Introduction to Programming

Vincent Boyer

UANL-FIME

Objectives

The main objectives of this presentation are:

- Educate on Computer Architecture: Provide an overview of computer systems and their components, emphasizing how hardware interacts with software.
- Explain Levels of Programming Languages: Clarify the distinction between low-level and high-level languages, highlighting their purposes and advantages.
- Guide Language Selection: Assist in choosing appropriate programming languages based on application requirements, performance considerations, and development preferences.
- Encourage Further Exploration: Stimulate interest in programming by showcasing its relevance, versatility, and impact across various fields and industries.

Before we start...

GitHub Repository

Google Colab

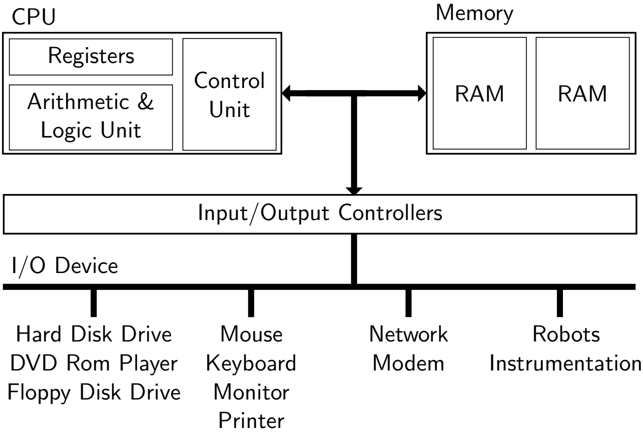
PyCharm Community

VS Community

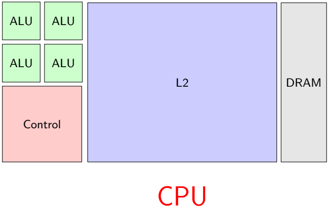
Agenda

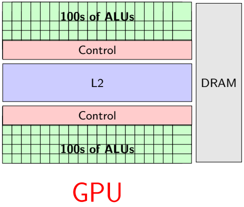
1. Introduction
2. Overview of computer architecture components
3. Programming Languages
4. Choosing the Right Language for the Job
5. Basic Concepts
6. Q&A and Coding

Overview of computer architecture components

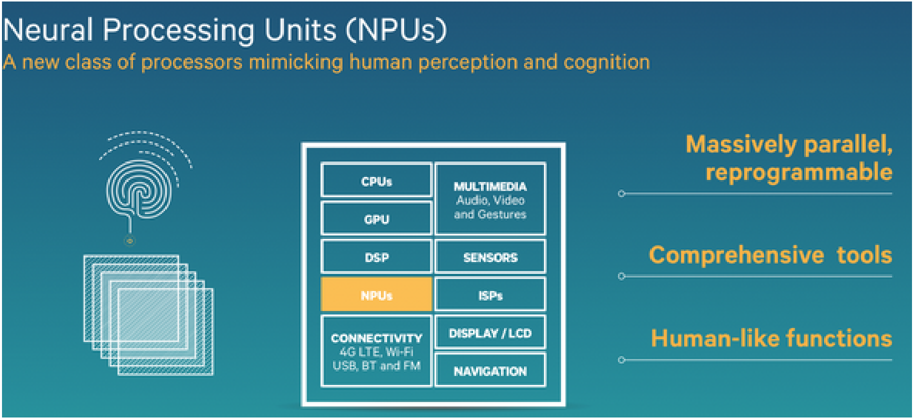


CPU VS GPU


ALU
ALU
ALU
ALU
Control
L2
DRAM
CPU


100s of ALUs
Control
L2
Control
100s of ALUs
DRAM
GPU

NPU



Neural Processing Units (NPUs)

