industry was in its infancy and technological advances were rapid. His prediction was first articulated in an Electronics Magazine article titled "Cramming more components onto integrated circuits" (1965). In his prediction's initial formulation, Moore suggested that the number of transistors on a microchip would double every year.

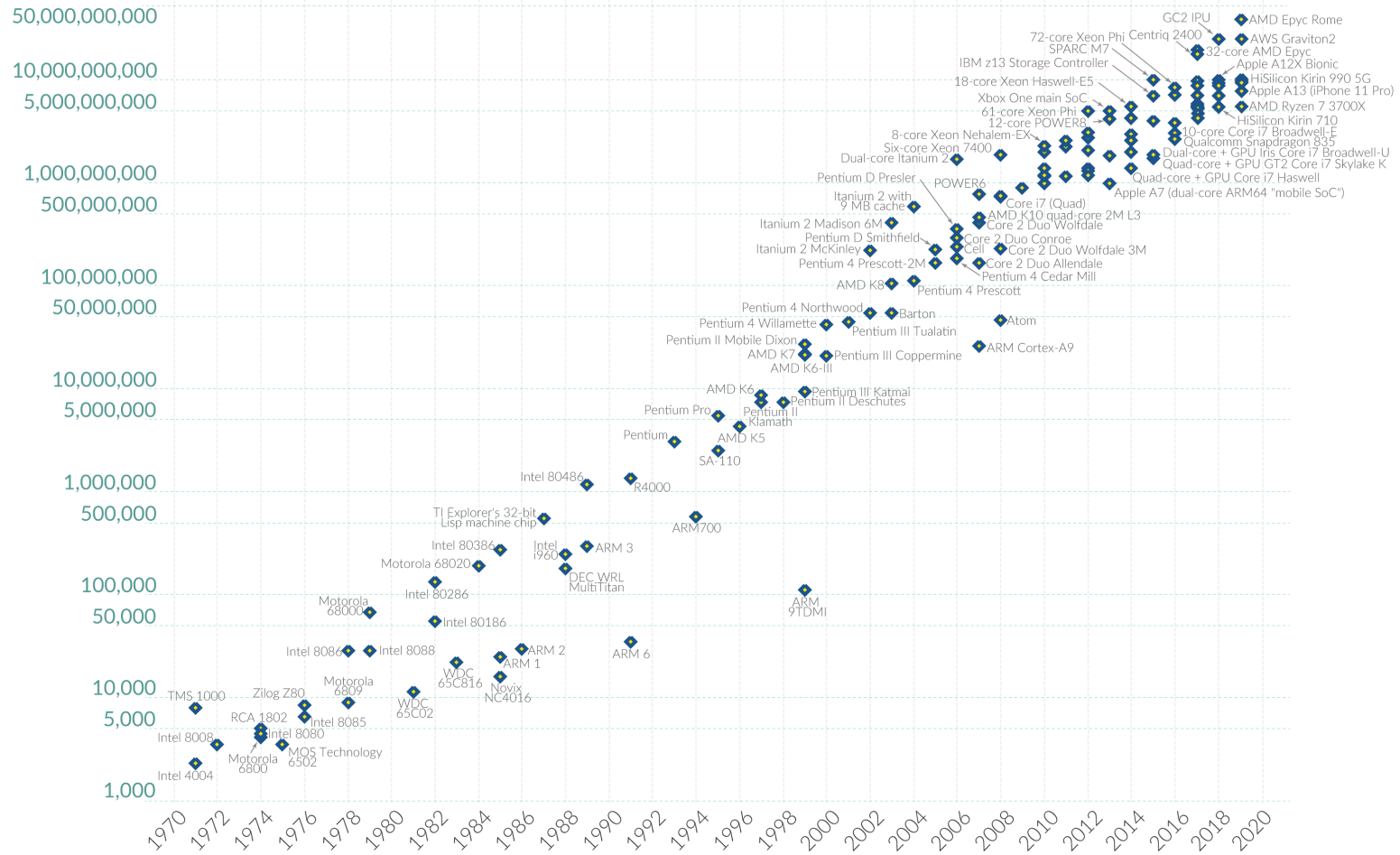
A decade later, in **1975**, based on the changing pace of technology, he revised his projection, estimating **a doubling approximately every two years**.

The significance of Moore's Law is that it has been used as a **guideline for setting targets in the semiconductor industry**, leading to the rapid evolution of electronic devices, which have become faster, cheaper, and more efficient with time. However, it is important to note that **Moore's Law is a projection, not a physical or natural law**, and as we approach the limits of physics and economics, the pace of advancement predicted by Moore's Law has slowed. **Despite the periodic predictions of its demise, Moore's Law has persisted** as a driving force of technological and social change.

Moore's Law: The number of transistors on microchips doubles every two years

Moore's law describes the empirical regularity that the number of transistors on integrated circuits doubles approximately every two years. This advancement is important for other aspects of technological progress in computing – such as processing speed or the price of computers.

Transistor count



Data source: Wikipedia ([wikipedia.org/wiki/Transistor_count](https://en.wikipedia.org/wiki/Transistor_count))
OurWorldInData.org – Research and data to make progress against the world's largest problems. Licensed under CC-BY by the authors Hannah Ritchie and Max Roser.

Hannah Ritchie and Max Roser, <https://ourworldindata.org/uploads/2020/11/Transistor-Count-over-time.png> <https://creativecommons.org/licenses/by/4.0>

Moore's Law **doesn't directly predict an increase in FLOPS, but it does have a strong correlation.** With more transistors on a chip, chip designers can dedicate more resources to floating-point operations, leading to a higher theoretical FLOPS rating. However, factors like chip architecture and memory bandwidth also play a role in determining real-world performance.

