COLLEGE OF SCIENCE
MATHEMATICS
VIRGINIA TECH

COLLEGE OF SCIENCE
ACADEMY OF DATA SCIENCE
VIRGINIA TECH

COMPUTATIONAL MODELING
AND DATA ANALYTICS

O Introducere în Vectorizare și Aplicațiile Sale în Limbaje Moderne de Programare

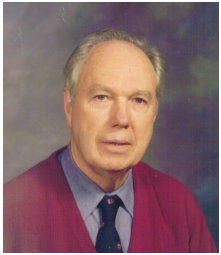
Ionuț Farcaș

17 Aprilie, 2025

Departamentul de Matematică și Divizia de Modelare Computațională și Analiză de Date
Virginia Tech

Today's agenda

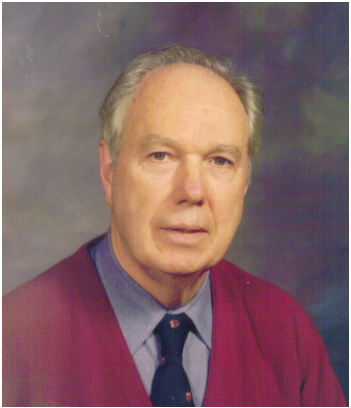
- A tale of two pioneers
  - von Neumann architecture
  - Flynn's taxonomy
- Moore's law
- The importance of parallel computing
- Data parallelism through vectorization
- Demos in `C` and `Python`



All materials are available at

https://github.com/ionutfarcas/tutorial_vectorization

A tale of two pioneers



An overview of the von Neumann model (architecture)

If Alan Turing was the Father, Charles Babbage the Grandfather, and Ada, Count of Lovelace, the Great-Aunt, John von Neumann was the impossibly talented, impossibly charismatic Uncle to Computing.

Computerphile

John von Neumann

- born on Dec. 28, 1903 in Budapest, Kingdom of Hungary as Neumann János Lajos
- child prodigy, genius
- died at only 53 in 1957
- regarded as one of the most important scientists of all time
- made fundamental contributions in a number of fields, ranging from quantum mechanics, game theory, algebra, ergotic theory, and computer science
- one of the early pioneers of computer science and computing

John von Neumann

- born on Dec. 28, 1903 in Budapest, Kingdom of Hungary as Neumann János Lajos
- child prodigy, genius
- died at only 53 in 1957
- regarded as one of the most important scientists of all time
- made fundamental contributions in a number of fields, ranging from quantum mechanics, game theory, algebra, ergotic theory, and computer science
- one of the early pioneers of computer science and computing

“There was a seminar for advanced students in Zürich that I was teaching and von Neumann was in the class. I came to a certain theorem, and I said it is not proved and it may be difficult. von Neumann didn’t say anything but after five minutes he raised his hand. When I called on him he went to the blackboard and proceeded to write down the proof. After that I was afraid of von Neumann." George Pólya

The von Neumann model (architecture)

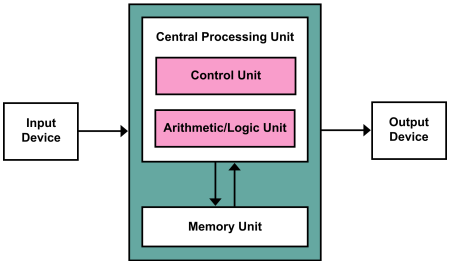
- Written by von Neumann in **1945** (First Draft of a Report on the EDVAC)
- The motivation was to create a **flexible, general-purpose computer design**
- Most **modern computers** are based on this model (with some additional ingredients)

The von Neumann model (architecture)

- Written by von Neumann in **1945** (First Draft of a Report on the EDVAC)
- The motivation was to create a **flexible, general-purpose computer design**
- Most **modern computers** are based on this model (with some additional ingredients)

Five main ingredients:

1. **Memory** (called “Store" in the early days)
2. **Control Unit (CU)**
3. **Arithmetic/Logic Unit (ALU)**
4. **Input Devices**
5. **Output Devices**



