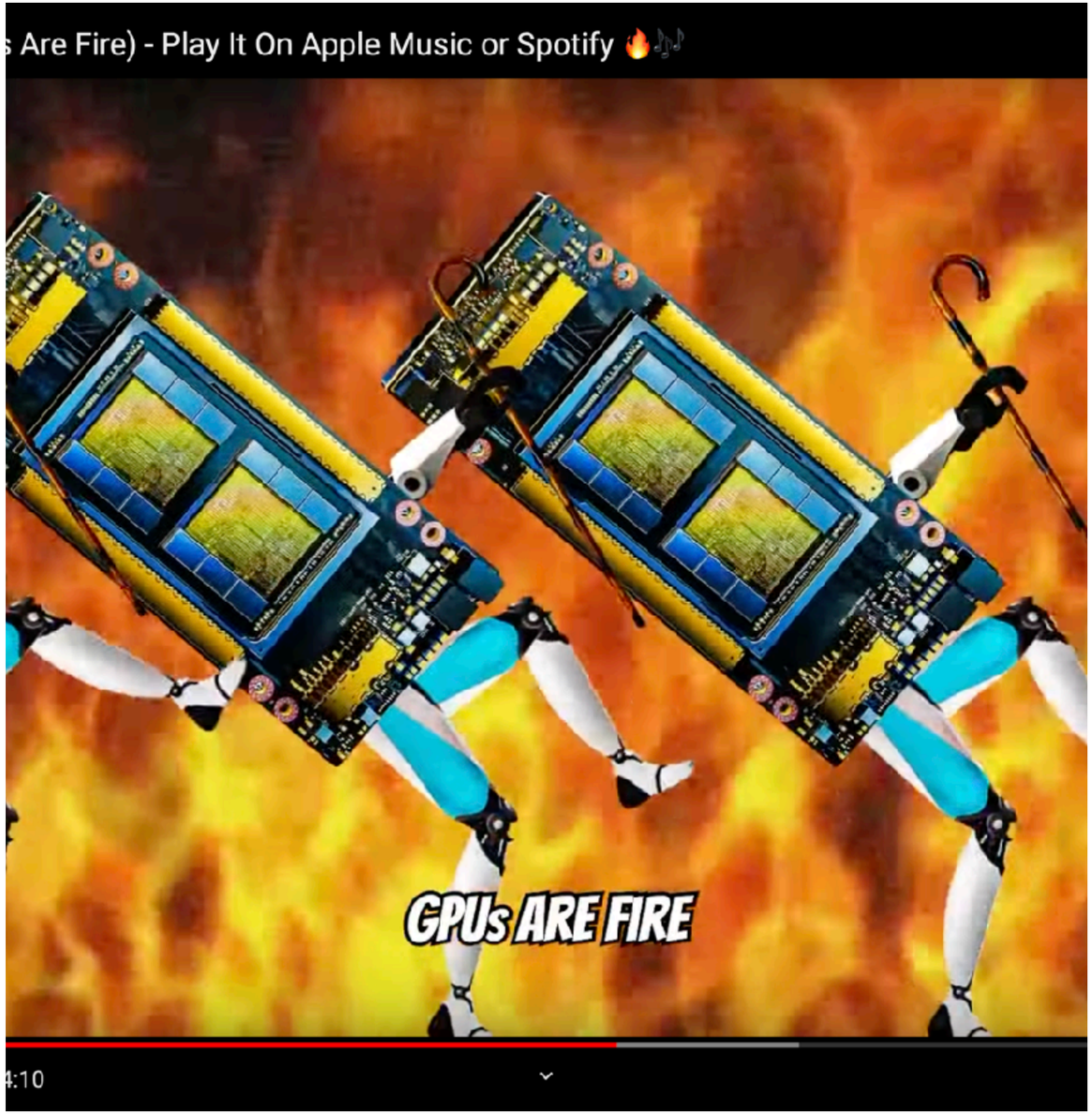
AI Chips: Blast From the Past

GPUs are FIRE 🔥

https://www.youtube.com/watch?v=YGpnXANXGUg



465.07 USD

+402.04 (637.85%) ↑ past 5 years

Closed: Aug 1, 7:59 PM EDT • Disclaimer

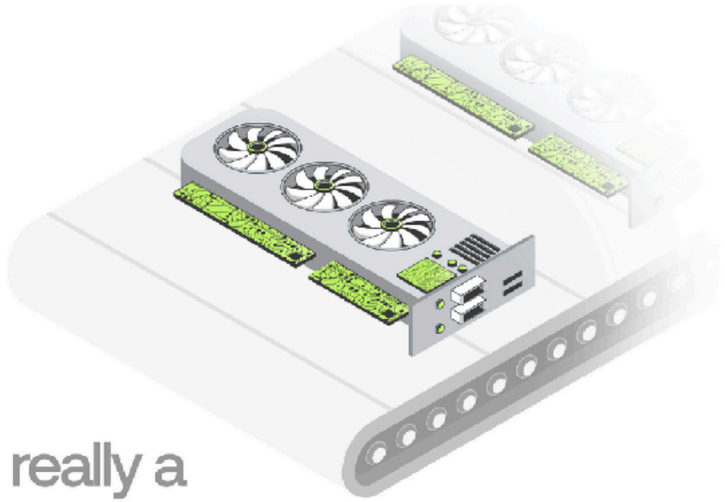
After hours 461.95 −3.12 (0.67%)

