Hardware-Software Codesign

Thomas Bourgeat - POCS - Fall 2025 - EPFL

Outline

Some historical perspective on the HW/SW dance.

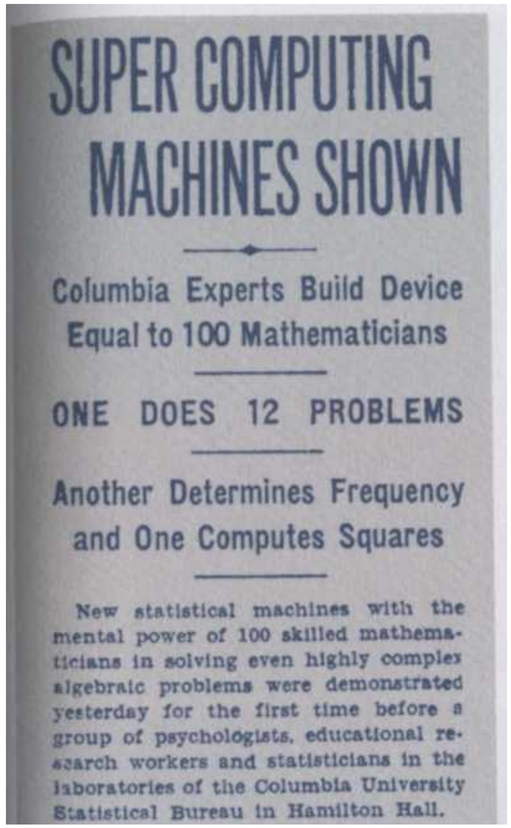
Cost of Abstraction - Value of Specialization

Canonical Hw/Sw Co-Design

Dally’s Principle of Accelerators

Hardware/Human Codesign

Pre-Turing Era - 1930s



SUPER COMPUTING MACHINES SHOWN

Columbia Experts Build Device Equal to 100 Mathematicians

ONE DOES 12 PROBLEMS

Another Determines Frequency and One Computes Squares

New statistical machines with the mental power of 100 skilled mathematicians in solving even highly complex algebraic problems were demonstrated yesterday for the first time before a group of psychologists, educational research workers and statisticians in the laboratories of the Columbia University Statistical Bureau in Hamilton Hall.

Fancy tabulating machine could High Value Computations: $\sum_i x_i^2$ or even $\sum_i w_i . x_i$

Technology was electromechanical relays

Typically, quite a few “telescoping" (pipelining) tricks (example)

Humans interact with the kernel performed by the machine:

Machine Input: Feed it punchcards with data

Machine Outputs:

- New punchcards (write once medium, for partially accumulated data)
- Human readable summary on fan-fold paper

Cool computations: SELECT/WHERE/GROUP BY !

Performance: Could ingest 150 punch cards per minutes

Source: www.columbia.edu/cu/computinghistory/packard.html

1945 - Vacuum Tube's Era

Von Neumann's magic - Death of hardware, birth of Software

Reports on Building an "Automatic Computing Device" (C[entral], M[emory], R[ecording] (for IO))

> The orders which are received by CC come from M, i.e. from the same place where the numerical material is stored.

Von Neumann's Insight/Suggestion (Mechanical - ms, Vacuum tube - us)

> Thus it seems worthwhile to consider the following viewpoint: The device should be as simple as possible, that is, contain as few elements as possible. This can be achieved by never performing two operations simultaneously, if this would cause a significant increase in the number of elements required.

> It is also worth emphasizing that up to now all thinking about high speed digital computing devices has tended in the opposite direction: Towards acceleration by telescoping processes at the price of multiplying the number of elements required. It would therefore seem to be more instructive to try to think out as completely as possible the opposite viewpoint

(https://web.mit.edu/STS.035/www/PDFs/edvac.pdf)

Very rich paper: why binary is best, thoughts about errors in computing, JIT, biomimetics, brain VS computer, synchrony/asynchrony …