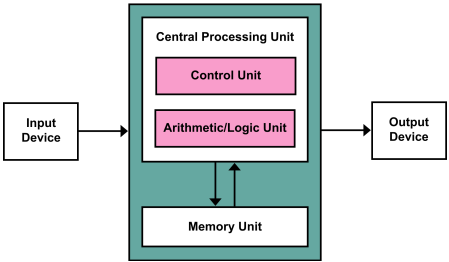
https://en.wikipedia.org/wiki/Von_Neumann_architecture

The von Neumann model (architecture)

- Written by von Neumann in **1945** (First Draft of a Report on the EDVAC)
- The motivation was to create a **flexible, general-purpose computer design**
- Most **modern computers** are based on this model (with some additional ingredients)

Five main ingredients:

1. **Memory** (called “Store" in the early days)
2. **Control Unit (CU)**
3. **Arithmetic/Logic Unit (ALU)**
4. **Input Devices**
5. **Output Devices**



https://en.wikipedia.org/wiki/Von_Neumann_architecture

Central idea: **Stored program concept**, i.e., **both data and instructions** for a **program** are stored in the **same memory unit**, allowing for **flexible** and **reprogrammable** computers

The von Neumann model (architecture)

- **Memory**: This is where **data** and **instructions** are stored. It is a crucial part of the computer system that allows for the storage and retrieval of information.

The von Neumann model (architecture)

- **Memory**: This is where **data** and **instructions** are stored. It is a crucial part of the computer system that allows for the storage and retrieval of information.
- **Control Unit (CU)**: This component **manages the operations** of the computer. It directs the flow of data between the CPU and other components.

The von Neumann model (architecture)

- **Memory**: This is where **data** and **instructions** are stored. It is a crucial part of the computer system that allows for the storage and retrieval of information.
- **Control Unit (CU)**: This component **manages the operations** of the computer. It directs the flow of data between the CPU and other components.
- **Arithmetic/Logic Unit (ALU)**: The ALU **performs** arithmetic and logical **operations**. It is responsible for **calculations**.

The von Neumann model (architecture)

- **Memory**: This is where **data** and **instructions** are stored. It is a crucial part of the computer system that allows for the storage and retrieval of information.
- **Control Unit (CU)**: This component **manages the operations** of the computer. It directs the flow of data between the CPU and other components.
- **Arithmetic/Logic Unit (ALU)**: The ALU **performs** arithmetic and logical **operations**. It is responsible for **calculations**.
- In modern computers, the CU and ALU (and other ingredients, such as **Registers**) form the **Central Processing Unit (CPU)**.

The von Neumann model (architecture)

- **Memory**: This is where **data** and **instructions** are stored. It is a crucial part of the computer system that allows for the storage and retrieval of information.
- **Control Unit (CU)**: This component **manages the operations** of the computer. It directs the flow of data between the CPU and other components.
- **Arithmetic/Logic Unit (ALU)**: The ALU **performs** arithmetic and logical **operations**. It is responsible for **calculations**.
- In modern computers, the CU and ALU (and other ingredients, such as **Registers**) form the **Central Processing Unit (CPU)**.
- **Input/Output devices**: Devices that allow the computer to **interact** with users and the outside world (e.g., keyboards, monitors etc.)

The fetch-execute-decode cycle

The **five basic stages** of the non Neumann architecture are:

1. **Fetch**: The CPU retrieves an instruction from memory.
2. **Decode**: The instruction is translated into signals the computer understands.
3. **Execute**: The CPU performs the instruction (e.g., calculations or data movement).
4. **Memory Access**: Data is read from or written to memory if needed.
5. **Write Back**: Results are saved back to memory or registers.