A new class of processors mimicking human perception and cognition

CPUs

GPU

MULTIMEDIA
Audio, Video
and Gestures

DSP

SENSORS

NPUs

ISPs

CONNECTIVITY
4G LTE, Wi-Fi
USB, BT and FM

DISPLAY / LCD

NAVIGATION

Massively parallel, reprogrammable

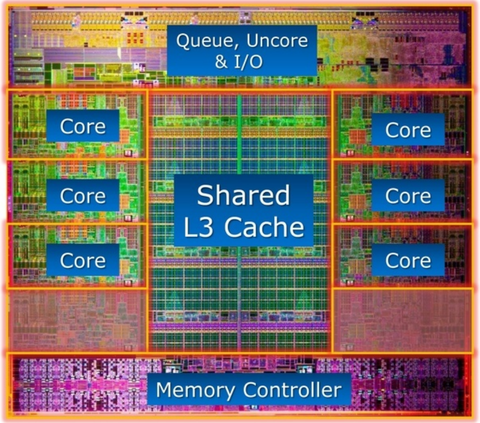
Comprehensive tools

Human-like functions

Key Differences

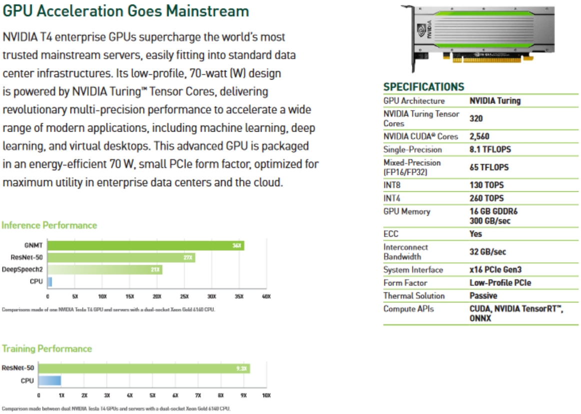
- **CPU (Central Processing Unit):**
  - General-purpose
  - Versatile but not specialized
  - Handles everyday computing tasks
- **GPU (Graphics Processing Unit):**
  - Originally for graphics rendering
  - Excellent at parallel processing
  - Used in scientific computing, simulations, and neural networks
- **NPU (Neural Processing Unit):**
  - Specialized for AI tasks
  - Optimizes neural network operations
  - Used in face recognition, NLP, and image processing

Parallel Computers: Multi-core Processor



Intel i7 Architecture

Parallel Computers: GPU



GPU Acceleration Goes Mainstream

NVIDIA T4 enterprise GPUs supercharge the world's most trusted mainstream servers, easily fitting into standard data center infrastructures. Its low-profile, 70-watt (W) design is powered by NVIDIA Turing™ Tensor Cores, delivering revolutionary multi-precision performance to accelerate a wide range of modern applications, including machine learning, deep learning, and virtual desktops. This advanced GPU is packaged in an energy-efficient 70 W, small PCIe form factor, optimized for maximum utility in enterprise data centers and the cloud.

Inference Performance

Comparisons made of one NVIDIA Tesla T4 GPU and servers with a dual-socket Xeon Gold 6140 CPU.

Training Performance

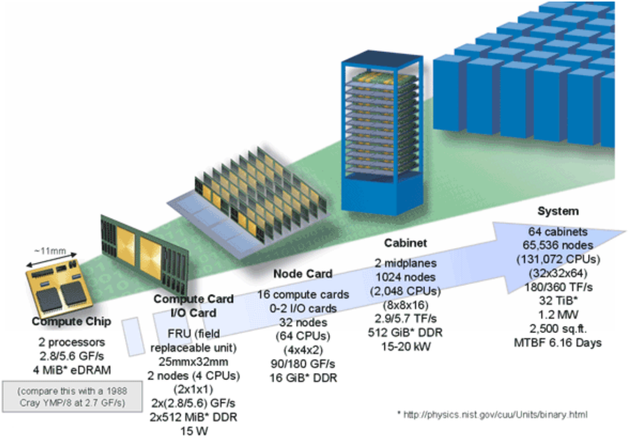
Comparison made between dual NVIDIA Tesla T4 GPUs and servers with a dual-socket Xeon Gold 6140 CPU.