1D | 5D | 1M | 6M | YTD | 1Y | 5Y | Max

Open	464.60	Mkt cap	1.15T	CDP score	B
High	469.00	P/E ratio	241.70	52-wk high	480.88
Low	460.27	Div yield	0.034%	52-wk low	108.13

Nvidia H100 GPUs: Supply and Demand

July 2023 · Updated: August 2023



Is there really a

Bottleneck?

How Many GPUs Are Needed?

- GPT-4 was likely trained on somewhere between 10,000 to 25,000 A100s.[20]
- Meta has about 21,000 A100s, Tesla has about 7,000 A100s, and Stability AI has about 5,000 A100s.[21]
- Falcon-40B was trained on 384 A100s.[22]
- Inflection used 3,500 H100s for their GPT-3.5 equivalent model.[23]

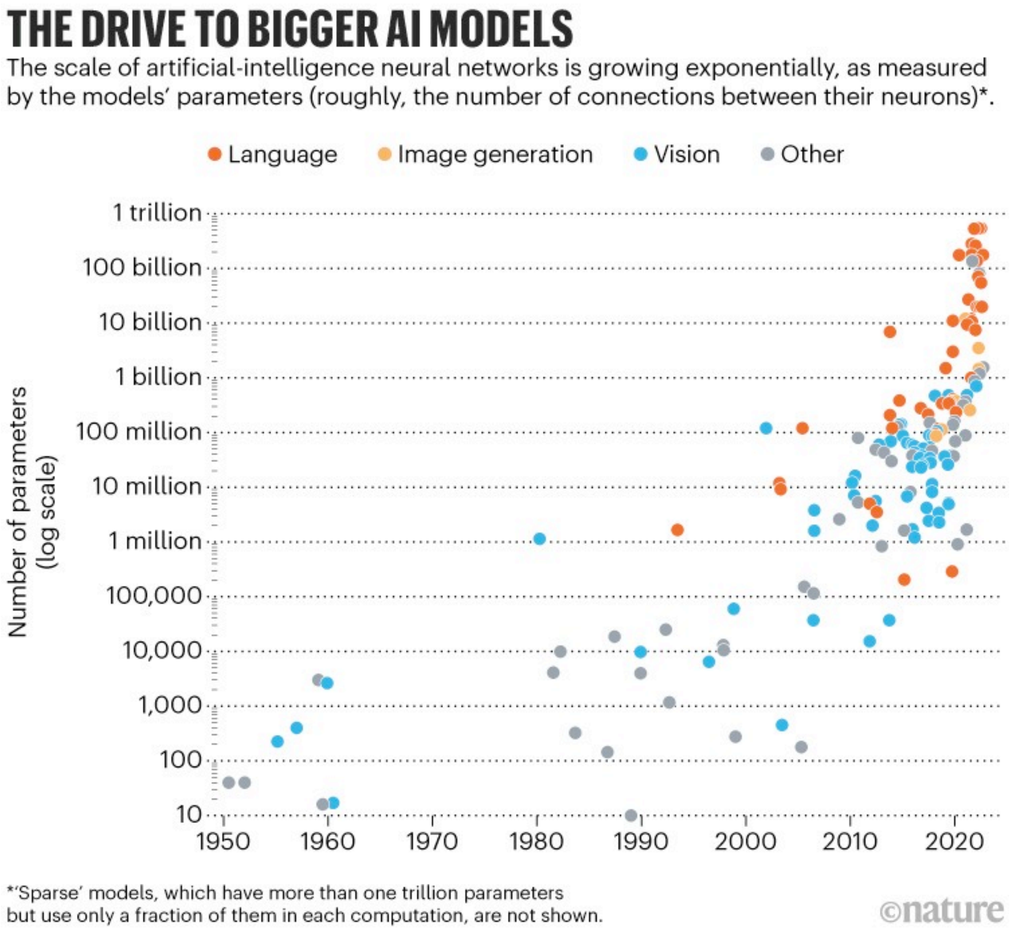
Specifications	
GPU	8x NVIDIA H100 Tensor Core GPUs
GPU memory	640GB total
Performance	32 petaFLOPS FP8
NVIDIA® NVSwitch™	4x
System power usage	10.2kW max
CPU	Dual Intel® Xeon® Platinum 8480C Processors 112 Cores total, 2.00 GHz (Base), 3.80 GHz (Max Boost)
System memory	2TB
Networking	4x OSFP ports serving 8x single-port NVIDIA ConnectX-7 VPI > Up to 400Gb/s InfiniBand/Ethernet 2x dual-port QSFP112 NVIDIA ConnectX-7 VPI > Up to 400Gb/s InfiniBand/Ethernet
Management network	10Gb/s onboard NIC with RJ45 100Gb/s Ethernet NIC Host baseboard management controller (BMC) with RJ45
Storage	OS: 2x 1.92TB NVMe M.2
Internal storage:	8x 3.84TB NVMe U.2