Feynman’s observation

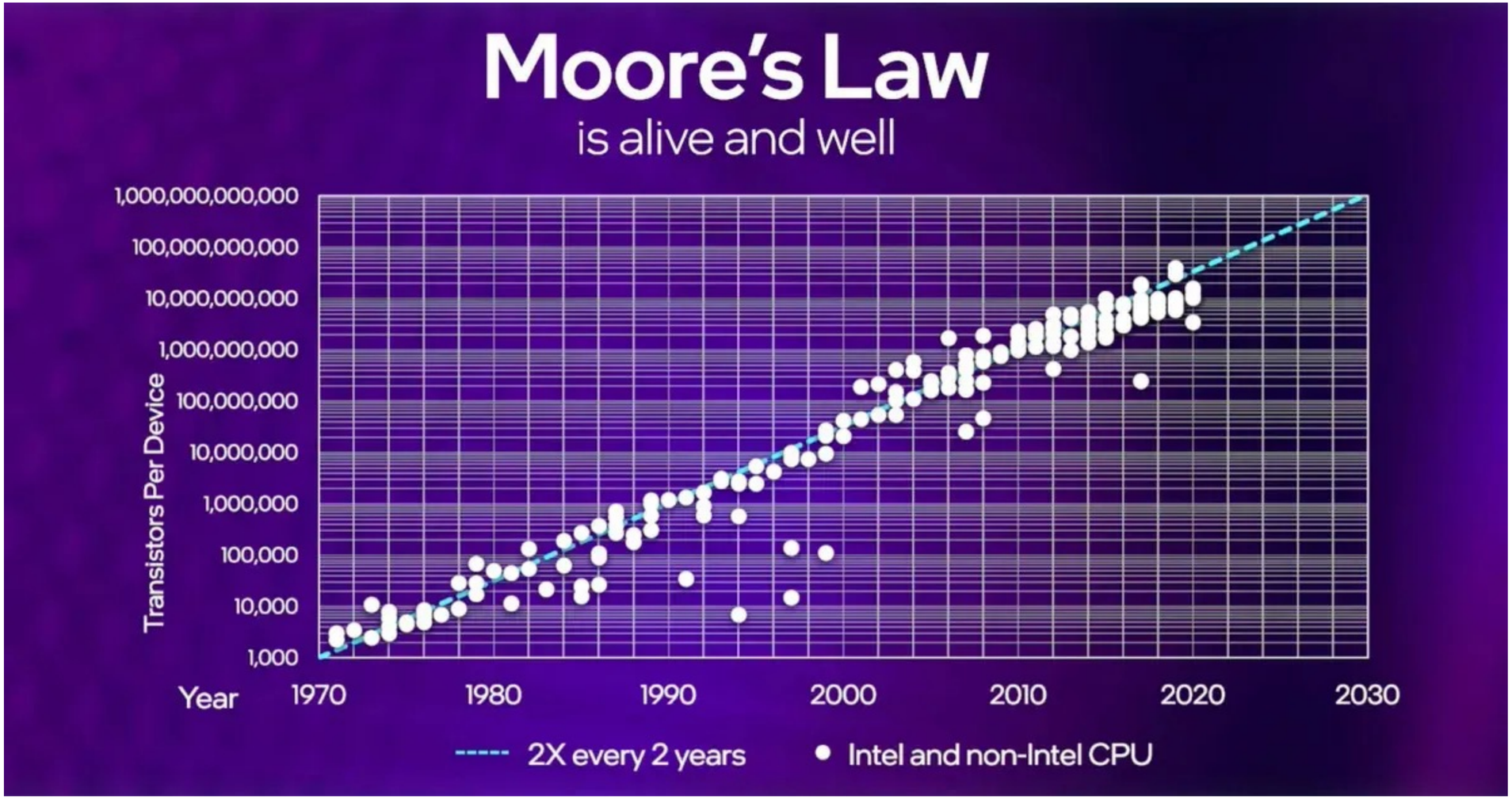
Plenty of Room at the Bottom

1959 Feynman mentally explores the future of miniaturisation:

> In the future (present) we should be able to do insanely small machines!
>
> Transistor Free Lunch Party!
>
> Smaller transistor ~ faster clock ~ no effort, same design go faster
>
> Smaller transistor ~ more transistors, what do we do with them?

https://web.pa.msu.edu/people/yang/RFeynman_plentySpace.pdf

Feynman was right



The Hard Floor: Landauer Limit

Moore's Law is an economic observation.

Landauer's Principle is a physical law

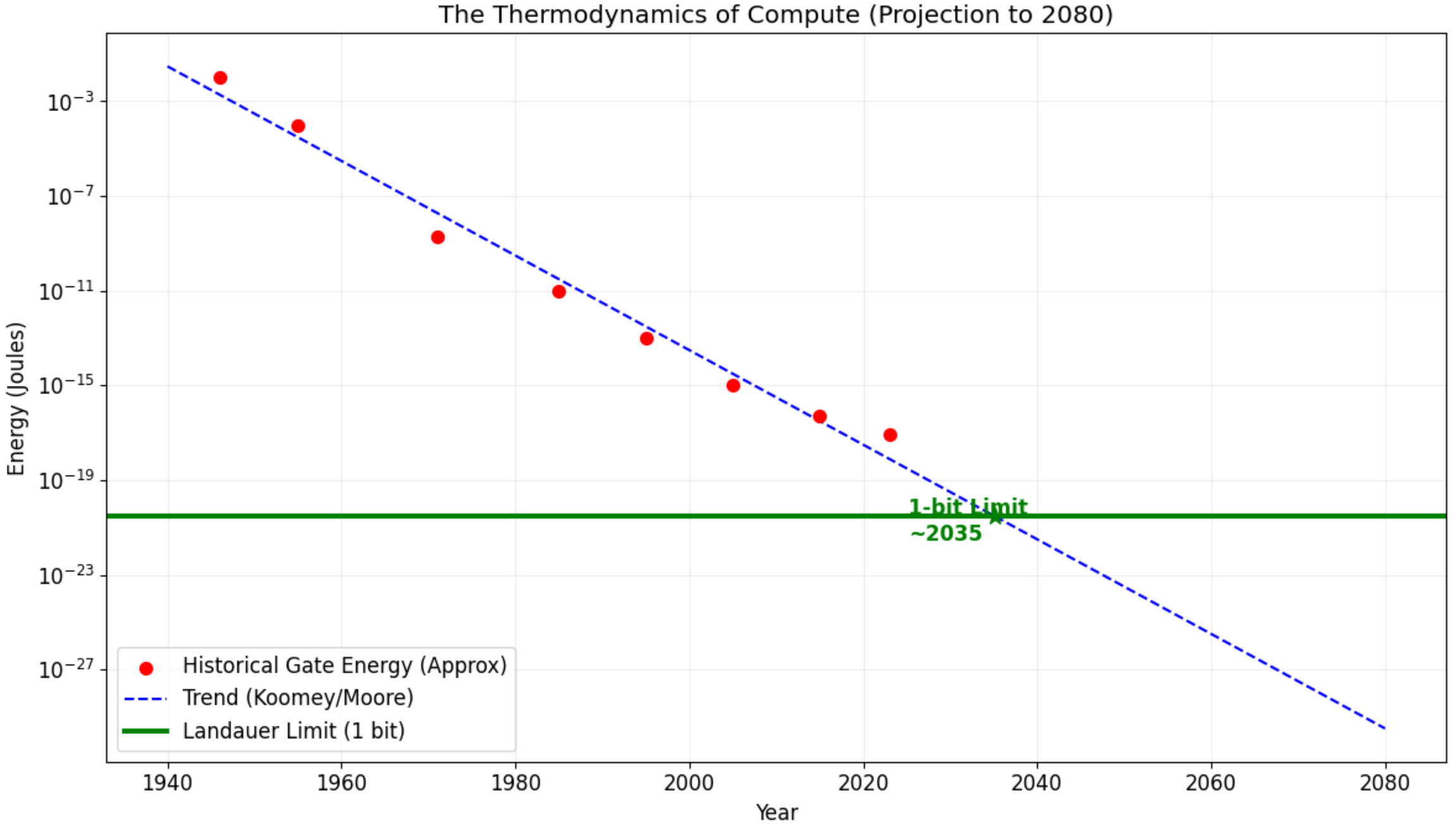
(Feynman actually derives it in his Lectures on Computation)