SPECIFICATIONS

GPU Architecture	**NVIDIA Turing**
NVIDIA Turing Tensor Cores	**320**
NVIDIA CUDA® Cores	**2,560**
Single-Precision	**8.1 TFLOPS**
Mixed-Precision (FP16/FP32)	**65 TFLOPS**
INT8	**130 TOPS**
INT4	**260 TOPS**
GPU Memory	**16 GB GDDR6 300 GB/sec**
ECC	**Yes**
Interconnect Bandwidth	**32 GB/sec**
System Interface	**x16 PCIe Gen3**
Form Factor	**Low-Profile PCIe**
Thermal Solution	**Passive**
Compute APIs	**CUDA, NVIDIA TensorRT™, ONNX**

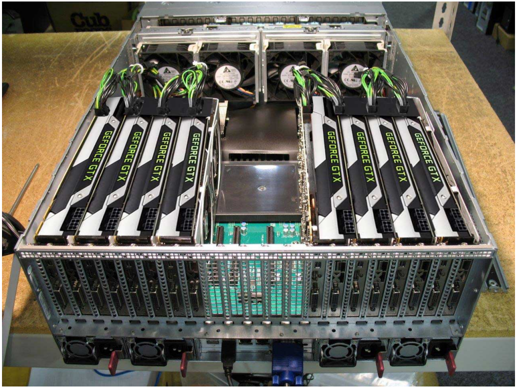
Nvidia T4

Parallel Computers: Cluster



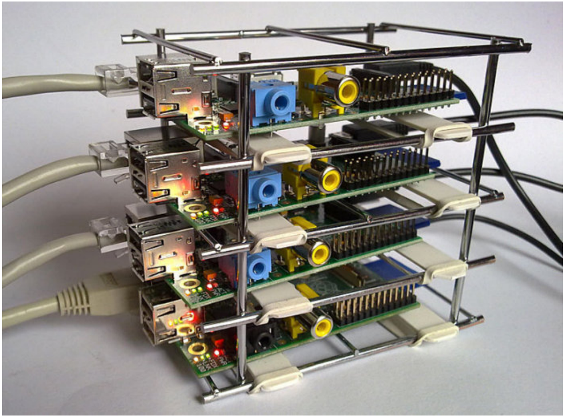
IBM Blue Gene

Parallel Computers: Cluster



Nvidia Cluster

Parallel Computers: Cluster



Rasberry Pi Cluster

Introduction to Programming Languages

Why So Many Languages?

- Diverse Needs: Different tasks require different tools.
- Evolution: Languages adapt to technological advancements.
- Community and Creativity: Developers invent new languages.

High-Level vs. Low-Level:

- High-Level Languages: Abstraction for easier coding.
- Low-Level Languages: Closer to machine code for performance.

Scripting/Interpreted Language:

Perl, Python, Shell, Java, ...

High/Middle Level Language:

C/C++, Fortran, Pascal, ...

Low Level Language:

Assembly Language.

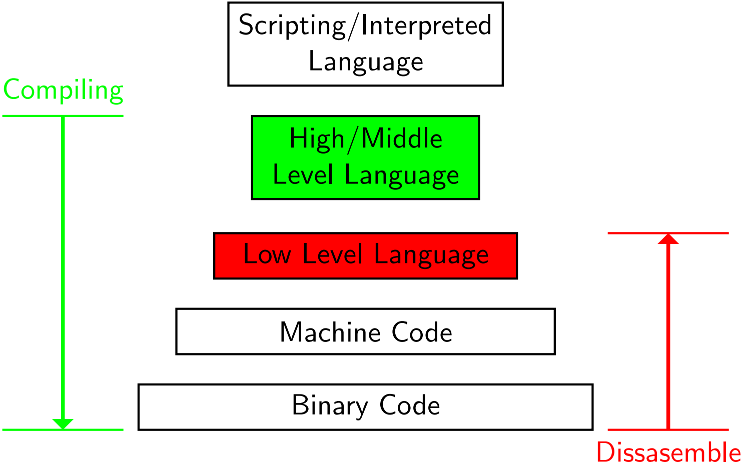
Machine Code:

Hexadecimal code read by the Operating System.

Binary Code:

Code read by hardware (not human-readable).