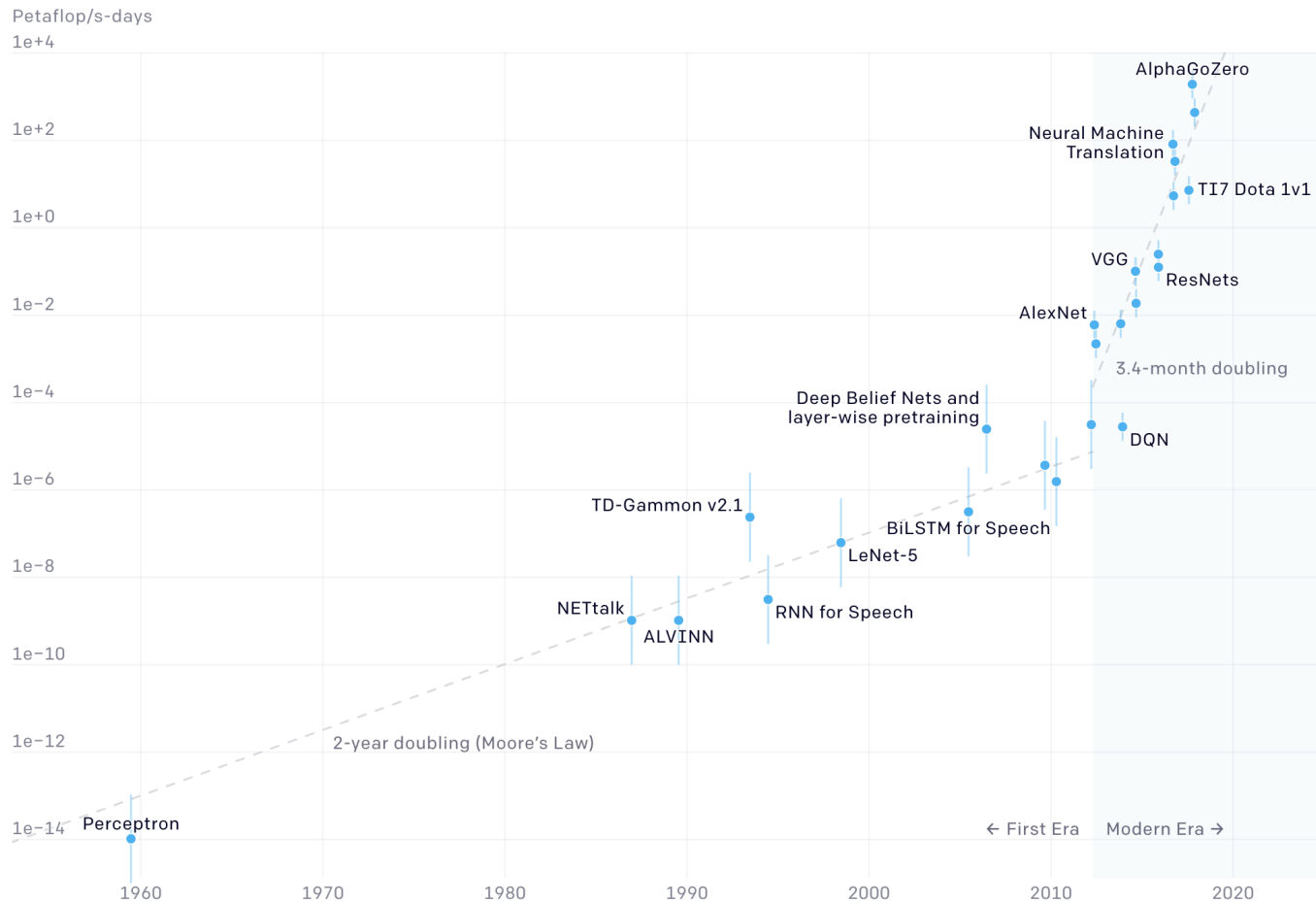
- **More Transistors:** Moore's Law allows for more transistors on a chip.
- **Dedicated Processing Units:** These additional transistors can be used to create specialized processing units optimized for floating-point calculations.
- **Increased FLOPS:** With more dedicated resources, the chip can potentially achieve a higher FLOPS rating.

3.8 AI and compute

Since 2102 we observe a **3.4-month doubling** in computing power used to train AI models.

<https://openai.com/research/ai-and-compute>

Two Distinct Eras of Compute Usage in Training AI Systems



Chapter 4

Programming Models in HPC

HPC applications are often developed using programming models that support parallelism:

- **OpenMP (Open Multi-Processing):** An application programming interface (API) that supports multi-platform shared-memory multiprocessing programming.
- **MPI (Message Passing Interface):** Used for programming parallel computers. It involves processes sending and receiving messages to achieve parallelism.
- **GPGPU (General-Purpose computing on Graphics Processing Units):** Utilizes GPUs to perform computation in applications traditionally handled by the CPU.

These models come with their own set of libraries and compilers that are designed to optimize performance by taking full advantage of the hardware capabilities present within HPC environments.

4.1 OpenMP (Open Multi-Processing)

OpenMP is an application programming interface (API) that supports multi-platform **shared-memory multiprocessing** programming in C, C++, and Fortran. It is used to direct multi-threaded shared-memory parallelism.

Key Concepts:

- **Shared Memory:** In a shared memory system, all processors have access to a common memory space.
- **Threads:** A thread is the smallest sequence of programmed instructions that can be managed independently by the scheduler.
- **Parallel Region:** A section of code that is executed by multiple threads simultaneously.
- **Directives:** Compiler directives are used to specify parallel regions and work sharing among threads.
- **Synchronization:** Mechanisms to coordinate the execution of multiple threads, such as barriers, locks and atomic operations.

Simple Example:

```
#include <omp.h>
#include <stdio.h>

int main() {
    #pragma omp parallel
    {
        printf("Hello from thread %d\n", omp_get_thread_num());
    }
    return 0;
}
```

In this example, the `#pragma omp parallel` directive indicates the start of a parallel region, where the enclosed code block is executed by each thread concurrently. The function `omp_get_thread_num()` returns the identifier of the current thread.

4.2 MPI (Message Passing Interface)

MPI is a standardized and portable message-passing system designed to function on a variety of parallel computing architectures. The API permits **point-to-point and collective communication among computing nodes in a distributed-memory environment**.

Key Concepts:

- **Distributed Memory:** Each processor has its own private memory. Processors require a **mechanism** (like MPI) to **communicate** data.
- **Processes:** An independent unit of program execution that runs in its **own memory** space.
- **Communicator:** An MPI object that defines a group of processes that can communicate with each other.
- **Point-to-Point Communication:** Sending a message from one process to another.
- **Collective Communication:** Performing operations involving a group of processes, such as broadcasting, gathering, and scatter.