The fetch-execute-decode cycle

The **five basic stages** of the non Neumann architecture are:

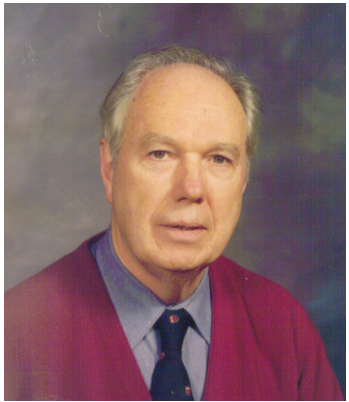
1. **Fetch**: The CPU retrieves an instruction from memory.
2. **Decode**: The instruction is translated into signals the computer understands.
3. **Execute**: The CPU performs the instruction (e.g., calculations or data movement).
4. **Memory Access**: Data is read from or written to memory if needed.
5. **Write Back**: Results are saved back to memory or registers.

The classical von Neumann architecture assumes a **sequential instruction execution**: Instructions are fetched and executed one at a time in a specific order.

The von Neumann model (architecture)–summary

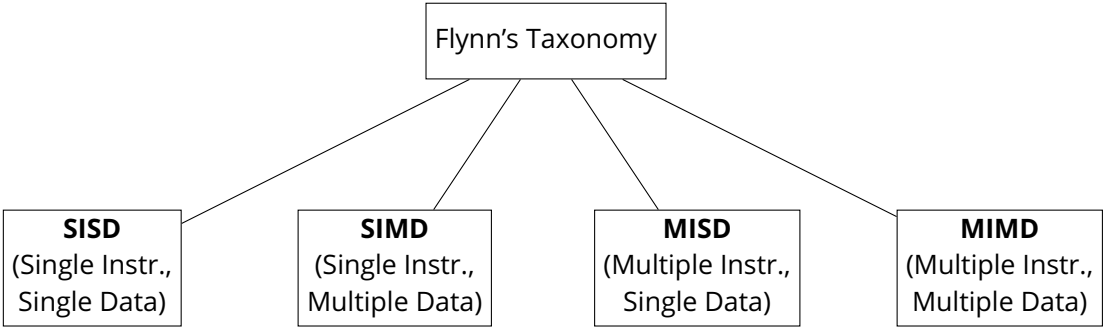
- Five important ingredients:
  1. **Memory**
  2. **Control Unit (CU)**
  3. **Arithmetic/Logic Unit (ALU)**
  4. **Input Devices**
  5. **Output Devices**
- **Fetch-Decode-Execute Cycle**: The process of retrieving an instruction from memory, decoding it, and then executing it.
- **Sequential Instruction Execution**: Instructions are fetched and executed one at a time in a specific order.
- Central idea: **Stored program concept**, i.e., **both data and instructions** for a **program** are stored in the **same memory unit**, allowing for **flexible** and **reprogrammable** computers.
- Most **modern computers** are based on this model (with some additional ingredients).

Flynn’s taxonomy for parallel computer architectures



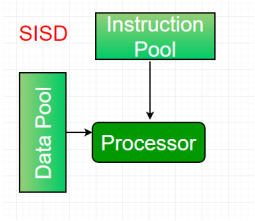
Flynn’s taxonomy—overview

- Proposed by Michael J. Flynn in 1966 and 1972.
- **Simple framework** for understanding different approaches to parallel processing.
- Utilizes **two key factors** for classification:
  - Number of concurrent **instruction streams**.
  - Number of concurrent **data streams**.



Single instruction, single data (SISD)

Single instruction, single data (SISD)



https://www.geeksforgeeks.org/computer-architecture-flynns-taxonomy/

- **Uniprocessor** machines.
- One processor executes **one instruction** stream on **one data** stream.
- Traditional **sequential computing**.
- **Classical** von Neumann architecture.
- Examples include **early personal computers** and **simple microcontrollers**.

Single instruction, single data (SISD)

Key characteristics

- **One** control unit and **one** processing unit.
- **Simple** to program and understand.
- **Limited** by the speed of a single processor.

Performance Considerations

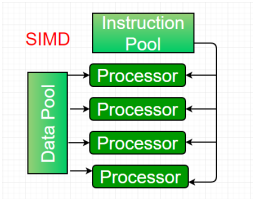
- Performance limited by **single instruction execution**.
- **No inherent parallelism**, relies on instruction-level optimizations.

Examples

- Traditional **single-core processors** (early Intel x86 CPUs).
- **Simple microcontrollers** in embedded systems (Arduino Uno).