500k USD / DGX H100
30k USD/Card
Almost a million H100s ordered for the next year

AI runs on GPUs

- AI = matrix multiplications, which is massively parallelizable
- GPUs are great at parallel programming
- CPU < 32 cores/threads, GPU> 4000 cores/threads!
- CPU is 10x slower, at least
- Impractical to train or even run any reasonable AI model outside GPUs and ASICs

AI is compute hungry

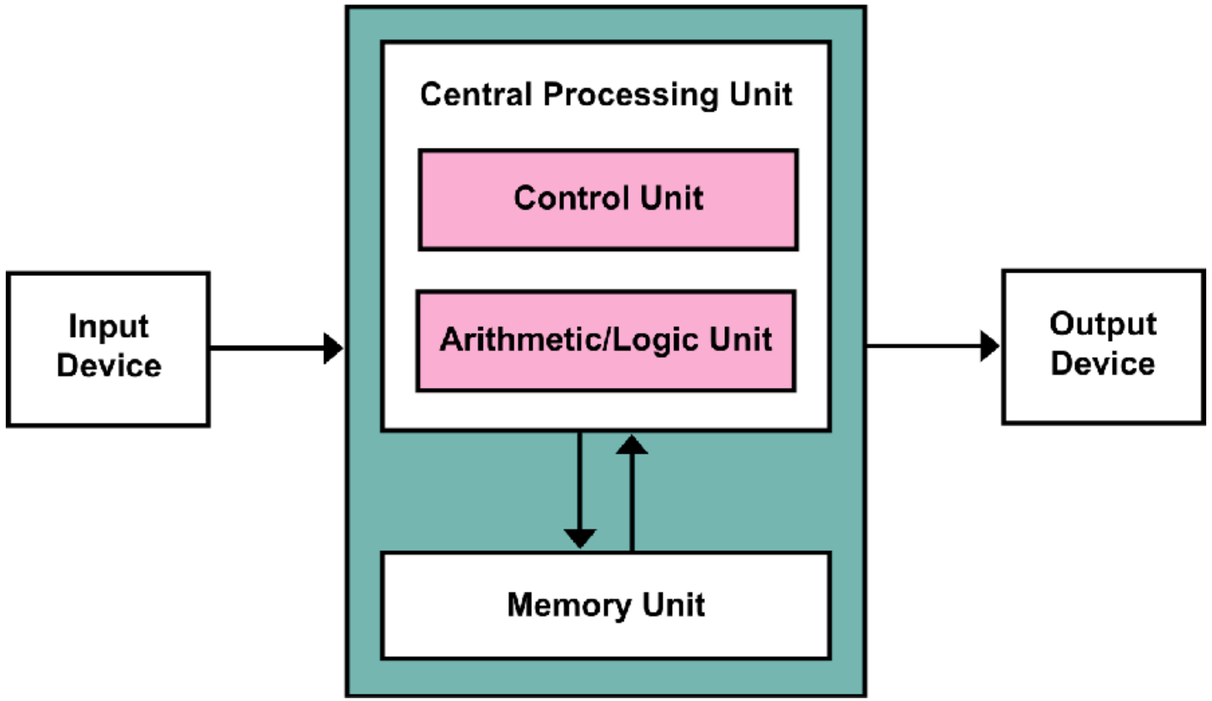


https://www.nature.com/articles/d41586-023-00641-w

Brief Review of Computer Architecture

Von Neumann Architecture

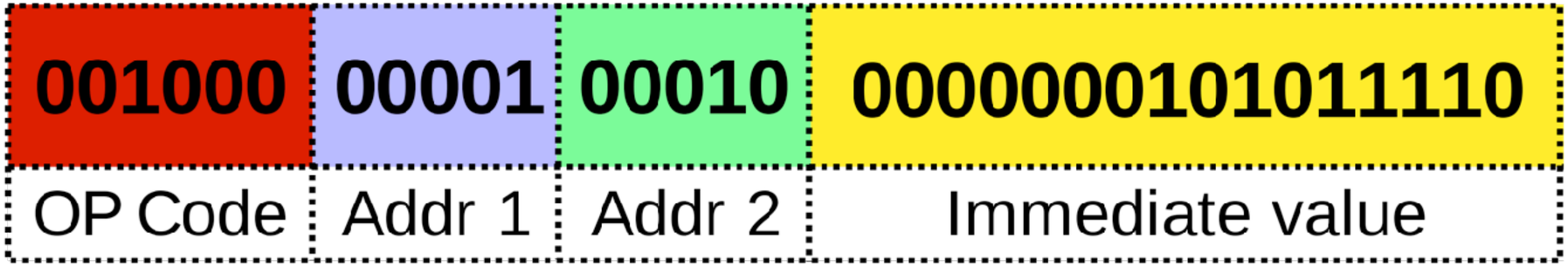
Memory has both data and instructions



Instruction Set

Language your CPU understands

