191.114 Basics of Parallel Computing

Author:

Max Tiessler

2025-03-26

Contents

1 Introduction

This summary is for the course "Basics of Parallel Computing" (course code: 191.114) at the Vienna University of Technology (TU Wien) and was created for the Summer Semester 2025 in preparation for the exam. The document covers most of the essential topics discussed in the course, focusing on key concepts and practical applications in parallel computing. While it aims to provide a comprehensive review of the material, some advanced topics might not be explored in full depth.

Almost all the content has been done following the subject's slides. Also personal examples and explanations from professional experience have been considered while doing this handbook.

This summary is available on GitHub. If you find typos, errors, want to add content (such as additional explanations, links, figures, or examples), or wish to contribute, you can do so at the following repository:
`https://github.com/mtiessler/191.114-Basics-of-Parallel-Computing`.

If the URL does not work, you can locate it on GitHub using the following details:

- **User**: mtiessler
- **Repo-Name**: 191.114-Basics-of-Parallel-Computing

2 Introduction to Parallel Hardware & Basic Notation

2.1 Hardware Fundamentals

2.1.1 Basic Definitions: CPU / Core / Socket

CPU (Central Processing Unit) The CPU, or processor, is the central unit that executes instructions stored in main memory. It features several registers (i.e., small, fast storage units with specific bit widths) for temporarily holding data during computations. A typical CPU is composed of:

- An **Arithmetic Logic Unit (ALU)** for performing arithmetic and logical operations.
- A **Control Unit** that orchestrates the execution sequence.
- Specialized units such as the **Floating Point Unit (FPU)**, which is optimized for floating-point arithmetic.

The CPU performs fundamental operations like load, store, operate, and jump commands.

Core Modern systems commonly use multi-core CPUs, meaning that several cores are integrated onto a single chip. Each core contains its own set of registers, an ALU, an FPU, and other essential components—essentially functioning as an independent processor. In contemporary architectures (e.g., those by Intel and AMD), cores typically share a common Level 3 (L3) cache (also known as the Last Level Cache or LLC), which is the largest and slowest cache in the hierarchy, while each core maintains its own private Level 1 (L1) and Level 2 (L2) caches. Cores communicate with each other via an interconnection network designed with a specific topology to ensure efficient data exchange.

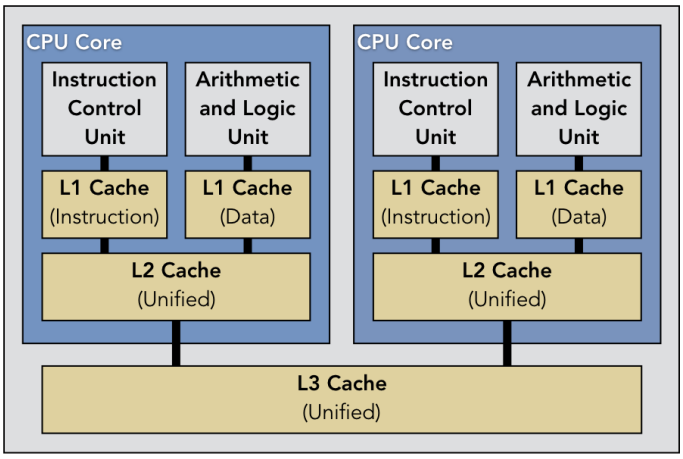


Figure 1: Example of dual-core processor with a shared L3 cache and dedicated L1/L2 caches [1].

Socket A socket (or CPU slot) is the physical connector on the motherboard that houses the CPU. Some motherboards provide multiple sockets, allowing for multi-processor configurations; an example is a four-socket system arranged in a crossbar configuration.