Single instruction, multiple data (SIMD)

Single instruction, multiple data (SIMD)



https://www.geeksforgeeks.org/computer-architecture-flynns-taxonomy/

- Multiple processing elements perform the **same operation on multiple data points simultaneously**.
- Well suited for tasks with **high data parallelism** (image processing, scientific simulations).
- Examples include **vector processors** and **Graphics Processing Units**.

Single instruction, multiple data (SIMD)

Key characteristics

- Exploits **data-level parallelism**.
- Efficient for tasks with **regular data structures** (matrices, vectors).

Performance Considerations

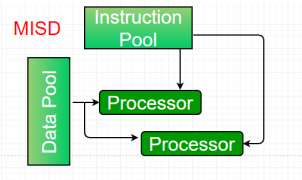
- High performance for **data-parallel tasks**.
- May suffer from poor utilization on **irregular data structures**.

Examples

- **Vector processors** (Cray-1 and Cray-2 supercomputers).
- **Graphics Processing Units** (NVIDIA, AMD Radeon).
- Intel's **SSE** (Streaming SIMD Extensions) and **AVX** (Advanced Vector Extensions).

Multiple instructions, single data (MISD)

Multiple instructions, single data (MISD)



https://www.geeksforgeeks.org/computer-architecture-flynns-taxonomy/

- **Multiple** instruction streams operate on a **single** data stream.
- Example: y = sin(x) + cos(x) + tan(x).
- **Rarely** implemented in practice.

Multiple instructions, single data (MISD)

Key characteristics

- **Multiple** instructions applied to a **single** data stream.
- Potential applications in fault-tolerant systems.

Performance Considerations

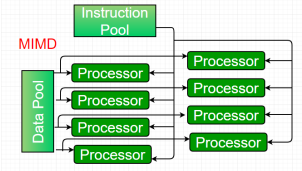
- Potential for **high redundancy and fault tolerance**.
- **Limited practical implementations** affect performance analysis.

Examples

- Some pipelined architectures in signal processing.
- Theoretical fault-tolerant systems with redundant processing.

Multiple instructions, multiple data (MIMD)

Multiple instructions, multiple data (MIMD)



https://www.geeksforgeeks.org/computer-architecture-flynns-taxonomy/

- Multiple processors execute different instruction streams on different data streams independently.
- Most **flexible and scalable** parallel architecture.
- Further classified into **shared memory** and **distributed memory** architectures.

Multiple instructions, multiple data (MIMD)

Key characteristics

- Most **flexible** and **general-purpose** parallel architecture.
- Supports **both task-level and data-level parallelism**.

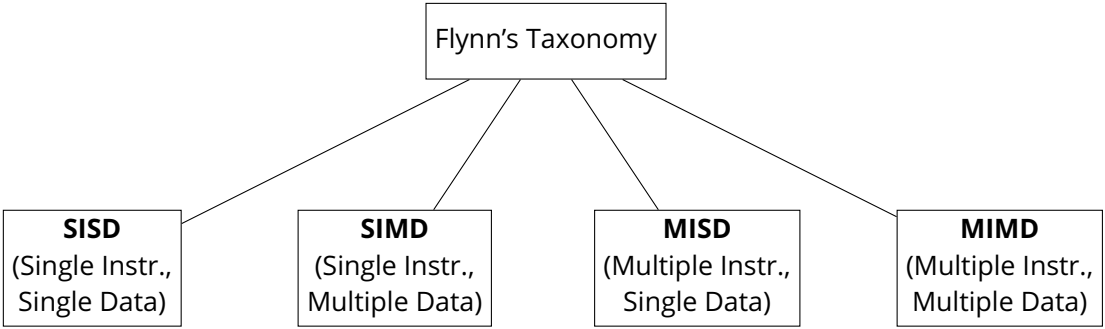
Performance Considerations

- Highest flexibility and potential for **performance**.
- However, this usually requires a **careful design** by the programmer.