Flow of Compilation



Integrated Development Environments (IDEs)

- An **IDE** (Integrated Development Environment) is a software application that provides comprehensive facilities for software development.
- Key features of IDEs:
  - Source code editor: Allows writing and editing code.
  - Build automation tools: Compile and execute code.
  - Debugger: Inspect variables, step through code, and find errors.
- IDEs combine these tools within a single graphical user interface (GUI).
- Benefits of using an IDE:
  - Increased productivity: Streamlines common development tasks.
  - Syntax highlighting: Visual cues for keywords and language elements.
  - Autocomplete: Predicts and completes code as you type.
  - Debugging tools: Helps find and fix errors.

Debugging vs. Profiling

- **Debugger**
  - Helps identify and fix issues in your code.
  - Allows you to step through and analyze code execution.
  - Pauses the program, examines variables, and manipulates them.
- **Profiler**
  - Focuses on measuring code performance.
  - Identifies hotspots and bottlenecks.
  - Provides insights for optimization.
  - Collects data on function call times, memory usage, CPU load, etc.

Some Popular IDEs and Language Compatibility

- **Visual Studio (VS)**
  - Supports: C#, C++, Python, JavaScript, and more.
  - Features: Robust debugging, code analysis, and extensions.
- **PyCharm**
  - Supports: Python, Django, and scientific computing libraries.
  - Features: Python-specific tools, debugging, and web development support.
- **Eclipse**
  - Supports: Java, C/C++, Python, and more via plugins.
  - Features: Extensible, cross-platform, and strong community support.
- **Jupyter**
  - Supports: Interactive Python (via Jupyter notebooks).
  - Features: Data exploration, visualization, and scientific computing.
- **DevC++ (Dev-Cpp)**
  - Supports: C and C++.
  - Features: Lightweight, simple interface, and compiler integration.

Importance of Choosing the Right Language

Purpose of the Project:

Define project requirements and goals.
Different languages excel in various domains (web, mobile, data science, etc.).

Learning Curve and Ease of Use:

Consider your skill level and experience.
Some languages are more beginner-friendly than others.

Performance and Scalability:

Choose a language that meets performance needs.
Scalability matters for long-term projects.

Importance of Choosing the Right Language

Community and Support:

Widespread languages have abundant resources.
Collaboration and community support matter.

Availability of Tools and Libraries:

Access to libraries and frameworks is crucial.
Ecosystem matters for productivity.

Job Market and Career Prospects:

Popular languages align with industry trends.
Better job prospects and career growth.

Programming Languages Landscape

Python

- Benefits: Simplicity, readability, versatility.
- Applications: Data analysis, machine learning, web development, automation.

JavaScript

- Benefits: Widely used for web development, both front-end and back-end.
- Applications: Building interactive web applications, browser extensions.

Programming Languages Landscape

Java

- Benefits: Platform independence (runs on the Java Virtual Machine).
- Applications: Enterprise applications, Android app development.

C++

- Benefits: High performance, low-level control.
- Applications: Systems programming, game development, embedded systems.

SQL

- Benefits: Specialized for database management.
- Applications: Querying databases, managing data.

Parallel Computing Frameworks

POSIX Threads (Pthreads)

- Standard for creating and managing threads in a shared-memory environment.
- Widely used for parallel programming on multi-core CPUs.

OpenMP (Open Multi-Processing)

- Directive-based API for shared-memory parallelism.
- Eases parallelization of loops, sections, and tasks.

CUDA (Compute Unified Device Architecture)

- NVIDIA's parallel computing platform for GPUs.
- Enables high-performance GPU programming.

Parallel Computing Frameworks

OpenCL (Open Computing Language)

- Cross-platform framework for heterogeneous computing.
- Supports CPUs, GPUs, FPGAs, and other accelerators.

OpenACC

- Directive-based approach for GPU acceleration.
- Simplifies porting code to GPUs.

Pseudo-Code

What is Pseudo-Code?

- A step-by-step description of an algorithm.
- Uses simple English language text (not a specific programming language).
- Intended for human understanding, not machine execution.

Why Use Pseudo-Code?