The Law: The minimum energy required to erase one bit of information is:

$$kT \ln 2$$

The Numbers:

At room temp **min energy to erase 1 bit** ≈ 2.8 × 10-21 J.


The Thermodynamics of Compute (Projection to 2080)
Energy (Joules)
10^{-3}
10^{-7}
10^{-11}
10^{-15}
10^{-19}
10^{-23}
10^{-27}
1940
1960
1980
2000
2020
2040
2060
2080
Year
1-bit Limit
~2035
Historical Gate Energy (Approx)
Trend (Koomey/Moore)
Landauer Limit (1 bit)

What will happen?

Hypothesis 1: Landauer’s limit is wrong

Hypothesis 2: We will just cool down our computers better and our phone will be running at 100K, 10K, 1K, 0.1K…

Hypothesis 3: Reversible Computing: Quantum/Adiabatic

Hypothesis 4: Computing systems stop riding an exponential, become like many other engineering and scientific discipline

There was plenty of room at the bottom, maybe now there is plenty of room at the top?

Mid 2010s, after 70 years of free lunch: Let’s assess the bloat

Cost of Abstraction

There is Plenty of Room at The Top

Table 1. Speedups from performance engineering a program that multiplies two 4096-by-4096 matrices. Each version represents a successive refinement of the original Python code. "Running time" is the running time of the version. "GFLOPS" is the billions of 64-bit floating-point operations per second that the version executes. "Absolute speedup" is time relative to Python, and "relative speedup," which we show with an additional digit of precision, is time relative to the preceding line. "Fraction of peak" is GFLOPS relative to the computer's peak 835 GFLOPS. See Methods for more details.

Version	Implementation	Running time (s)	GFLOPS	Absolute speedup	Relative speedup	Fraction of peak (%)
1	Python	25,552.48	0.005	1	—	0.00
2	Java	2,372.68	0.058	11	10.8	0.01
3	C	542.67	0.253	47	4.4	0.03
4	Parallel loops	69.80	1.969	366	7.8	0.24
5	Parallel divide and conquer	3.80	36.180	6,727	18.4	4.33
6	plus vectorization	1.10	124.914	23,224	3.5	14.96
7	plus AVX intrinsics	0.41	337.812	62,806	2.7	40.45
	TPU v1 (2016)	<0.001	~90,000.000	~15,000,000	~270	100% (???)

Source: https://www.microsoft.com/en-us/research/uploads/prod/2020/11/Leiserson-et-al-Theres-plenty-of-room-at-the-top.pdf