Simple Example:

```
#include <mpi.h>
#include <stdio.h>

int main(int argc, char *argv[]) {
    MPI_Init(&argc, &argv);

    int rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    printf("Hello from process %d\n", rank);

    MPI_Finalize();
    return 0;
}
```

In the above MPI example, **MPI_Init** initializes the MPI environment, **MPI_Comm_rank** gets the rank of the current process, and **MPI_Finalize** is used to clean up the MPI environment.

4.3 GPUs (Graphics Processing Units)

GPUs are specialized processors that rapidly manipulate and alter memory to accelerate the creation of images in a frame buffer intended for output to a display. Due to their **highly parallel structure**, GPUs are also used for more general computing tasks requiring high throughput.

Key Concepts:

- **SIMD (Single Instruction, Multiple Data):** GPUs follow the SIMD paradigm, executing one instruction across many data elements in parallel.
- **Stream Processors:** The basic compute units in a GPU that carry out computations in parallel.
- **Kernel:** In the context of GPUs, a kernel is a **function** that is executed on the GPU to perform **data-parallel** computations.
- **Threads and Blocks:** The execution configuration of GPU kernels, where threads are the smallest units of execution, and **blocks are groups of threads** that execute on the GPU.

4.4 CUDA (Compute Unified Device Architecture)

CUDA is a **parallel computing platform** and programming model developed by **NVIDIA** for general computing on its own GPUs. With CUDA, developers can dramatically speed up computing applications by harnessing the power of GPUs.

Key Concepts:

- **Device vs. Host:** In CUDA, the term "**device**" refers to the **GPU**, and "**host**" refers to the **CPU**.
- **Memory Model:** CUDA features different memory spaces, such as global, shared, constant, and local memory, each with its own scope and lifetime.
- **Threads Hierarchy:** **Threads** are organized into **blocks**, and blocks are organized into **grids**.
- **CUDA Kernels:** A **piece of code** that runs on the GPU. Kernels are defined using the `__global__` declaration specifier and are called from the host code.

Simple Example:

```
__global__ void add(int *a, int *b, int *c, int size) {  
    int index = threadIdx.x + blockIdx.x * blockDim.x;  
    if (index < size) {  
        c[index] = a[index] + b[index];  
    }  
}
```

```

int main() {
    // Initialize a, b, c arrays on the host and copy them to device.
    // Define total number of elements 'n' and so on.
    ...
    int *d_a, *d_b, *d_c; // Device pointers
    // Allocate memory for device copies of a, b, c
    // Handle errors for each cudaMalloc and cudaMemcpy as needed
    ...
    // Launch add() kernel on GPU with N blocks
    add<<<N, 1>>>(d_a, d_b, d_c, n);
    // Copy result back to host
    // Handle errors for cudaMemcpy as needed
    ...
    // Cleanup
    ...
    return 0;
}

```

This CUDA C code snippet outlines how to define a kernel function for parallel execution on a device (GPU). It performs vector addition across GPU threads. The main function handles necessary device memory allocations, data transfer between the host and device, the execution of the kernel, and eventually the cleanup. This general pattern forms the basis of computing with NVIDIA GPUs using CUDA.

Chapter 5

State of the art machines

5.1 The Top 500 list

- The Top 500 list is a ranking of the **world's 500 most powerful non-distributed computer systems**. The list is compiled twice a year.
- **Performance** of the supercomputers on the Top 500 list **is measured using the LINPACK Benchmark**. This benchmark tests the system's ability to solve a dense system of linear equations, providing a measure of the computer's floating-point rate of execution.
- Top 5 as of the Latest List:
 - **Frontier (USA)**: The only exascale machine on the list with an HPL score of $1.194 EFlop/s$.
 - **Aurora (USA)**: Entered the list at No. 2 with an HPL score of $585.34 PFlop/s$, based on half of the final planned system.
 - **Eagle (Microsoft Azure Cloud, USA)**: Achieved No. 3 spot with an HPL score of $561.2 PFlop/s$, the highest rank a cloud system has ever achieved.
 - **Fugaku (Japan)**: Moved to No. 4 with an HPL score of $442.01 PFlop/s$.
 - **LUMI (Finland)**: Positioned at No. 5 with an HPL score of $379.70 PFlop/s$.