Examples

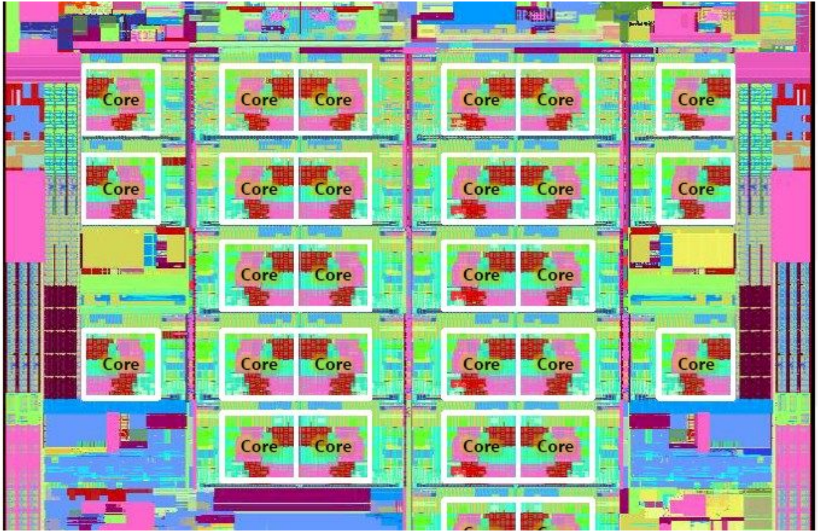
- **Multi-core** processors.
- **Distributed computing** systems.
- **Massively parallel processors** in supercomputers or compute clusters.

Flynn’s taxonomy–summary



Hybrid architectures often combine multiple categories to optimize performance.

Modern CPUs are getting increasingly more complex

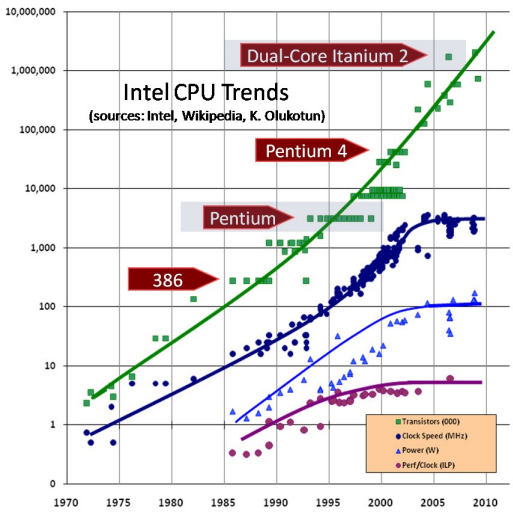


source: https://www.nextplatform.com/2020/08/17/the-ticking-and-tocking-of-intels-ice-lake-xeon-sp/

There’s plenty of room at the top: What will drive computer performance after Moore’s law?

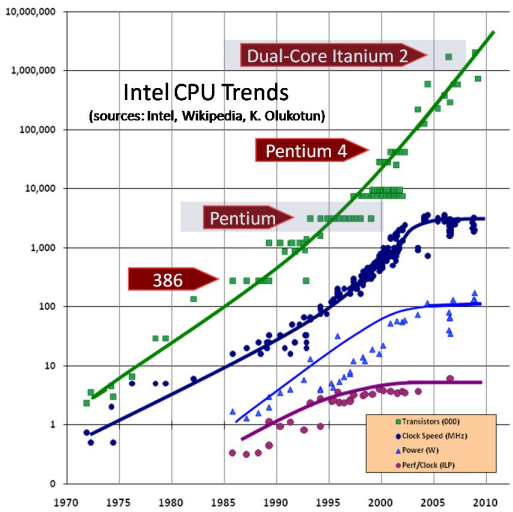
Moore's Law: The Observation

- In 1965, Gordon Moore (co-founder of Intel) observed that the number of transistors on a microchip **doubles** approximately every two years.
- This observation became known as **Moore's Law**.
- It has generally held true for several decades, leading to **exponential growth in computing power**.