TPUv1/Processor: 28/22nm, ~350mm2, ~700MHz/~3GHZ

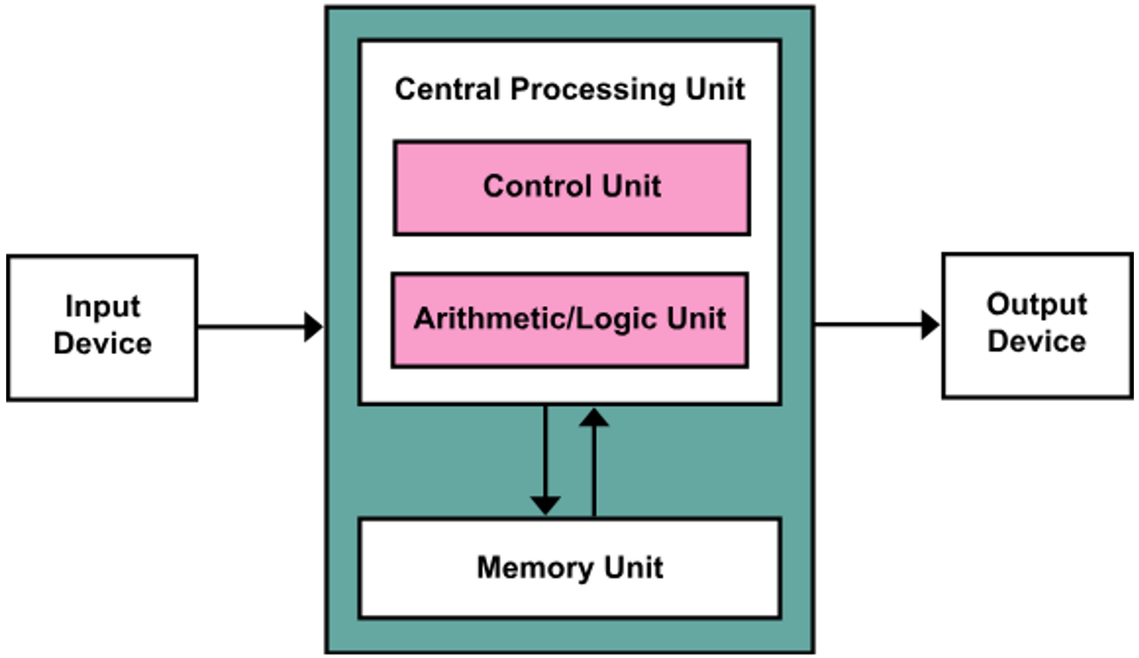
Von Neumann's suggestion of a simple, fully general, and programmable machine is dead.

-

Before we get to accelerators and co-design, let' s understand why

5.6 Accelerating these arithmetical operations does therefore not seem necessary —at least not until we have become thoroughly and practically familiar with the use of very high speed devices of this kind, and also properly understood and started to exploit the entirely new possibilities for numerical treatment of complicated problems which they open up. Furthermore it seems questionable whether

The Danse Between Memory and Processor



First Insight: "It's the Memory, Stupid"

Richard Sites, 1996 (DEC Alpha Architect)

“across the industry, today's chips are largely able to execute code faster than we can feed them with instructions and data.”

In other words, high-performance microprocessors are often high-speed no-ops machine.

Little's Law: How to Balance A System

Little's Law is a fundamental theorem from queuing theory that relates three variables in a stationary system:

$$L = \lambda \times W$$

L (Concurrency): The amount of data "in flight" (Queue Length).

λ (Throughput): The rate of data delivery (Bandwidth).

W (Wait Time): The round-trip time for a request (Latency).

To achieve a target Bandwidth (λ), the system must maintain sufficient Concurrency (L) to cover the Latency (W). If you don't have enough requests in flight, your throughput cannot reach the peak bandwidth.

Little’s Law In Practice

Memory viewpoint

System: 100 GB/s BW, 100 ns Latency.

Requirement: You need 10,000 Bytes of active requests (Processor needs to emit many concurrent requests) at all times to stop the bus from going idle.

Evaluating the Speed of Light

Hw/Sw Co-Design from First Principle

Processor vs. Memory

At a hardware level, two distinct operations are constantly (potentially simultaneously) costing time:

Compute (Work): The processor/computing unit executing operations (FLOPs) on data present in registers/caches.

Memory Traffic (Movement): Moving bytes between DRAM (Memory) and the Processor.

The core question of efficiency is:

Once the processor pays the 'cost' to fetch a chunk of data, how much 'work' can the processor do with it before it needs to fetch more?

Arithmetic Intensity

Arithmetic Intensity is the ratio of total floating-point operations performed to total data bytes moved.

$$\text{AI} = \frac{\textit{Total FLOPs}}{\textit{Total Bytes Accessed}}$$

This is an **average** metric over a specific kernel or time window. It aggregates all math done and divides it by all memory traffic incurred during that period.

Examples of Intensity

Vector Addition: A[i] + B[i]

You load 2 numbers and store 1 result for just 1 addition. AI is $\sim\frac{m}{2m} = O(1)$

The processor is constantly asking for more data,

Low FLOPs/Byte (Memory Bound)

Matrix Multiplication: C = A x B

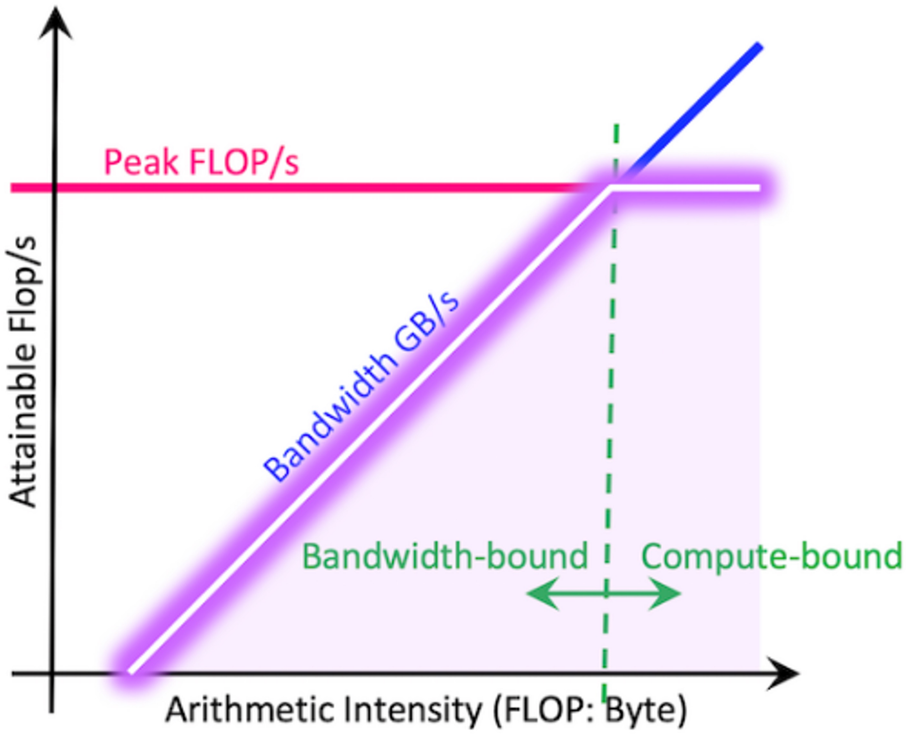
If the full matrices fit in the local memory/registers, AI is $\sim\frac{m^3}{2m^2} = O(m)$