5.1.1 Exponential Growth

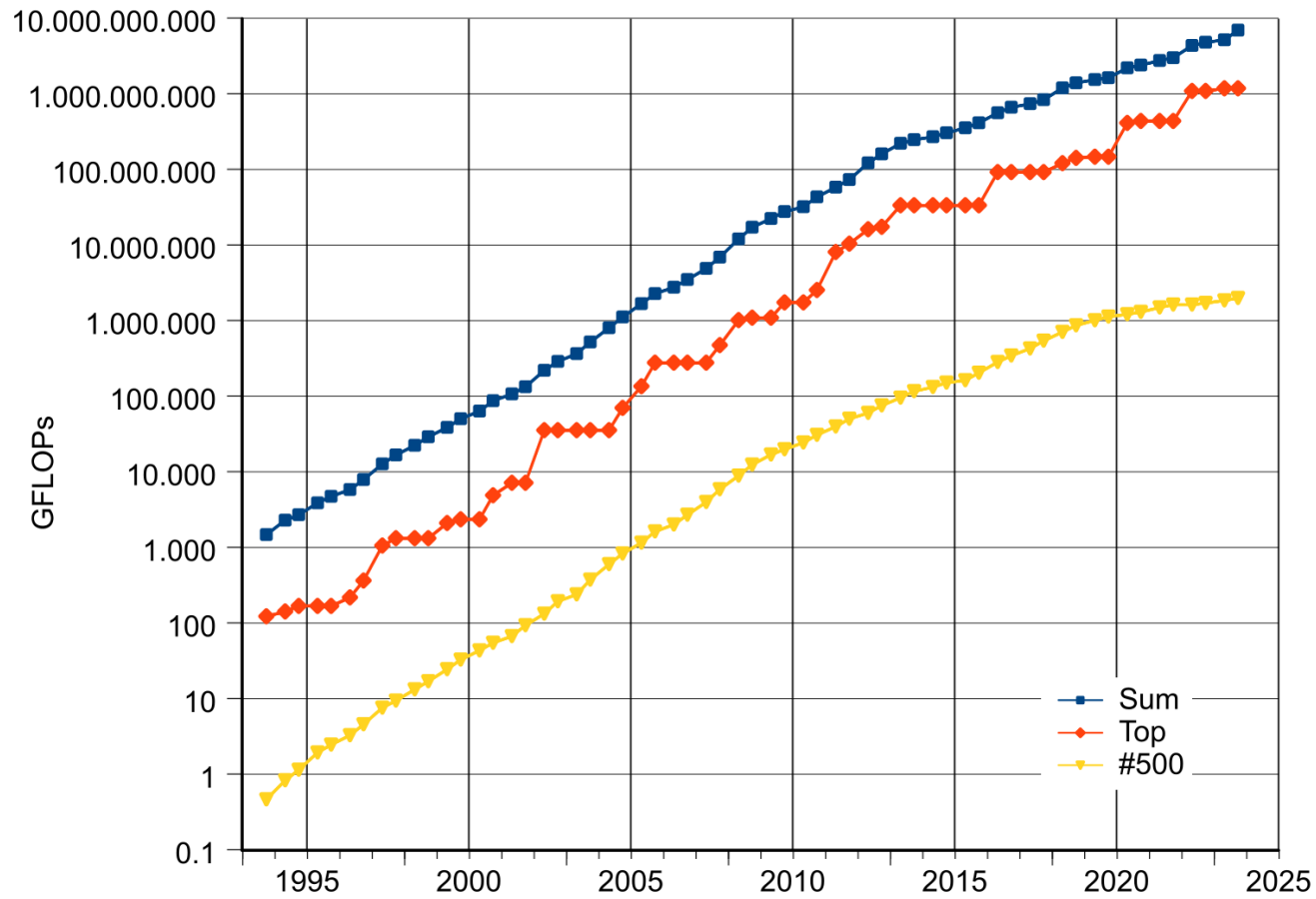


Figure 5.1: AI.Graphic - Own work

<https://creativecommons.org/licenses/by-sa/3.0/>

<https://en.wikipedia.org/wiki/TOP500#/media/File:Supercomputers-history.svg>

5.2 Top 8 European Supercomputers

The EuroCC project, Europe aims to *democratize access to supercomputing resources*, enabling SMEs across the continent to leverage AI, high-performance data analytics, and simulations, thereby fostering innovation, competitiveness, and growth in the European digital economy.

It utilizes computing resources at the *National and European Level*, exploiting the **EuroHPC Joint Undertaking** Supercomputers: https://eurohpc-ju.europa.eu/supercomputers/our-supercomputers_en

1. LUMI (CSC, Finland)

- LUMI-C: 1536 nodes, 128 cores/node, 256-1024 GB RAM/node
- GPU: 2560 nodes, 64 cores/node, 4 GPUs, 128 GB GPU-RAM
- Visualization: 64 nodes, 1 GPU, 48 GB GPU-RAM
- Peak Performance: 550 petaflops
- URL: <https://www.lumi-supercomputer.eu/lumis-full-system-architecture-revealed/>

2. Leonardo (Cineca, Italy)

- Booster Module: 3456 nodes, 32 cores/node, 512 GB RAM/node, 4 GPUs, 64 GB GPU-RAM
- Data Centric Module: 1536 nodes, 112 cores/node, 512 GB RAM/node

- Peak Performance: 323.4 petaflops
- URL: <https://leonardo-supercomputer.cineca.eu/hpc-system/>

3. MareNostrum 5 (Barcelona Supercomputing Center, Spain)

- General Purpose Partition: 6408 nodes, 112 cores/node, 256 GB RAM/node
- Accelerated Partition: 1120 nodes, 64 cores/node, 512 GB RAM/node, 4 GPUs, 64 GB GPU-RAM
- Peak Performance: 314 petaflops
- URL: <https://www.bsc.es/innovation-and-services/marenostrum/marenostrum-5>

4. MeluXina (LuxProvide, Luxembourg)

- Cluster: 573 nodes, 128 cores/node, 512 GB RAM/node
- Accelerator-GPU: 200 nodes, 64 cores/node, 512 GB RAM/node, 4 GPUs, 40 GB GPU-RAM
- Large memory: 20 nodes, 128 cores/node, 4096 GB RAM/node
- Peak Performance: 18.29 petaflops
- URL: <https://docs.lxp.lu/system/overview/>

5. Karolina (IT4I, Czech Republic)

- CPU: 828 nodes, 128 cores/node, 256-24000 GB RAM/node
- GPU: 72 nodes, 8 GPUs, 40 GB GPU-RAM
- Peak Performance: 15.69 petaflops

- URL: <https://www.it4i.cz/en/infrastructure/karolina>

6. Vega (IZUM, Slovenia)

- GPU partition: 60 nodes, 128 cores/node, 512 GB RAM/node, 4 GPUs, 40 GB GPU-RAM
- CPU node Standard: 768 nodes, 128 cores/node, 256 GB RAM/node
- CPU node Large Memory: 192 nodes, 128 cores/node, 1000 GB RAM/node
- Peak Performance: 10.05 petaflops
- URL: <https://doc.vega.izum.si/architecture/>

7. Deucalion (Guimarães, Portugal)

- ARM cluster: 1632 nodes, 48 cores/node
- X86 cluster: 500 nodes, 48+ cores/node
- Accelerated partition: 33 nodes
- Peak Performance: 10 petaflops
- URL: <https://macc.fcn.pt/resources#deucalion>

8. Discoverer (Sofia Tech Park, Bulgaria)

- CPU: 1128 nodes, 128 cores/node, 256 GB RAM/node
- CPU-Fat: 18 nodes, 128 cores/node, 1000 GB RAM/node
- Peak Performance: 5.94 petaflops
- URL: https://docs.discoverer.bg/resource__overview.html