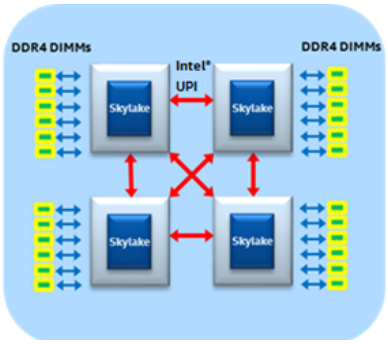


Figure 2: four-socket crossbar configuration [2]

Altogether, a system may contain multiple CPUs, where each CPU resides in its own socket and each CPU can have multiple cores, as illustrated below:

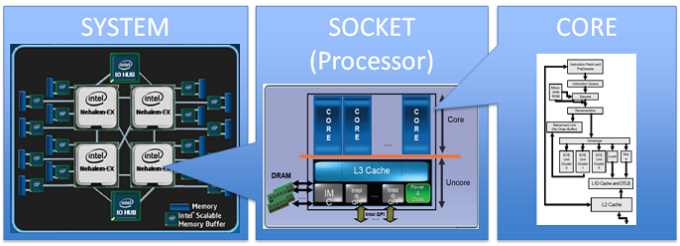


Figure 3: A modern multi-processor, multi-core system [3].

2.1.2 Additional Hardware Definitions

FPU (Floating Point Unit): A dedicated component within the CPU for performing floating-point arithmetic operations, crucial for handling real-number calculations.

Registers: High-speed storage locations within the CPU that temporarily hold data and instructions. The size of a register (its bit width) directly influences the amount of data it can process at once.

Cache Memory: A small, fast memory component used to store frequently accessed data, reducing access time from the main memory. Modern CPUs usually implement a multi-level cache hierarchy (L1, L2, and L3) to optimize performance.

2.2 Advanced Architectures

2.2.1 SIMD Instructions

Modern processors provide SIMD (Single Instruction, Multiple Data) instructions, which allow a single instruction to operate on multiple data elements simultaneously. This capability is key for accelerating multimedia processing, scientific computations, and other data-parallel tasks.

x86 Architectures x86 is a family of complex instruction set computing (CISC) architectures used primarily in personal computers and servers. SIMD on x86 is implemented through instruction sets such as:

- **SSE (Streaming SIMD Extensions):** Introduced in 1999, supports 128-bit vector operations.
- **AVX (Advanced Vector Extensions):** Offers vector widths of 256 and 512 bits, enhancing performance for floating-point intensive applications.

ARM Architectures ARM is a family of reduced instruction set computing (RISC) architectures designed for low power consumption and widely used in mobile and embedded devices. SIMD functionality in ARM is provided by:

- **Neon:** The primary SIMD extension for ARM, supporting 128-bit vector operations, commonly used in multimedia and signal processing.
- **Helium (M-Profile Vector Extension, MVE):** A more advanced SIMD extension supporting additional vector operations, beneficial for embedded applications.

Additional SIMD Definitions

Instruction Set: A collection of commands that a processor can execute. SIMD instruction sets extend these commands to process multiple data elements in parallel.

Vector Operations: Operations that apply a single instruction simultaneously to a group of data elements.

Scalar Mode: The mode in which instructions are executed on individual data elements one at a time.

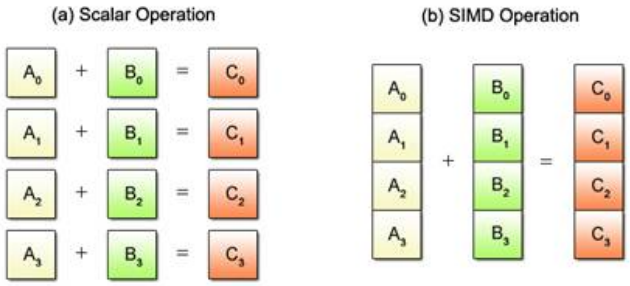


Figure 4: Comparison of SIMD vs. scalar execution.