The high reuse rate drives up the average work per byte.

High FLOPs/Byte (Compute Bound)

The Roofline Model

Map of Speed of Light

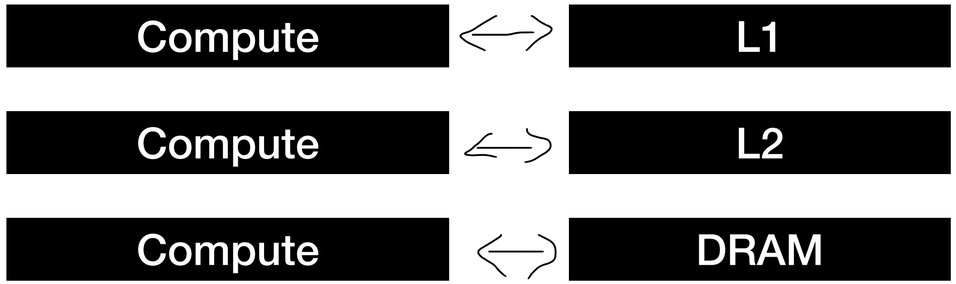


1. What is the Peak FLOP/s your hardware can achieve
2. What is the peak Bandwidth your hardware can achieve?
3. What is the arithmetic intensity of your problem?

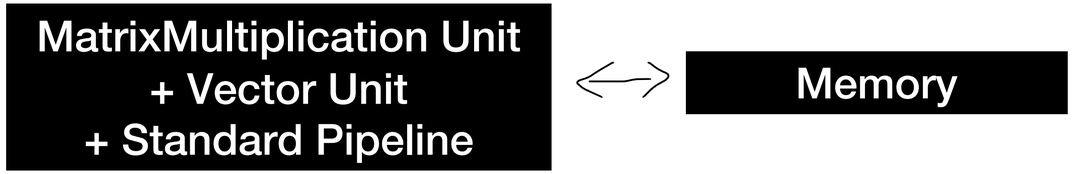
We will focus on the first two questions

Roofline Complexity

- What is the peak Bandwidth your hardware can achieve?
- Different answers!



- What is the Peak FLOP/s your hardware can achieve?



Things look simple? 1024x1024 Int8 matmul

What is the AI

In a naive implementation, a matmul must read an entire row of Matrix A and an entire column of Matrix B directly from the main memory to compute one element:

Read 2*(1024 + 1024) bytes for each output: 2*2048*1024^2 bytes total

Compute 1024^3 multiplications (if we don’t count additions)

This gives 0.25 FLOPs/byte

But if we tile with a size of tile of 128, we only count moving the data from main memory into the cache (where we assume it can fit)?

Exercise left to the student

Arithmetic Intensity is not a property of a problem, but a property of a specific implementation (on a specific machine).

One must carefully tile the problems to increase reuse in fast memory to increase arithmetic intensity and reach the roof

To make things harder

In Von Neumann's model of computing, moving things to and from memory … utilizes Ops!!

Ld and St instructions

*We need to **compute** to be allowed to compute more, but due to the fact that we compute, we loose some compute resources.* 😖

In other words, the Peak FLOP/s cannot be attained because we need to use "FLOPs" of the processor to request data (not just waiting for it)!

A concrete example

Vanilla LLM Inference

The math of an LLM is :

Doing multiplications and additions

Doing a few softmax (exp), and vector normalization (square root?)

How can we achieve speed-of-light on this kind of workload?

Roofline and Little's Law

Step 1: Build a high roof

Specialized units for matrix multiplication, and semi-specialized units for exp, square roots and the rest of the less common arithmetic

GPU Architecture	CUDA Cores (FP32)	Tensor Cores (FP8)	% FLOPs in Tensor
NVIDIA H100	~67 TFLOPS	~1,979 TFLOPS	**~96%**
RTX 4090	~83 TFLOPS	~330 TFLOPS	**~80%**

CUDA cores are semi-general purpose cores.