LUMI
FINLAND



LEONARDO
ITALY



MELUXINA
LUXEMBOURG



KAROLINA
CZECH REPUBLIC



DISCOVERER
BULGARIA



VEGA
SLOVENIA



DEUCALIO
PORTUGAL



MARENOSTRUM 5
SPAIN

Chapter 6

EuroCC Services

6.1 Greece's HPC Competence Center

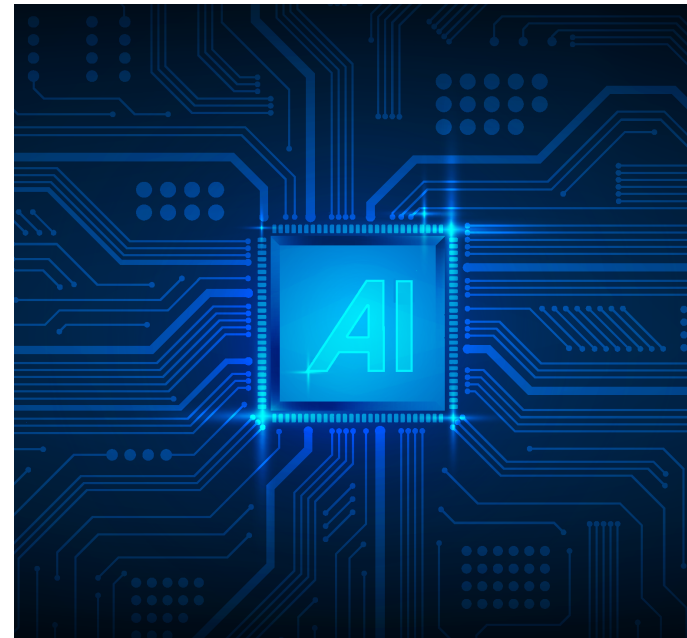
Enhancing innovation capacity in Business, Industry and Science by utilizing advanced High Performance Computing services.



EuroCC@Greece

6.2 Artificial Intelligence (AI)

- **Access to Cutting-edge Technology:** *Research organizations and Enterprises* gain access to advanced AI computing resources, enabling them to develop and refine AI models with efficiency and unprecedented speed.
- **Innovation and Competitive Edge:** Leveraging AI through supercomputers helps organizations *innovate, creating new products and services*, thus staying competitive in a rapidly evolving digital landscape.
- **Large Language Models:** Enable *fine-tuning for custom applications*, enhancing precision and relevance in industry-specific contexts, driving targeted outcomes and efficiencies.
- **AI Skill Development:** Offers *training and skill development* in AI technologies for organizations and individuals, helping them to build in-house expertise and apply AI solutions effectively.



6.3 High-Performance Data Analytics (HPDA)

- **Data Processing at Scale:** HPDA equips SMEs with the capability to process and analyze large datasets, overcoming the limitations often faced due to smaller infrastructures
- **Large Image Analysis for Climate Studies:** HPDA enables the analysis of large and complex image datasets (such as satellite imagery) for SMEs involved in environmental research, climate studies, or related fields.
- **Healthcare Applications:** accelerate the analysis of medical images or genetic data, leading to faster diagnosis and personalized treatments.
- **Retail Markets:** process customer data to tailor marketing strategies or improve personalized customer experience.



6.4 High-performance Simulations

- **Engineering Simulations:** Utilizing methods like finite elements to analyze and design complex structures, mechanisms, or systems in various engineering fields.
- **Molecular Simulations:** Modeling and understanding the behavior of molecular systems, including the discovery of new materials. By simulating the interactions at the atomic or molecular level, researchers can predict the properties of materials before they are synthesized, leading to advances in materials science.
- **Drug Discovery:** By simulating the interaction between drug molecules and biological targets, researchers can identify promising drug candidates, significantly speeding up the drug discovery process and reducing the need for early-stage laboratory experiments.
- **Physics Simulations:** Pivotal for understanding complex physical phenomena into areas like astrophysics, quantum mechanics, and particle physics, enhancing our understanding of the universe's fundamental laws.



6.5 EuroCC Success Stories

The EuroCC ACCESS success stories showcase a variety of successful experiments conducted within the EuroCC projects, highlighting the business benefits derived from these collaborations.

These experiments involve partners from industry, society, and academia, and cover a wide range of applications such as:

- Transfer and optimization of CFD calculations workflow in HPC environment
- Anomaly Detection in Time Series Data: Gambling prevention using Deep Learning
- Multimodal Prediction of Alexithymia from Physiological and Audio Signals
- Enabling HPC Usage for Expensive ML Tasks on Manufacturing Environments
- Rebar cutting optimisation using a cloud computing environment
- The Estonian COVID-19 Data Portal and KoroGeno-EST
- And many more!

<https://www.eurocc-access.eu/success-stories/>

Chapter 7

Apply for Access at EuroHPC JU

7.1 EuroHPC JU Benchmark Access

The purpose of the EuroHPC JU Benchmark Access calls is to support researchers and HPC application developers by giving them the opportunity **to test or benchmark their applications** on the upcoming/available EuroHPC Pre-exascale and/or Petascale system prior to applying for an Extreme Scale and/or Regular Access. The EuroHPC Benchmark call is designed **for code scalability tests** or for **test of AI applications** and the outcome of which is to be included in the proposal in a future EuroHPC Extreme Scale and Regular Access call. Users receive a **limited number of node hours**; the maximum allocation period is **three months**.

7.2 EuroHPC JU Development Access

The purpose of the EuroHPC JU Development Access calls is to support researchers and HPC application developers by giving them the opportunity to **develop, test and optimise their applications** on the upcoming/available EuroHPC Pre-exascale and/or Petascale system prior to applying for an Extreme Scale and/or Regular Access. The EuroHPC Development call is designed for projects focusing on **code and algorithm development and optimisation**, as well as **development of AI application methods**. This can be in the context of research projects from **academia or industry, or as part of large public or private funded initiatives** as for instance Centres of Excellence or Competence Centres. Users will typically be allocated a **small number of node hours**; the allocation period is **one year** and is renewable up to two times.