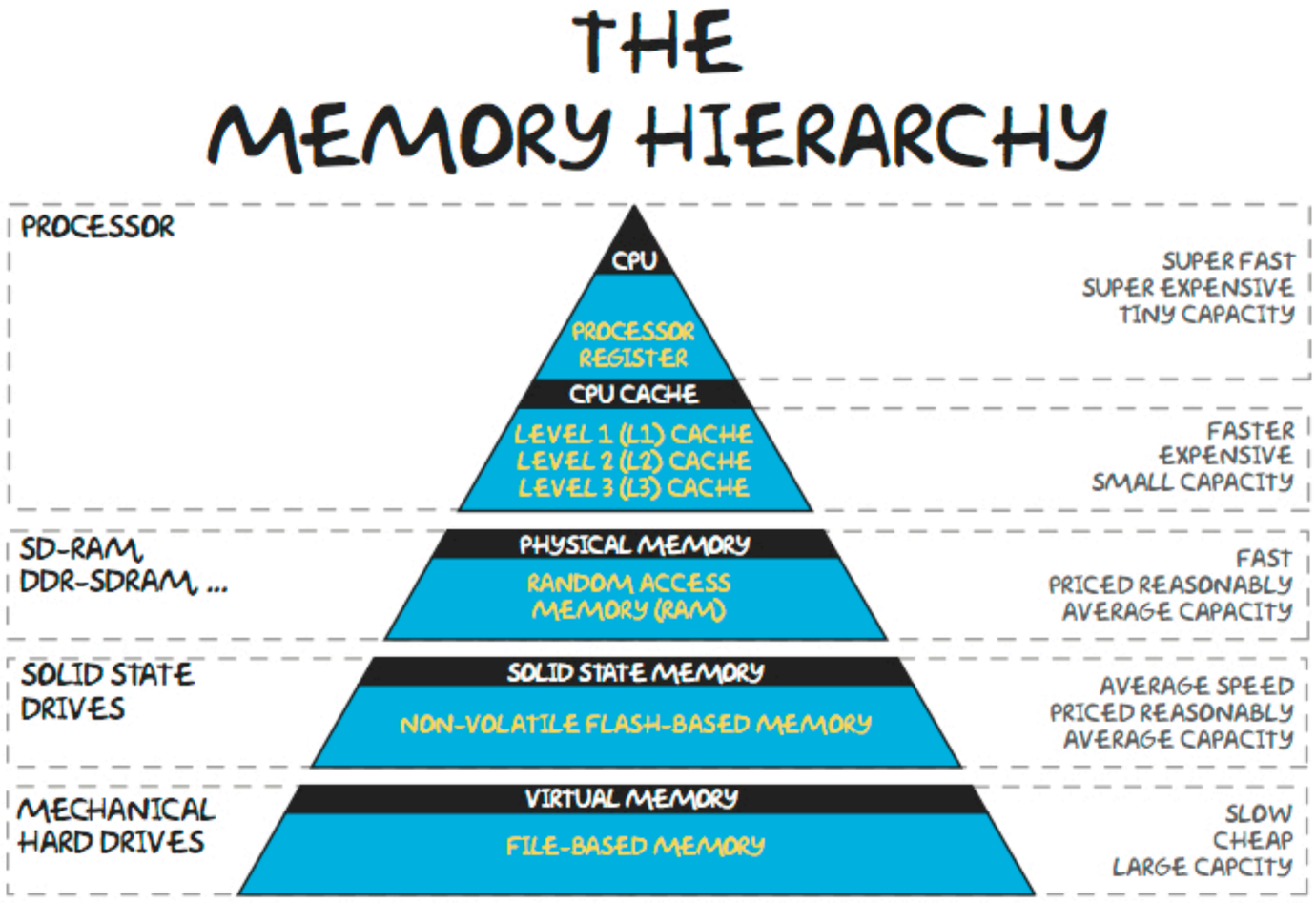
MIPS32Add Immediate Instruction



Equivalent mnemonic: **addi $r1, $r2, 350**

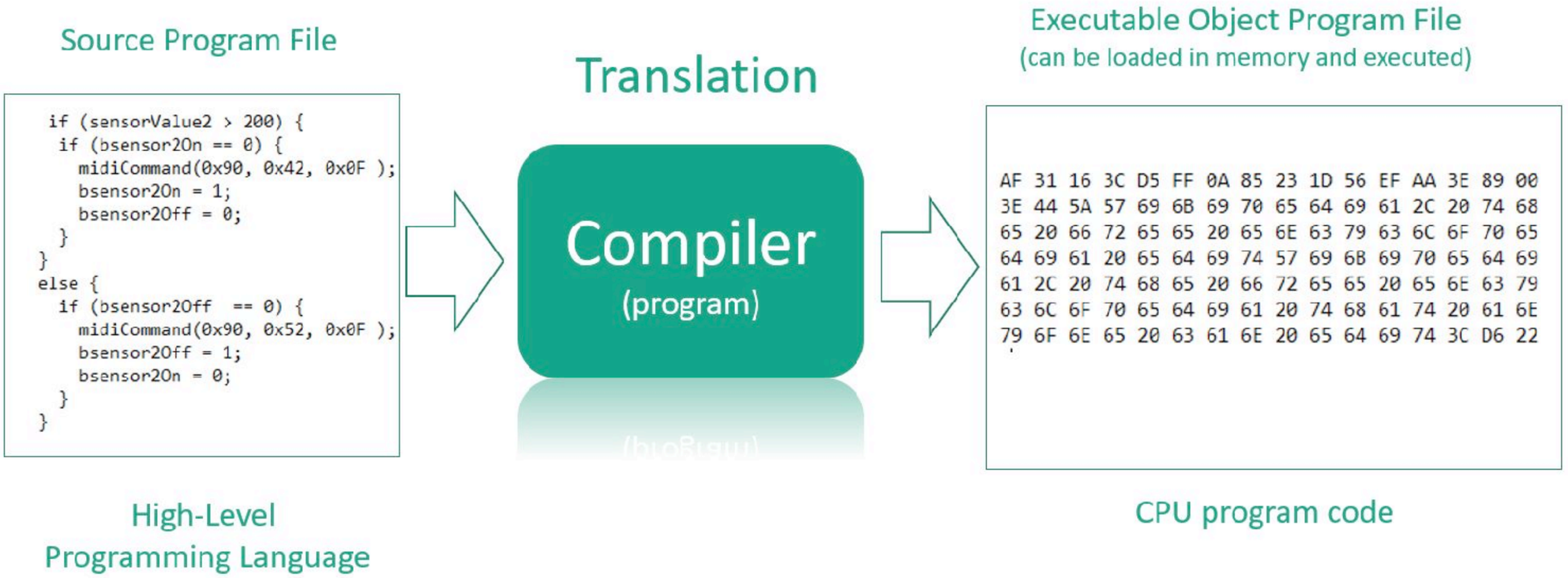
Memory Hierarchy

Because you have limited memory



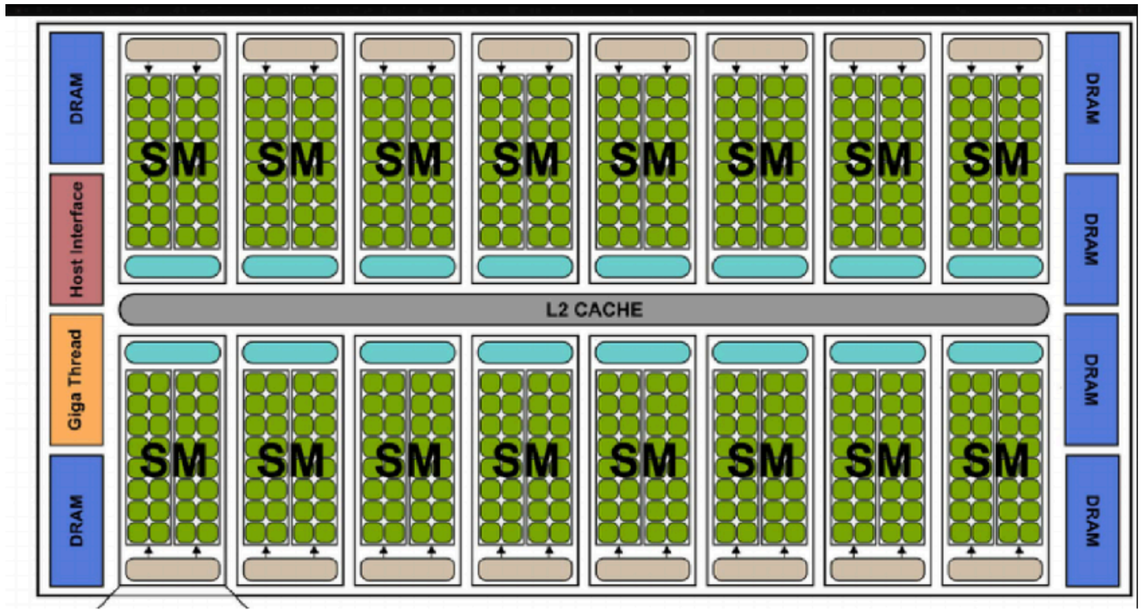
Compilers

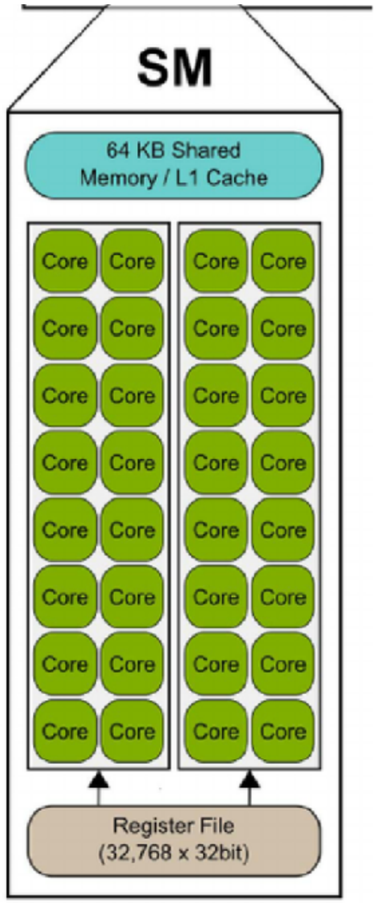
Convert high-level programming language to CPU