They are up to ~100x faster than a CPU for FLOP heavy

They are > ~100x slower on irregular, control-heavy code (running Scala)

Roofline and Little’s Law

Step 2: Build a big pipe to these units

Feature	NVIDIA A100
Architecture	Ampere
HBM Bandwidth	**2.03 TB/s** (HBM2e)
HBM Capacity	80 GB
L2 Cache Bandwidth	**~4.8 TB/s**
L2 Cache Capacity	40 MB
L1 Cache Bandwidth	**~19.5 TB/s**
L1 Cache Capacity	20.7 MB (192KB/SM)
NVLink/Fabric	**600 GB/s** (NVLink 3)
PCIe Bandwidth	64 GB/s (Gen 4)

Roofline and Little's Law

Step 3: Build little workers to move data around

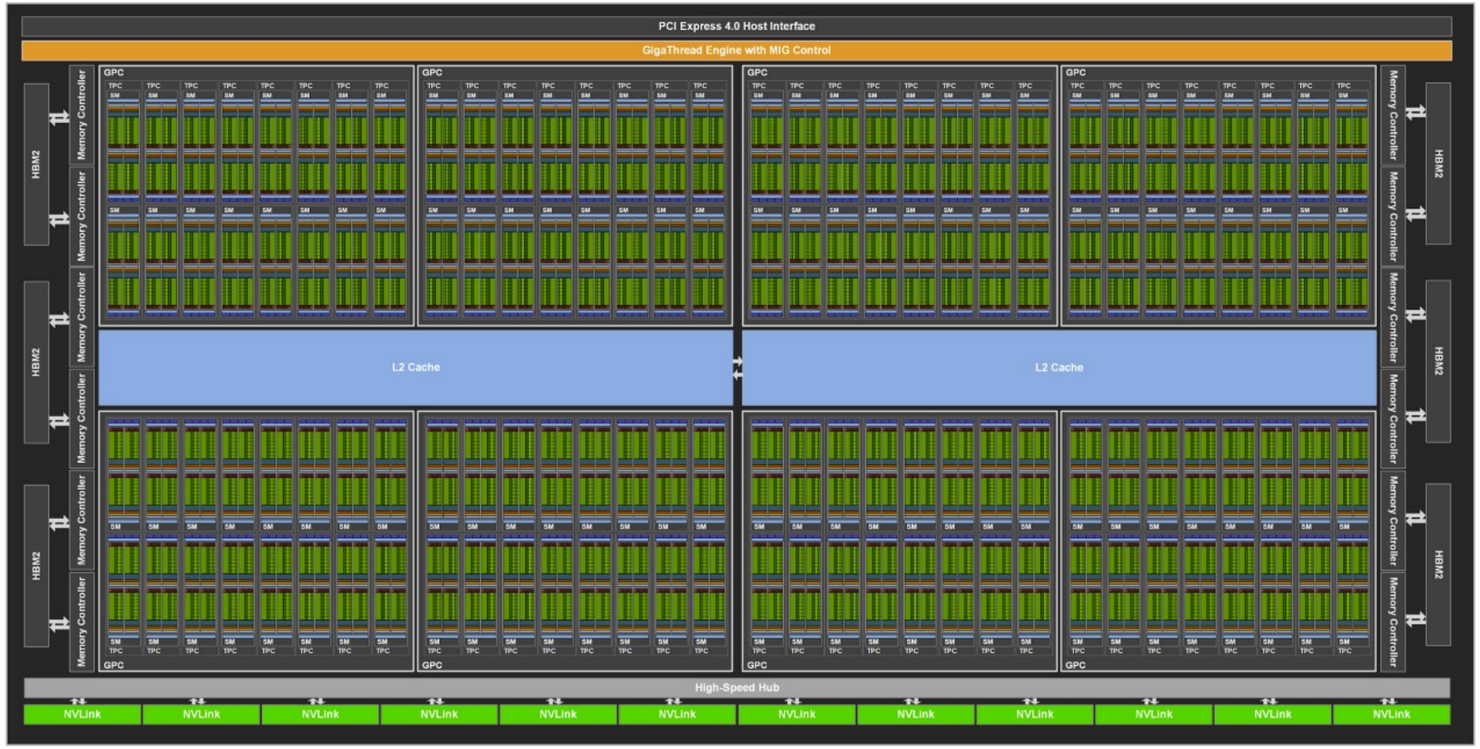
Maybe we could use semi-general purpose CUDA cores, but if we use them for that, we might not have enough FLOPs for EXP computations!

Step 4: Orchestrate all these players!

(Currently still using semi-general purpose CUDA cores, but for how long?)

Canonical Example of HW codesigns: a GPU

Figure 7. GA100 Streaming Multiprocessor (SM)



Step 5. Program the Machine 😢

An End-To-End Simple Example

A very simple pipeline:

- Someone upload a JPEG image (1024x1024) to a server
- We want to process it:
  - Decompress it
  - Apply a neural network to it