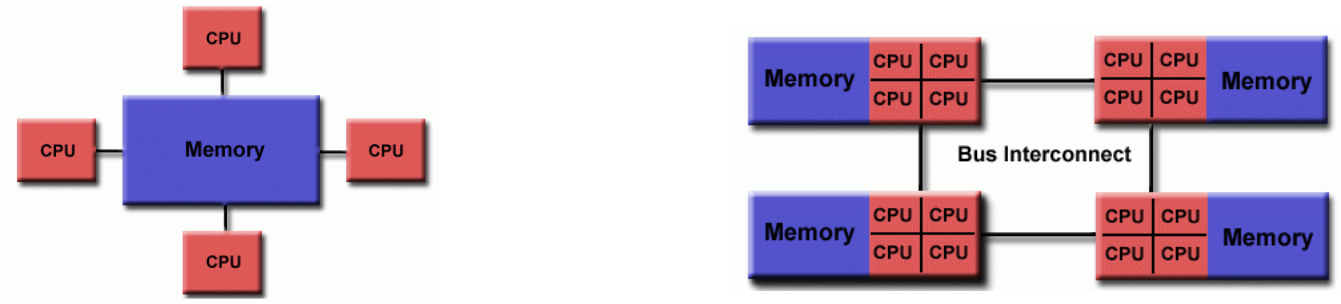
2.2.2 UMA & NUMA Architectures

Shared memory parallel computers are systems in which all processors have access to a common, global address space. In these systems, the main memory (DRAM) is shared among all processors, and any change made by one processor is visible to all others. However, the efficiency of this model depends on how the main memory is physically placed relative to the processors. Multi-processor systems are generally classified into two types based on this placement:

Uniform Memory Access (UMA) UMA systems, typically represented by Symmetric Multiprocessor (SMP) machines, feature identical processors with equal access times to memory. Often

referred to as CC-UMA (Cache Coherent UMA), these systems ensure that cache updates by one processor are immediately visible to others.

Non-Uniform Memory Access (NUMA) In NUMA systems, often formed by linking two or more SMPs, one SMP can directly access the memory of another. This results in different processors having varying access times; memory access over inter-SMP links is slower. When cache coherency is maintained, these systems are called CC-NUMA.



UMA Diagram NUMA Diagram

Figure 5: Comparison of UMA and NUMA architectures [4]

General Characteristics Shared memory systems provide the advantage of a global address space that simplifies programming and ensures fast, uniform data sharing due to the proximity of memory to CPUs. However, as more CPUs are added, scalability issues can arise. Increased traffic on the memory-CPU path and overhead from maintaining cache coherency may limit performance, requiring careful design of synchronization constructs to manage access correctly.

2.2.3 GPUs

Graphics Processing Units (GPUs) can be found as dedicated cards (e.g., via PCIe) or integrated into the CPU. The key idea behind GPUs is the use of many small processing cores; for example, an RTX 3080 features 8704 cores. These scalar processors execute threads optimized for data-parallel, throughput computations, and hide latency by overlapping execution from other threads.

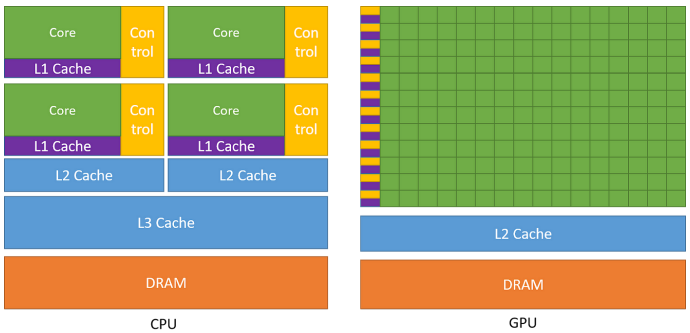


Figure 6: Comparison CPU & GPU [5]

2.3 Parallel Computing Concepts

2.3.1 Parallel Computing

Parallel computing is the **simultaneous use of multiple compute resources** to solve a computational problem. The problem is divided into discrete parts that can be solved concurrently, with instructions from each part executing simultaneously on different processors.