7.3 EuroHPC JU Regular Access

The Regular Access mode is designed to serve research domains, industry open R&D and public sector applications that require **large-scale resources** or that require more frequent access to **substantial computing and storage resources**. This access mode distributes resources, mostly from the EuroHPC JU petascale systems.

This Regular Access Call offers three distinctive application tracks:

- **Scientific Access** – Intended for applications from the academia and public research institutes.
- **Industry Access** – Intended for applications with Principal Investigator (PIs) coming from industry.
- **Public Administration Access** – Intended for applications with PIs coming from the public sector.

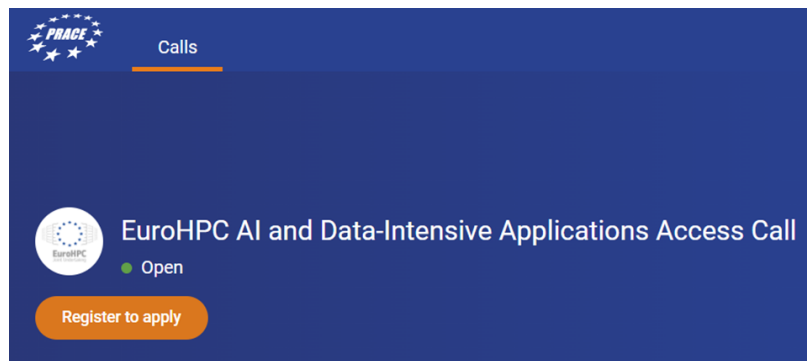
7.4 EuroHPC JU Extreme Access

The Extreme Scale Access Mode call is targeting HPC **applications with high-impact and high-gain innovative research**. This access mode distributes resources, from the EuroHPC pre-exascale systems.

This call offers three distinctive application tracks:

- **Scientific Access** – Intended for applications from the academia and public research institutes.
- **Industry Access** – Intended for applications with Principal Investigator (PIs) coming from industry.
- **Public Administration Access** – Intended for applications with PIs coming from the public sector.

7.5 EuroHPC JU Access Call for AI and Data-Intensive Applications







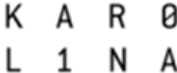

Call Details:

The EuroHPC JU AI and Data-Intensive Applications Access call aims to support **ethical artificial intelligence, machine learn-**

ing, and in general, data-intensive applications, with a particular focus on **foundation models and generative AI** (e.g. large language models).

The call is intended to serve **industry organizations, small to medium enterprises (SMEs), startups, as well as public sector entities**, requiring access to supercomputing resources to perform artificial intelligence and data-intensive activities.

https://eurohpc-ju.europa.eu/eurohpc-ju-access-call-ai-and-data-intensive-applications_en

SYSTEM*	SITE (COUNTRY)	ARCHITECTURE	PARTITION	TOTAL RESOURCES**	FIXED ALLOCATION
 MN5 MARENOSTRUM	BSC (ES)	Atos BullSequana XH3000	MN5 ACC	129 377	32 000
 LEONARDO CINECA	CINECA (IT)	Atos BullSequana XH2000	Leonardo Booster	545 865	50 000
 LUMI	CSC (FI)	HPE Cray EX	LUMI-G	351 455	35 000
 MELUXINA HIGH PERFORMANCE COMPUTING IN LUXEMBOURG	LuxProvide (LU)	Atos BullSequana XH2000	MeluXina GPU	25 000	25 000
 KAROLINA	IT4I VSB-TUO (CZ)	HPE Apollo 2000 Gen10 Plus and HPE Apollo 6500	Karolina GPU	7 500	7 500
 VEGA HPC	IZUM Maribor (SI)	Atos BullSequana XH2000	Vega GPU	7 100	7 100

Partition information

Partition name*

Requested amount of resources (node hours)*

This value is pre-defined

Previous Benchmark or Development Access allocations

Project ID

Please provide an ID in case you have previously obtained data relevant for this proposal via the Benchmark and/or Development Access call

[+ Previous Benchmark or Development Access allocations](#)

7.6 Frequently Asked Questions (FAQ)

— Which organisations are eligible for access to EuroHPC machines?

Any European organisation is eligible for access to perform Open Science research (the results of the work are made available for open access). This includes public and private academic and research institutions, public sector organisations, industrial enterprises and SMEs.

+ What documents are required for access?

— What is the cost?

Currently access is free of charge.

— What are the participation conditions?

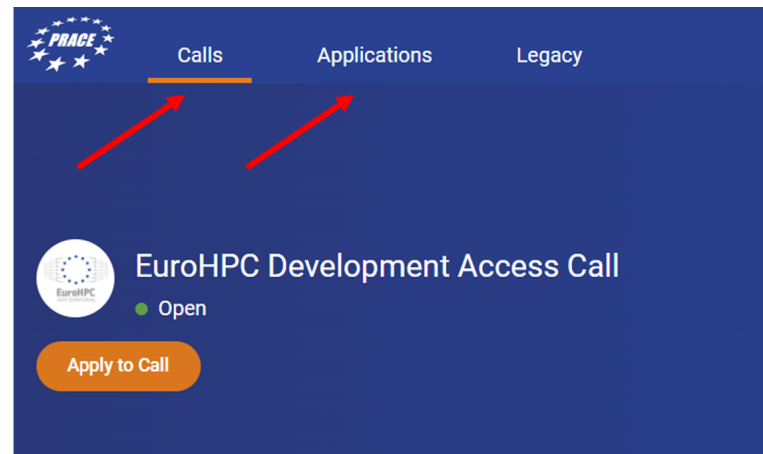
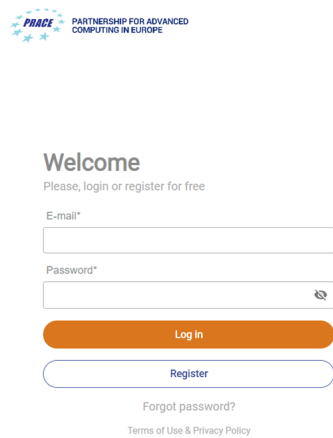
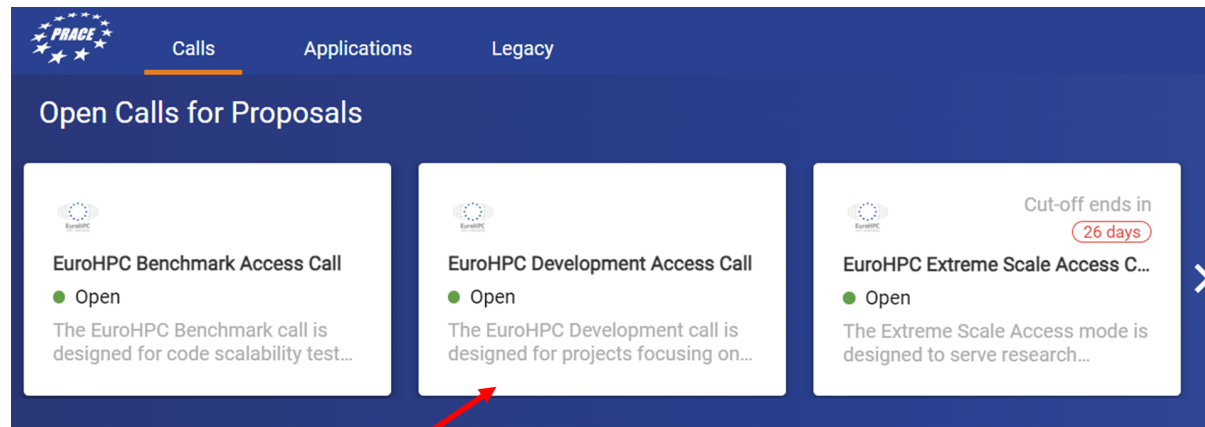
Participation conditions depend on the specific access call that a research group has applied. In general users of EuroHPC systems commit to:

- acknowledge the use of the resources in their related publications,
- contribute to dissemination events,
- produce and submit a report after completion of a resource allocation.

More information on participation conditions can be found in the call's Documents section.



https://eurohpc-ju.europa.eu/access-our-supercomputers/access-policy-and-faq_en


7.7 Indicative Application



<https://pracecalls.eu/>

7.7.1 The project

Calls **Applications** Legacy Applicant 

- Project Application** 
- The Project**
- Principal Investigator
- Contact Person and Team Members Information
- Partitions
- Code Details and Development
- Data Consent

The Project

Project details

Project title*


HPC Support on Machine Learning and Generative AI Algorithms

Project summary (abstract)*


This project aims to develop, test, and provide open-access Machine Learning (ML) and Generative AI algorithms tailored for HPC environments. We aim to boost the efficiency, accessibility, and innovation capacity of HPC users in Greece for a wide range of scientific and engineering applications. By offering open access to cutting-edge algorithms, we aim to empower researchers and developers to optimize HPC workloads and enhance computational efficiency. The initiative will focus on creating and testing scalable and easily integrated ML models and generative AI frameworks that can be readily adopted by the HPC community in Greece. This approach ensures that the benefits of advanced computational techniques are not confined to institutions with significant resources but are available to all researchers and developers.

Explain the **scientific case** of the project for which you intend to use the code(s)*

Through this project, we aim to assist our efforts for open and collaborative HPC computing, fostering innovation and breakthroughs across various disciplines. Access to supercomputing facilities for testing and refining these algorithms is critical. It will allow us to ensure their performance at scale and their applicability across different HPC systems and research domains.


 Save Changes Next

Deadline
01/04/2024 12:00:00 PM


 Documents

Delete Application

7.7.2 Research Fields

Proposal for civilian purposes* 

Is any part of the project confidential?*


Yes No 

Research fields

Research field title*

PE6 Computer Science and Informatics 

Research field sub-title*

PE6_7 Artificial intelligence, intelligent systems, natural language processing 

Research field share (%)*

50

The sum of all research fields should not exceed the total of 100%

Figure 7.1: Enter Caption

Research fields #2

Research field title*

Research field sub-title*

Research field share (%)*

The sum of all research fields should not exceed the total of 100%

Remove + Research fields

AI set of technologies selection

Machine Learning Natural Language Processing Deep Learning

If applicable, please select used AI technologies. This is a multi-select field so you are able to choose more than one option.

7.7.3 Societal impact

Please specify how does your project ensure ethical principles and addresses potential societal impacts associated with the development and deployment of AI technologies*

We prioritize transparency by openly sharing our AI methodologies and limitations. By providing open access to our algorithms, we democratize access to advanced computational resources. Our project serves as a catalyst for innovation across scientific and engineering disciplines. By enhancing computational efficiency, we enable researchers to tackle complex problems, from climate change to healthcare, contributing to societal well-being. We commit to responsible Generative AI deployment, adhering to ethical guidelines.

Submission details

Project duration*

6 months 12 months

Preferable starting date*

01-04-2024



Industry involvement*

As Team Member



Public sector involvement*

As Team Member



7.7.4 CPU Partition

Partitions

Partition name*

MeluXina CPU

Code(s) used*

XGBoost

MPI

Horovod

Pytorch

This field is a multi-text field, for adding another code separate it with a comma

Requested amount of resources (node hours)*

4 000

Average number of processes/threads*

128

Average job memory (total usage over all nodes in GB)*

400

Maximum amount of memory per process/thread (MB)*

10 000

Total amount of data to transfer to/from (GB)*

100

7.7.5 Input / Output

I/O libraries, MPI I/O, netcdf, HDF5 or other approaches*

Custom Python scripts with multiprocessing for parallel data loading and process.

Frequency and size of data output and input*

Data input/output occurs hourly, with inputs averaging 10 GB per batch and outputs around 5 GB, optimized for real-time ML model training and analysis.

Number of files and size of each file in a typical production run*

Around 500 files, each averaging 20 MB, facilitating efficient batch processing for ML training on HPC.

Total storage required (GB)

100

7.7.6 GPU Partition

Partitions #2 

Partition name*

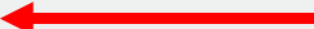
MeluXina GPU 

Code(s) used*

Llama Falcon Mistral

This field is a multi-text field, for adding another code separate it with a comma

Requested amount of resources (node hours)*

800 

Average number of processes/threads*

64

Average job memory (total usage over all nodes in GB)*

800

Maximum amount of memory per process/thread (MB)*

12 500

7.7.7 Code Details

Code Details and Development

This tab should overall include the following: description of main algorithms, how they have been implemented and parallelized, and their main performance bottlenecks and the solutions to the performance issues you have considered. For each code that needs to be optimized, please provide the details below. Codes can be added by clicking on the Add code button.