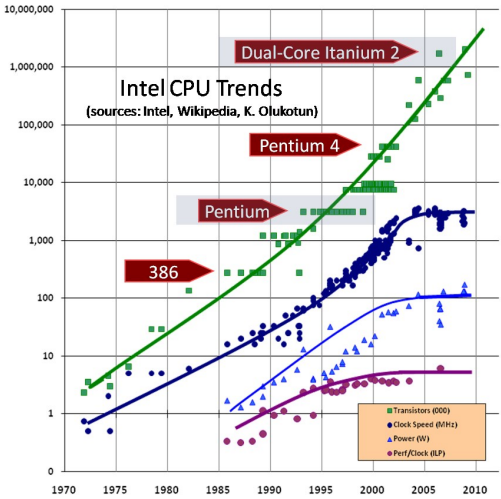
The Impact of Moore's Law

- Moore's Law has driven incredible **advancements in technology**:
  - **Faster processors**.
  - **Smaller** and **more powerful** devices.
  - The rise of the **personal computer**, the **internet**, and **mobile computing**.
- It has fueled **innovation** across countless industries.



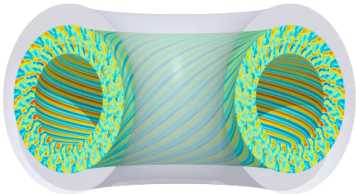
The End of Moore's Law?

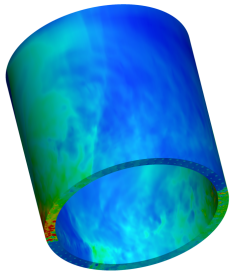
- Scaling transistors indefinitely is becoming **increasingly difficult** due to:
  - **Physical limitations** (quantum effects).
  - **Clock speed**
  - **Heat dissipation** problems.
  - **Economic constraints**.
- Moore's Law is **slowing down**.



The Demand for Large-Scale Computation

- Modern challenges in **science, engineering, and data analysis** require immense computational power.
- Examples:
  - Drug discovery and molecular simulations.
  - Artificial intelligence and machine learning.
  - Large-scale data processing and analytics.
  - High-energy physics simulations.
  - Designing the next generation propulsion devices.
  - Plasma Fusion.
- Sequential computing is increasingly **hitting its limits** due to physical and economical constraints (clock speed, power consumption, transistor miniaturization).





The Importance of Parallelism

- **Parallel computing** utilizes multiple processing units to solve a problem concurrently.
- Executing or processing **smaller sub-tasks or datasets simultaneously** allows for significant speedups through parallelism.
- Furthermore, the **scale** of most scientific computing and data analysis problems **surpasses the capabilities of a single computer**.
- Parallelism us to tackle problems that were previously intractable with traditional sequential approaches.
- Different levels of parallelism exist:
  - **Instruction-level parallelism** : pipelining, superscalar execution.
  - **Thread-level parallelism**: multi-core processors, shared-memory computing.
  - **Process-level parallelism**: distributed memory systems and computing.
  - **Data-level parallelism**: Vectorization $\Rightarrow$ This presentation.

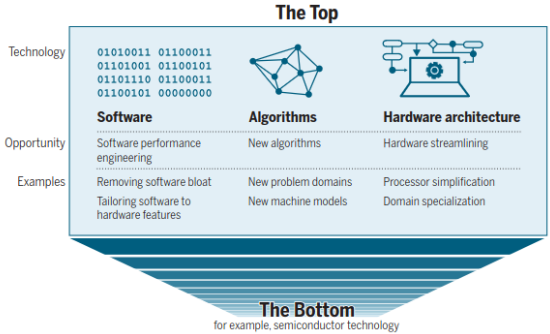
On modern hardware, parallel computing should be the **norm**, not the exception !

There is plenty of room at the top in computing

- The future of high-performance computing lies in **effectively harnessing the power of parallel architectures**.
- **Understanding parallel programming paradigms and techniques** is becoming increasingly crucial for researchers and developers.
- "**There is plenty of room at the top**" signifies the vast potential for advancements and discoveries enabled by parallel computing.
- **Parallel computing** extends the scope of computationally tractable problems, facilitating progress in addressing significant scientific and societal challenges.