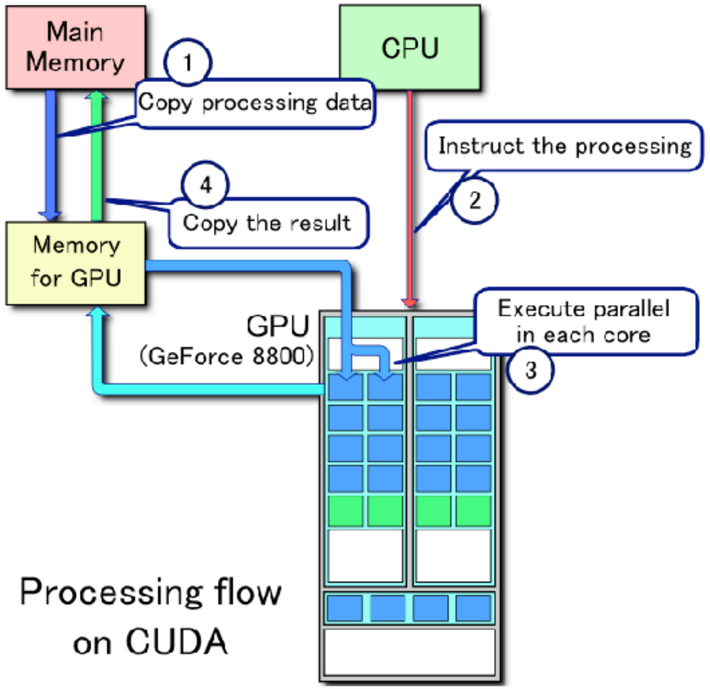


GPU Architecture

GPU = Multi core processors with hardware support for multi threading





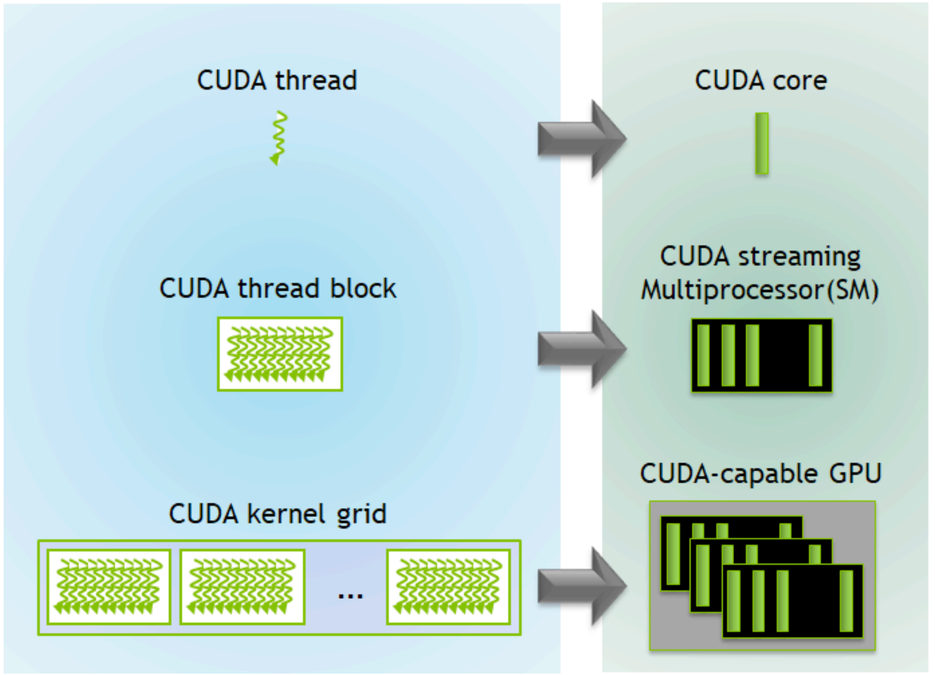


Note Memory Hierarchy

https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0061892

CUDA Programming Model

- Programming model mapped to architecture: threads, blocks and grid
- Memory hierarchy is mapped
- Shared Memory per SM (block) acts as configurable cache
- Global DRAM shared across the GPU



https://developer.nvidia.com/blog/even-easier-introduction-cuda/

Lisp

Original ‘AI’ Programming language

Dartmouth Workshop (1956)

Foundational Event for AI

PROPOSAL FOR RESEARCH BY JOHN MCCARTHY

During next year and during the Summer Research Project on Artificial Intelligence, I propose to study the relation of language to intelligence. It seems clear that the direct application of trial and error methods to the relation between sensory data and motor activity will not lead to any very complicated behavior. Rather it is necessary for the trial and error methods to be applied at a higher level of abstraction. The human mind apparently uses language as its means of handling complicated phenomena. The trial and error processes at a higher level frequently take the form of formulating conjectures and testing them. The English language has a number of properties which every formal language described so far lacks.

1. Arguments in English supplemented by informal mathematics can be concise.

2. English is universal in the sense that it can set up any other language within English and then use that language where it is appropriate.

3. The user of English can refer to himself in it and formulate statements regarding his progress in solving the problem he is working on.

4. In addition to rules of proof, English if completely formulated would have rules of conjecture.

The logical languages so far formulated have either been instruction lists to make computers carry out calculations specified in advance or else formalization of parts of mathematics. The latter have been constructed so as:

1. to be easily described in informal mathematics,

2. to allow translation of statements from informal mathematics into the language,

3. to make is easy to argue about whether proofs of (???)