Principles of Accelerators

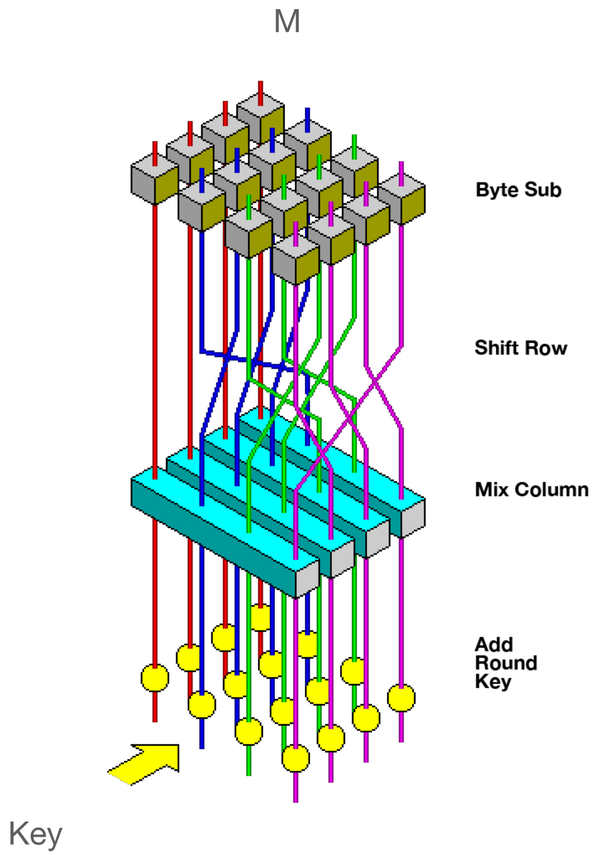
Guidelines for accelerators

(Borrowed from Bill Dally's talk)

1. Parallelism
2. Locality
3. Optimize Memory Orchestration and Control
4. Custom datatypes and operations

Parallelism

1. Look at the dataflow graph of your computation, at a fine granularity
2. What is the critical path of your computation, compared to the amount of compute nodes?
3. Even if there is a long chain of data dependencies, does it make sense to pipeline the computation?



Optimize memory orchestration and Control

Application specific “memory pipelining” / Decoupled execution

- Software orchestration can become the bottleneck (CPU spinning on memory location to wait an accelerator is done)

- Simply copying data around, decompressing data, are commonly bottlenecks

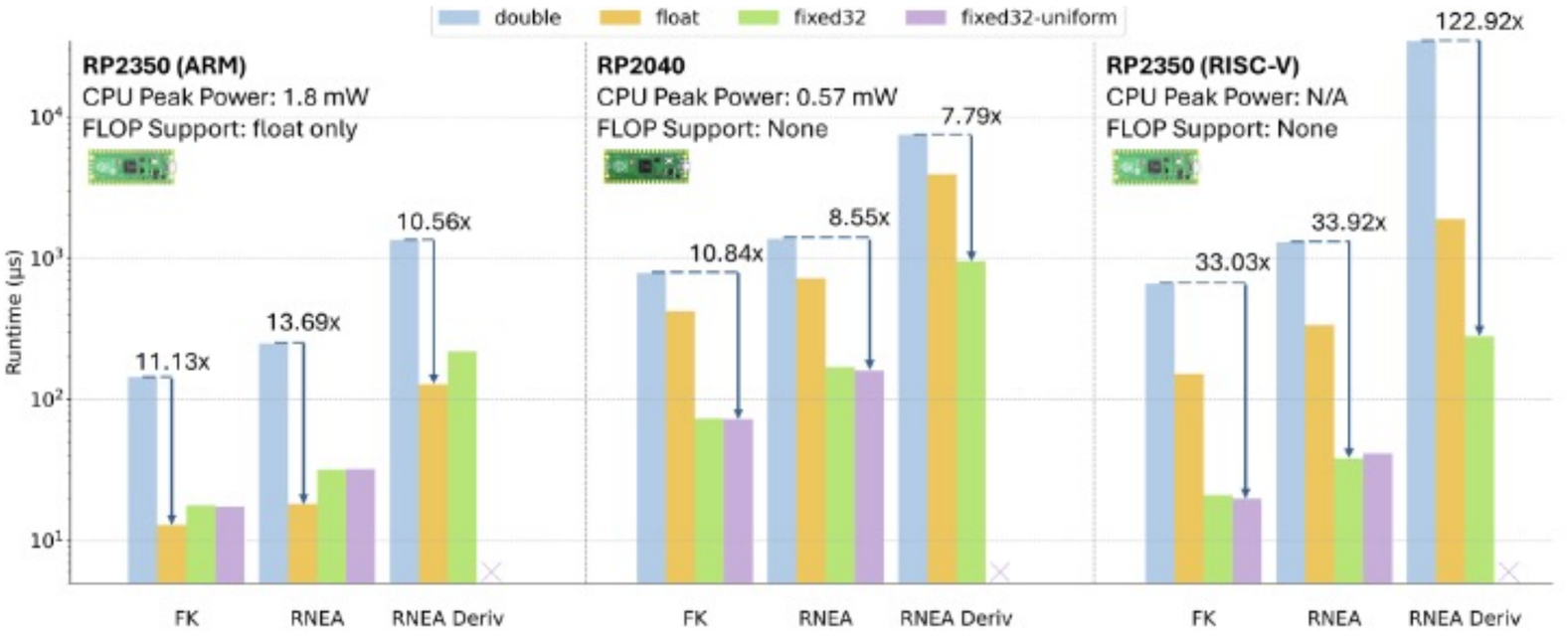
- The instructions of general-purpose machines can be bottlenecks as well