- Algorithm Design: Helps plan the solution to a problem.
- Transition to Code: Acts as an intermediate step between idea and implementation.
- Language Independence: Not tied to any specific programming language.

Pseudo-Code Example: Calculating Average

Problem Statement:

Calculate the average of two given numbers.

Pseudo-Code:

- Get user input for ‘number1‘.
- Get user input for ‘number2‘.
- Calculate ‘average‘ as the sum of ‘number1‘ and ‘number2‘, divided by 2.
- Display the ‘average‘.

Operators in Programming

Arithmetic Operators:

- ‘$+$‘ (Addition) and ‘-‘ (Subtraction)
- ‘$\times$‘ (Multiplication) and ‘/‘ (Division)
- ‘%‘ (Modulus)

Relational Operators:

- ‘$==$‘ (Equal to) and ‘$\neq$‘ (Not equal to)
- ‘$<$‘ (Less than) and ‘$\leq$‘ (Less than or equal to)
- ‘$>$‘ (Greater than) and ‘$\geq$‘ (Greater than or equal to)

Operators in Programming

Logical Operators:

- ‘$\wedge$‘ (Logical AND)
- ‘$\vee$‘ (Logical OR)
- ‘$\neg$‘ (Logical NOT)

Assignment Operator:

- ‘$=$‘ (Simple assignment)

Variables in Programming

What are Variables?

- Named storage locations in memory.
- Hold data (values, references, etc.).
- Used to manipulate and store information.

Data Types:

These define the kind of data a variable can hold. Common data types include integers, floating-point numbers, strings, and booleans.

Variables in Programming

Data Type	Range
int (integer)	-2^{63} to $2^{63}-1$
float (single-precision floating point)	$\pm 1.18 \times 10^{-38}$ to $\pm 3.4 \times 10^{38}$
double (double-precision floating point)	$\pm 2.23 \times 10^{-308}$ to $\pm 1.8 \times 10^{308}$
char (character)	0 to 255 (ASCII values)
bool (boolean)	true or false

Flow Control Structure

Conditionals (if/else):

These allow you to make decisions based on conditions. For instance, you can execute different code blocks depending on whether a condition is true or false.

Loops (for/while):

Loops help you repeat a set of instructions. For example, you can iterate over a list of items or perform a task a specific number of times.

Flow Control Structure Example

```
Input: $n$ (integer)
Output: Sum of first $n$ positive integers
$N \leftarrow 10$
$sum \leftarrow 0$
for $i \leftarrow 1$ to $n$ do                          ▷ Loop from 1 to $n$
    $sum \leftarrow sum + i$                            ▷ Add $i$ to the sum
Print ”Sum of first $N$ positive integers: $sum$”
if $sum > 50$ then
    Print ”The sum is greater than 50.”
else
    Print ”The sum is not greater than 50.”
while $N > 0$ do
    Print ”Countdown: $N$”
    $N \leftarrow N - 1$
```

Functions and Procedures

Functions

- A **function** is a reusable block of code that performs a specific task.
- It takes input (arguments) and produces an output (return value).
- Functions are essential for modular and organized programming.

Procedures

- A **procedure** is a set of commands or instructions executed sequentially.
- It doesn't necessarily return a value; its purpose is to perform actions.
- Procedures are often used for side effects (e.g., printing, updating data).

Syntax in Programming

What is Syntax?

- Syntax refers to the rules governing how code is written in a programming language.
- It defines the structure and format of instructions.
- Correct syntax ensures that computers can understand and execute code.

Why is Syntax Important?

- Valid syntax is essential for successful communication with the machine or compiler.
- Syntax errors occur when code violates language rules.

Syntax in Programming

Examples

- Python: Uses indentation for code blocks.
- C++ and Java: Requires semicolons to end statements.
- C++: Uses curly braces for code blocks.
- Python uses # for single-line comments.
- Java and C++ use // for single-line comments.

Questions?

"Give someone a program; you frustrate them for a day; teach them how to program, and you frustrate them for a lifetime" – David Leinweber

Any questions from the audience?

Register your participation