2.3.2 Distributed Computing

Distributed computing arises when a problem is solved by a set of distributed entities (e.g., processors, nodes, processes, actors, agents, sensors, or peers), where each entity has only **partial knowledge** of the entire problem. Communication among these entities is required to coordinate the solution.

2.3.3 High-Performance Computing (HPC)

High-Performance Computing (HPC) refers to aggregating computing power to deliver performance far exceeding that of a typical desktop or workstation. HPC is used to solve large problems in science, engineering, or business.

2.3.4 Supercomputers

A **supercomputer** is a large-scale computer system made up of many smaller computers (**nodes**), **each containing multiple processors or cores**. These nodes communicate over a **high-speed network** (e.g., InfiniBand or Omni-Path). An example is the SuperMUC-NG, which features:

- A total core count of 311,040 (peak performance of 26.9 PetaFlop/s).
- 719 TB of main memory.
- 6,336 thin compute nodes (each with 48 cores and 96 GB memory) and 144 fat compute nodes (each with 48 cores and 768 GB memory).
- Intel OmniPath networking in a “fat tree” configuration.

2.3.5 Processes & Threads

Both **processes** and **threads** can execute **independent instructions** and work with **different data**. Within a process, multiple threads can be created.

Processes: Heavier units that have their own memory space (address space). Communication between processes requires inter-process communication (IPC), which is generally slower.

Threads: Lighter units that share the same memory space, allowing direct access to shared variables. Context switches between threads are generally cheaper than between processes.

Unique to each thread are:

- Thread ID
- Saved registers, stack pointer, and instruction pointer
- Stack (for local variables, temporary variables, and return addresses)

Processes	Threads
Heavier	Lightweight
Own memory space (address space)	Shared address space
Require inter-process communication (often slow)	Direct access to shared variables
Context switches more expensive	Cheaper context switches

Table 1: Comparison between Processes and Threads

- Signal mask and priority (scheduling information)

Items shared among threads within a process include:

- Text segment (instructions)
- Data segment (static and global data)
- BSS segment (uninitialized data)
- Open file descriptors, signals, current working directory, user and group IDs

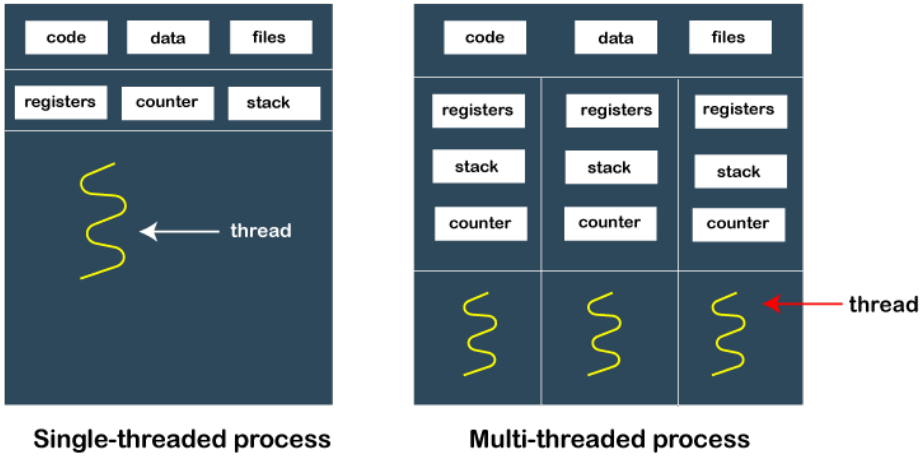


Figure 7: Processes vs Threads [6]

2.3.6 Parallel Computing Hierarchy

Parallelism can be exploited at several levels:

1. **Core Level:** Utilize **SIMD** instructions, **pipelining**, instruction-level parallelism (ILP) and **multiple functional units** arithmetic logical units (ALUs), floating point units (FPUs) within a single core.
2. **CPU/GPU Level:** Employ multiple control flows using **threads or processes** across **different cores or GPU** multiprocessors.
3. **Cluster Level:** Use several compute nodes (each with multiple CPUs) in parallel.