The main drivers are **hardware architectures** + (scalable) **algorithms** + their **efficient software implementation**.

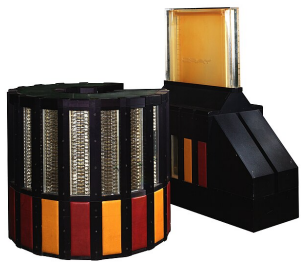
There is plenty of room at the top in computing



The main drivers are **hardware architectures** + (scalable) **algorithms** + their **efficient software implementation**.

[Leiserson et alia; Science, 2020]

Data parallelism through vectorization

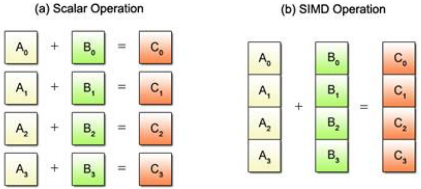


If you were plowing a field, which would you rather use: Two strong oxen or a thousand-twenty-four chickens ?

Seymour Cray, Computing Pioneer

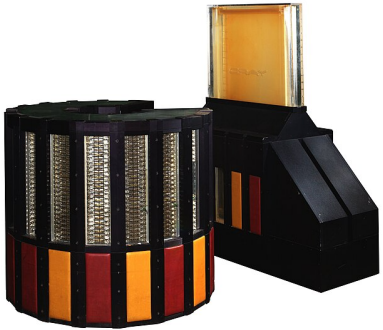
What is vectorization

- Simply put, vectorization means **performing same operation on multiple data elements simultaneously**.
- Instead of processing data elements one by one (scalar processing), vectorization processes **multiple elements in parallel** (i.e., **data parallism**).



- Benefits of vectorization:
  - Potential significant **performance improvements for data-parallel tasks**.
  - Increased **throughput and efficiency**.
  - **Reduced** instruction overhead.

Vectorization is not new



https://en.wikipedia.org/wiki/Cray-2

- One of the first prominent supercomputers, `Cray-2` released in 1985 was a **vector computer**.
- It featured **four vector processors** and a large 256 million 64-bit memory.
- The Cray-2 reached a peak speed of **1.9 GFLOPS** and was the fastest machine in the world when it was released.
- As a comparison, the Google Pixel 8 operates at 1594.8 GFLOPS.

SIMD Instructions on Intel Processors

Intel has a long history of SIMD extensions:

- **MMX (MultiMedia eXtensions)**: Introduced with the Pentium MMX in 1997. Operates on 64-bit integer data.
- **SSE (Streaming SIMD Extensions)**: Introduced with the Pentium III in 1999. Supports 128-bit floating-point and integer data. Evolution includes SSE2 up to SSE4.2.
- **AVX (Advanced Vector Extensions** (2011 Sandy Bridge). Supports 256-bit floating-point and integer data. AVX2 (Haswell, 2013) added integer vector instructions.
- **AVX-512**: Introduced with Knights Landing (Xeon Phi) and later on server and high-end desktop processors. Supports 512-bit floating-point and integer data. Offers significant parallelism.

Intel's SIMD instructions provide parallel operations on vectors in **dedicated registers**.

Compilers can often **auto-vectorize loops**, but **manual optimization using intrinsics or assembly** can provide further gains.

SIMD Instructions on AMD Processors

AMD also implements SIMD extensions, largely compatible with Intel's offerings:

- AMD adopted SSE with the Athlon XP.
- AMD64 architecture (x86-64) included support for SSE2.
- Later processors added support for SSE3, SSSE3, SSE4a (AMD's extension) to SSE4.2.
- AMD implemented AVX with the Bulldozer architecture in 2011.
- AVX2 support was added with the Steamroller architecture in 2014.
- AMD's Zen architecture (released in 2017) also supports AVX-512 in some server-class processors (e.g., EPYC). Client processors typically do not fully implement the wide 512-bit units for power efficiency reasons, often supporting 256-bit operations with AVX-512 encoding.

Similar to Intel, AMD's SIMD capabilities enable **parallel data processing**.