Development of the code(s) description*

For machine learning we will be using custom codes such as <https://github.com/nbakas/hpml/> for hyperparameter tuning in parallel. For Gen-AI we will be using open access LLMs, such as Mistral, Llama, and Falcon

Code details

Name and version of the code

High Performance Machine Learning Algorithms for Tabular Datasets
<https://github.com/nbakas/hpml/>
Version 1.0

Figure 7.2: Enter Caption

7.7.8 Scalability & Performance

Scalability and performance ←

Describe the scalability of the application and performance of the application*

Large language models (LLMs) to be used, like Mistral, Llama, and Falcon are designed with scalability in mind, allowing them to handle extensive computational loads efficiently. These models can scale across multiple GPUs and nodes in a high-performance computing environment, enabling parallel processing of large datasets and complex computations. E.g. see:
<https://huggingface.co/blog/falcon-180b>
<https://www.truefoundry.com/blog/benchmarking-llama2-falcon-and-mistral>

What is the target for scalability and performance?*

Our target for scalability and performance involves optimizing our application to efficiently utilize HPC resources, aiming to support concurrent processing of large datasets and ensuring rapid response times for machine learning tasks involving open-access LLMs like Mistral, LLaMA, and Falcon.

i.e. what performance is needed to reach the envisaged scientific goals

Any scalability tests prior to the application are useful to highlight here!
The machine should be utilized as optimally as possible!

7.7.9 Optimization

Optimization of the work proposed

Explain how the optimization work proposed will contribute to future Tier-0 projects*

The proposed optimization work enhances computational efficiency, reduces runtime, and scales effectively across HPC resources, directly supporting the ambitious goals of future Tier-0 projects by enabling more complex and larger-scale computations.

Describe the impact of the optimization work proposed - is the code widely used; can it be used for other research projects and in other research fields with minor modifications; would these modifications be easy to add to the main release of the software?*

The optimization work aims to make the code highly versatile and adaptable, allowing for broad use across various research fields with minimal adjustments. These modifications will be made available to the users we support.

Describe the main algorithms and how they have been implemented and parallelized*

LLMs implement a variety of sophisticated algorithms, primarily based on the Transformer architecture, which is renowned for its self-attention mechanism. This architecture allows the models to weigh the importance of different words within the input text, enabling a deeper understanding of the context. They are implemented using frameworks that support parallel processing, such as PyTorch and TensorFlow. These frameworks provide built-in support for data, model, and pipeline parallelism, facilitating the development and training of large models on distributed computing resources.

7.7.10 Performance

Performance

Main performance bottlenecks*

The main performance bottlenecks in training and deploying LLMs typically include memory constraints, data loading and I/O throughput, inter-node communication overheads, and the computational cost of forward and backward passes.

Describe possible solutions you have considered to improve the performance of the project*

Pretrained LLMs already incorporate advanced parallel processing, model compression techniques like quantization and pruning, and efficient data loading and caching mechanisms to optimize performance and address computational and memory constraints.

Describe the application enabling/optimization work that needs to be performed to achieve the target performance*

To achieve target performance, optimization work includes implementing dynamic batching for efficient GPU utilization, fine-tuning parallel processing strategies, optimizing memory allocation, and incorporating adaptive learning rate schedulers.

Which computational performance limitations do you wish to solve with this project?*

This project aims to solve computational performance limitations related to high memory demands, slow model training and inference speeds, and inefficient data handling. Our goal is to enable users to efficiently run and benefit from the latest LLMs like Mistral, LLaMA, and Falcon, making these advanced AI technologies more accessible and practical for diverse research and development purposes.

7.7.11 Data Consent

Data Consent

The EuroHPC JU will process personal data in accordance with Regulation (EU) 2018/1725 on the protection of natural persons with regard to the processing of personal data by the Union institutions, bodies, offices and agencies and on the free movement of such data, and repealing Regulation (EC) No 45/2001 and Decision No 1247/2002/EC.

In case the proposal is awarded, EuroHPC JU would like to publish the Principal Investigator's and Team Members' names and organizations. This may involve sharing this information on our website, social media channels, or in other promotional materials related to the project. Please provide your consent below.*

I consent I do not consent

In order to submit the proposal, you must accept the terms and conditions stated in the Access Policy, hence confirming that you have read and understood the call procedures. The documentation can be found at https://eurohpc-ju.europa.eu/eurohpc-ju-call-proposals-development-access_en*

I accept the terms and conditions stated in the Access Policy

Back

Save Changes

Submit

https://eurohpc-ju.europa.eu/eurohpc-ju-call-proposals-development-access_en

Deadline

01/04/2024 12:00:00 PM

  Documents

Delete Application

Chapter 8

Resources

- EuroCC Sweden: How to apply for access to EuroHPC JU supercomputers <https://www.youtube.com/watch?v=g5jOio006-E>
- ENCCS (NCC Sweden): "How to use the PRACE-calls portal - Application to JU supercomputers" Seminar <https://www.youtube.com/watch?v=N1QqMh7H0mQ>
- NCC Greece: <https://eurocc-greece.gr/how-to-apply-for-access-to-eurohpc-ju-supercomputers/>
- HPC wiki: https://hpc-wiki.info/hpc/HPC_Wiki

Thank you!

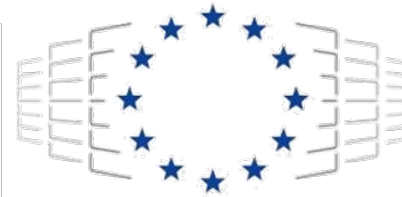
Introduction to High-Performance Computing

Dr Nikos Bakas nibas@grnet.gr

National Infrastructures for Research and Technology - GRNET



Co-funded by
the European Union



EuroHPC
Joint Undertaking



EuroCC@Greece

<https://eurocc-greece.gr/>