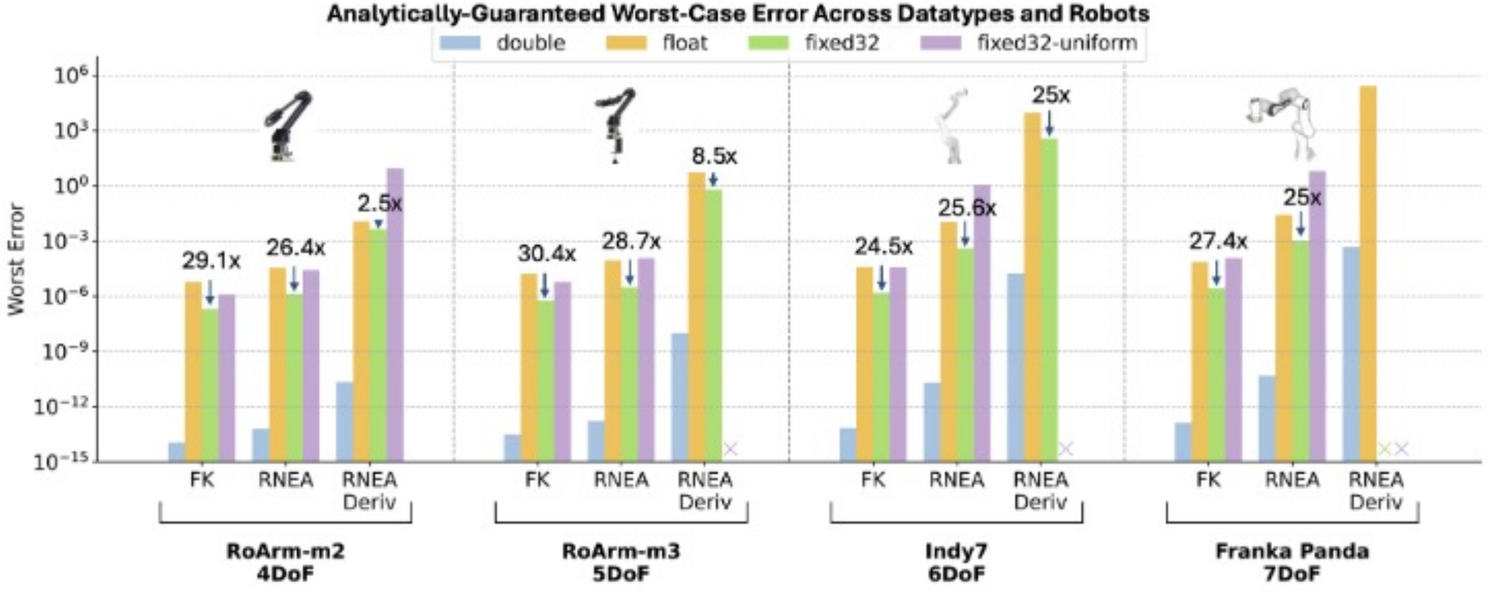
Specialized Datatypes

Domain specific analysis of structure of computation:

Modern ML -> lot to gain if reducing number of bits in representation

In Robotics, same opportunities? *(To appear, RA/L and ICRA),*


double
float
fixed32
fixed32-uniform
RP2350 (ARM)
CPU Peak Power: 1.8 mW
FLOP Support: float only
RP2040
CPU Peak Power: 0.57 mW
FLOP Support: None
RP2350 (RISC-V)
CPU Peak Power: N/A
FLOP Support: None
Runtime (µs)
11.13x
13.69x
10.56x
10.84x
8.55x
7.79x
33.03x
33.92x
122.92x
FK
RNEA
RNEA Deriv


Analytically-Guaranteed Worst-Case Error Across Datatypes and Robots
double
float
fixed32
fixed32-uniform
Worst Error
29.1x
26.4x
2.5x
30.4x
28.7x
8.5x
24.5x
25.6x
25x
27.4x
25x
FK
RNEA
RNEA Deriv
RoArm-m2
4DoF
RoArm-m3
5DoF
Indy7
6DoF
Franka Panda
7DoF

The Cost of Specialization

Remarks

Flame war between C/Java is a fight between 0.01% efficiency and 0.03% efficiency (This depends from program to program, compiler to compiler!)

Modern processors already embed a whole bunch of accelerators: Vector unit, often vastly underused

The size of employable workforce drastically decreases when going to harder-to-program machines

Software Integration

Having defined new instructions is not the end of the story:

- If I add AESENC, my libssl library won't suddenly start using it
- Have to worry about cross-platform

If I have matmul 16x16, or matmul (nxm forall n,m<64),

- 512x512 matmul?
- convolution?
- Arbitrary linear/tensor algebra operator

First drawback of HW/SW codesign

Take application X, handwrite code that leverage Accelerator W (V1), performs great

[…] Time elapse

10 years later, Accelerator W (V13)

-> Handwritten code performs poorly (when it works)

Second drawback of HW/SW codesign

Take application X, handwrite code that leverage Accelerator W (V1), performs great

[…] 3 months happens, DeepSeek releases DeepSeek Sparse Attention

Oops, completely bottleneck on an operation that I do not have accelerator for.

Hw/Sw co-designs are often performance fragile because driven by 1 high-value application

Compilation – Sync issues

Sync issues:

Need to add explicit data movement instructions if I want to use the data computed on CPU

Hot research ideas:

Decoupling functionality and scheduling/performance [Halide/Exo]

Maybe not a fully push-button compilation, more an “assistant-compiler”

Maybe enable user to augment the compiler - easily add transformations with triggers

Conclusion

“Accelerators" and Codesigns are not new - Yesterday they enabled census, today they are enabling AI

There are costs to abstraction

There are good guidelines/models for what can be accelerated

There are costs to specialization