2.3.7 Hyper-Threading

Hyper-Threading (an Intel term for **simultaneous multithreading, SMT**) allows a single CPU core to run two software threads concurrently by duplicating parts of the CPU (mainly the registers) while **sharing execution units (ALUs, FPUs)**. This makes the CPU appear to have twice as many cores (logical cores) as physical ones, typically increasing performance by around 30%. However, **it does not double performance**.

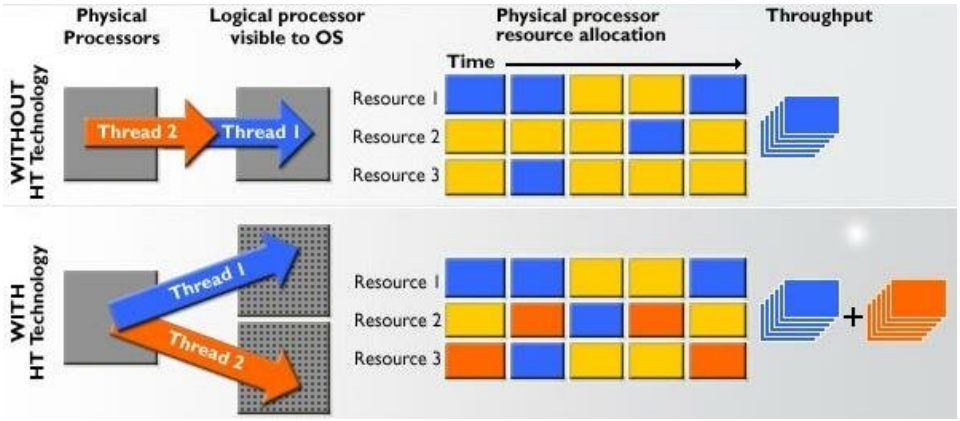


Figure 8: How Hyper-Threading works [7]

2.4 Speedup and Efficiency

2.5 Amdahl's Law

2.6 Relative Speed-up in Practice

2.7 NUMA Performance in Practice

2.8 Gustafson-Barsis's Law

2.9 Strong and Weak Scaling

2.10 Cost and Work of Parallel Programs

2.11 Cost Analysis / Example

3 Introduction to Parallel Programming with R and Python

3.1 Parallel Programming with R & `mclapply`

3.2 Parallel Computing Concepts in Python

References

[1] Paul A. M. S. Kirkpatrick. *Parallelism Versus Concurrency*. https://w3.cs.jmu.edu/kirkpams/OpenCSF/Books/csf/html/ParVConc.html. Accessed: 2025-03-26. 2021.

[2] Intel Corporation. *Intel® Xeon® Processor Scalable Family Technical Overview*. https://www.intel.com/content/www/us/en/developer/articles/technical/xeon-processor-scalable-family-technical-overview.html. Accessed: 2025-03-26. 2017.

[3] Intel Corporation. *Intel® Performance Counter Monitor*. https://www.intel.com/content/www/us/en/developer/articles/tool/performance-counter-monitor.html. Accessed: 2025-03-26. 2023.

[4] Lawrence Livermore National Laboratory. *Introduction to Parallel Computing Tutorial*. https://hpc.llnl.gov/documentation/tutorials/introduction-parallel-computing-tutorial. Accessed: 2025-03-26. 2023.

[5] University of New England. *COSC330 Lecture 18: Parallel Processing*. https://turing.une.edu.au/~cosc330/lectures/display_notes.php?lecture=18. Accessed: 2025-03-26. 2023.

[6] TPoint Technologies. *Process vs Thread*. https://www.tpointtech.com/process-vs-thread. Accessed: 2025-03-26. 2024.

[7] ResearchGate. *Fonctionnement de l'Hyperthreading sur processeur Intel*. https://www.researchgate.net/figure/Fonctionnement-de-lHyperthreading-sur-processeur-Intel_fig6_278638340. Accessed: 2025-03-26. 2015.