Software development for vectorization often targets **common instruction sets** supported by both vendors.

Example: Vector Addition (Conceptual)

- Imagine adding two arrays of four numbers: $A = [a_0, a_1, a_2, a_3]$ and $B = [b_0, b_1, b_2, b_3]$.
- **Scalar approach (4 instructions):**

$$c_0 = a_0 + b_0$$
$$c_1 = a_1 + b_1$$
$$c_2 = a_2 + b_2$$
$$c_3 = a_3 + b_3$$

- **Vectorized approach (1 instruction using a 128-bit SIMD register):**
  - **Load** $[a_0, a_1, a_2, a_3]$ into a vector register.
  - **Load** $[b_0, b_1, b_2, b_3]$ into another vector register.
  - Perform a **single vector addition** operation to get $[c_0, c_1, c_2, c_3] = [a_0 + b_0, a_1 + b_1, a_2 + b_2, a_3 + b_3]$.
- For larger arrays, the **speedup from vectorization** can be significant.

Considerations for Vectorization

Data Alignment: Efficient vector loads and stores often require data to be aligned in memory.

Data Dependencies: If the computation of one element depends on another, vectorization might be challenging.

Loop Unrolling: Can sometimes improve vectorization opportunities.

Compiler Optimization: Relying on the compiler's auto-vectorization capabilities is common, but understanding compiler reports and using optimization flags is important.

Intrinsics: Using processor-specific intrinsic functions allows for fine-grained control over SIMD instructions.

Libraries: Many libraries (e.g., NumPy, Eigen, Intel MKL, AMD Optimizing CPU Libraries) are heavily vectorized.

Intel Intrinsics for SIMD

Intrinsics are compiler built-in functions that directly map to specific assembly instructions, including SIMD instructions.

They provide a **higher-level abstraction** than raw assembly, making it easier to utilize SIMD capabilities in C/C++ code.

Intrinsics are **specific to the target architecture** and SIMD extension (SSE, AVX, etc.).

Header files (e.g., ‘<immintrin.h>‘, ‘<xmmintrin.h>‘, ‘<emmintrin.h>‘, ‘<pmmintrin.h>‘, ‘<tmmintrin.h>‘, ‘<smmintrin.h>‘, ‘<nmmintrin.h>‘, ‘<avxintrin.h>‘, ‘<avx2intrin.h>‘, ‘<avx512f.h>‘) provide the declarations for these intrinsics.

Naming conventions often indicate the data type, operation, and vector width (e.g., ‘_mm256_add_ps‘ for adding packed single-precision float in a 256-bit vector).

Simple Example: SSE Addition using Intrinsics in C

```c
#include <stdio.h>
#include <xmmintrin.h> // For SSE intrinsics

int main(int argc, char **argv) {
    float a_data[4] = {1.0f, 2.0f, 3.0f, 4.0f};
    float b_data[4] = {5.0f, 6.0f, 7.0f, 8.0f};
    float result_data[4];

    // Load the float arrays into 128-bit XMM registers (four floats)
    __m128 a = _mm_loadu_ps(a_data);
    __m128 b = _mm_loadu_ps(b_data);

    // Perform vectorized addition
    __m128 sum = _mm_add_ps(a, b);

    // Store the result from an `__m128` register back to memory
    _mm_storeu_ps(result_data, sum);
    return 0;
}
```

Let’s have a look at some simple code examples in `C` and `Python`

Conclusion

- Vectorization through SIMD instructions can be crucial for achieving high performance on modern Intel and AMD processors.
- Understanding the different SIMD extensions and their capabilities is important for performance-critical applications.
- Both Intel and AMD offer powerful SIMD capabilities, and software development often aims for compatibility across these platforms.
- Effective vectorization requires careful consideration of data structures, algorithms, and optimization techniques.
- Many modern libraries used by high-level language (and written in lower-level, faster languages) make use of vectorization.

Further materials

- SIMD tutorial.
- Great article: What Every Programmer Should Know About Memory.
- Paper There's plenty of room at the Top: What will drive computer performance after Moore's Law?
- SIMD in Julia: see this and this page.