No attempt has been made to make proofs in the artificial languages as short as informal proofs. It therefore seems to be desirable to attempt to construct an artificial language which a computer can be programmed to use on problems requiring conjecture and self-reference. It should correspond to English in the sense that short English statements about the given subject matter should have short correspondents in the language and so should short arguments or conjectural arguments. I hope to try to formulate a language having these properties and in addition to contain the notions of physical object, event, etc., with the hope that using this language it will be possible to program a machine to learn to play games well and do other tasks .

McCarthy Invents LISP

Key Features

- Lisp language was designed in 1960 by John McCarthy
- LISP = list processing. Everything is a list
- **Symbolic programming**: manipulate symbols rather than numbers
- **Recursive**: self-referential functions
- **Code is Data**: Manipulate code as if it is data
- **Macros:** Create your own syntax by ‘expanding’ macro
- Still in use today

Lisp Example Programs

```
> (+ 2 2) => 4

> (- (+ 9000 900 90 9) (+ 5000 500 50 5)) => 4444

> (append '(Pat Kim) '(Robin Sandy)) => (PAT KIM ROBIN SANDY)

> (defun last-name (name)
  "Select the last name from a name represented as a list."
  (first (last name)))
```

Many Basic Programming Concepts Were Invented

- Conditionals: if else
- First class functions: functions are also objects
- Garbage collection: leave it to another program to remove old data
- Led to object oriented programming
- Easy to extend and create new syntax: Lot of dialects such as scheme, common lisp etc

Early AI Programs

- **General Purpose Solver**: search algorithm to search for solution given ‘any’ problem
- **Eliza**: Original Chatbot! Simulate speaking with a psychoanalyst
- **MYCIN**: Expert system to diagnose an infection and recommend antibiotics
- **Automatic Theorem Prover**: As in name. Eg. type checking!
- Most of this is coded up in Lisp or related languages
- Used to be called ‘Symbolic AI’ or ‘Good Old Fashioned AI’

Symbolic AI vs Neural Networks

- Was dominant stream of AI
- So dominant that neural networks got relegated into no funding
- Geoff Hinton, Lecun and others persisted in believing in neural networks despite this
- Back propagation was the break-through for multi-layer neural networks
- Now Symbolic AI is relegated to no funding. Opportunity? :P

Lisp Chips

Lisp Implementations

- Back then, LISP was so resource intensive that most people couldn’t use it
- So, people tended to use Fortran/ALGOL instead more -> Later became C
- First implementation of lisp was on IBM704 - interpreter
- In 1962, compiler was written using above interpreter in lisp itself!

Lisp Chips

- Computer designed to use LISP as their main programming language
- ‘High-level language computer architecture’
- Basically CPUs with instruction set very similar to LISP
- Basically LISP Accelerators

Why relevant now?

- Moore’s law is ending
  - Exponential CPU speed up is no more guaranteed
  - Need special purpose hardware architectures like GPUs to speed up
- Design methodology is very interesting and is top down than bottom up
- Software-hardware co-design
- Design tools are also interesting
- GPU: CUDA :: LispChip : Lisp

Scheme-79 Chip

- Designed by a student for his course project, Guy Steele
- Instructor Gerry Sussman invented scheme - simplified version of Lisp
- He was also interested in computer aided design tools for engineers
- So they used very interesting methods to design this chip

How Scheme-79 Works

- Think of it as hardware interpreter of lisp
- Interpreter basically checks for current op and runs the operations recursively
- Machine language in scheme-79 is built using a typed pointer
- Lisp is converted to ’S-Code’ which is sort of machine language
- Type in pointer is similar to opcode

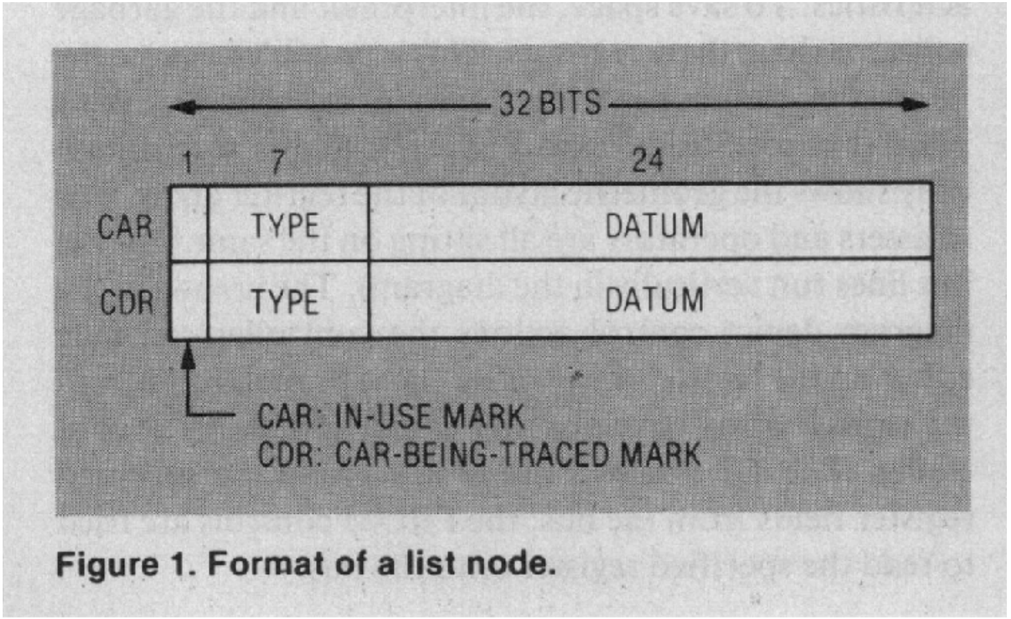


Figure 1. Format of a list node.

Separate garbage collector

- Data, programs and stack are allocated from heap memory
- Hardware interpreter is not interested in how memory is allocated
- But memory is finite - so we need garbage collection of useless objects
- So separate storage management system
- Implemented as separate state machine

Synthesis of the chip

Things get even more interesting

- Because lisp is so extensible, the design of the chip was also done in Lisp!
- Two languages/dialects were designed:
  - ‘Micro-lisp’ suitable for expressing low-level S-code
  - Layout language to actually ‘artworks’ for burning on chips using lithography
- A compiler was written to convert micro-lisp to layout.
- Essentially, use lisp to compile itself into hardware!

Interesting tidbits about Scheme-79

- There’s no ALU in scheme-79 chip! Can still do symbolic derivatives!
- Still has some “arithmetic” using list addition and deletion as ‘number system’
- Garbage collector is basically interface b/w this arithmetic and standard ordered numbers of memory
- Interesting way to look at this architecture: everything above a boundary is a processor which treats everything below boundary as a memory system
- Layered approach to chip design?

Fun Project

- Implement scheme-79 in a modern FPGA?
- Use CIRCT/Calyx in place of layouts to generate verilog
- Use MLIR sort of approach for transforming b/w representations