Introduction to Computer Science

Computer Architecture: The Past & The Future

Mingyu Gao

gaomy@tsinghua.edu.cn

IIIS, Tsinghua

Tsinghua University

Oct 5, 2021

About Me: Mingyu Gao 高鸣宇

- Assistant Professor at Tsinghua University (2019.3. – now)
- Ph.D. from Stanford University (2012.9. – 2018.6.)
- B.S. from Tsinghua University (2008.9. – 2012.7.)

- Research: Computer Architecture 计算机系统结构
  - Domain-specific (e.g., AI) systems 领域专用加速
  - Memory systems 存储系统架构
  - Hardware security 硬件安全计算
  - ……

- More on my webpage: http://people.iiis.tsinghua.edu.cn/~gaomy/

Terminology & Vocabulary

- Von Neumann Architecture 冯诺依曼架构
- Arithmetic/logic unit (ALU) 算术逻辑单元
- Register file 寄存器堆
- Memory 内存
- Assembly code 汇编代码
- Machine code 机器码
- Instruction 指令
- Instruction set architecture (ISA) 指令集架构

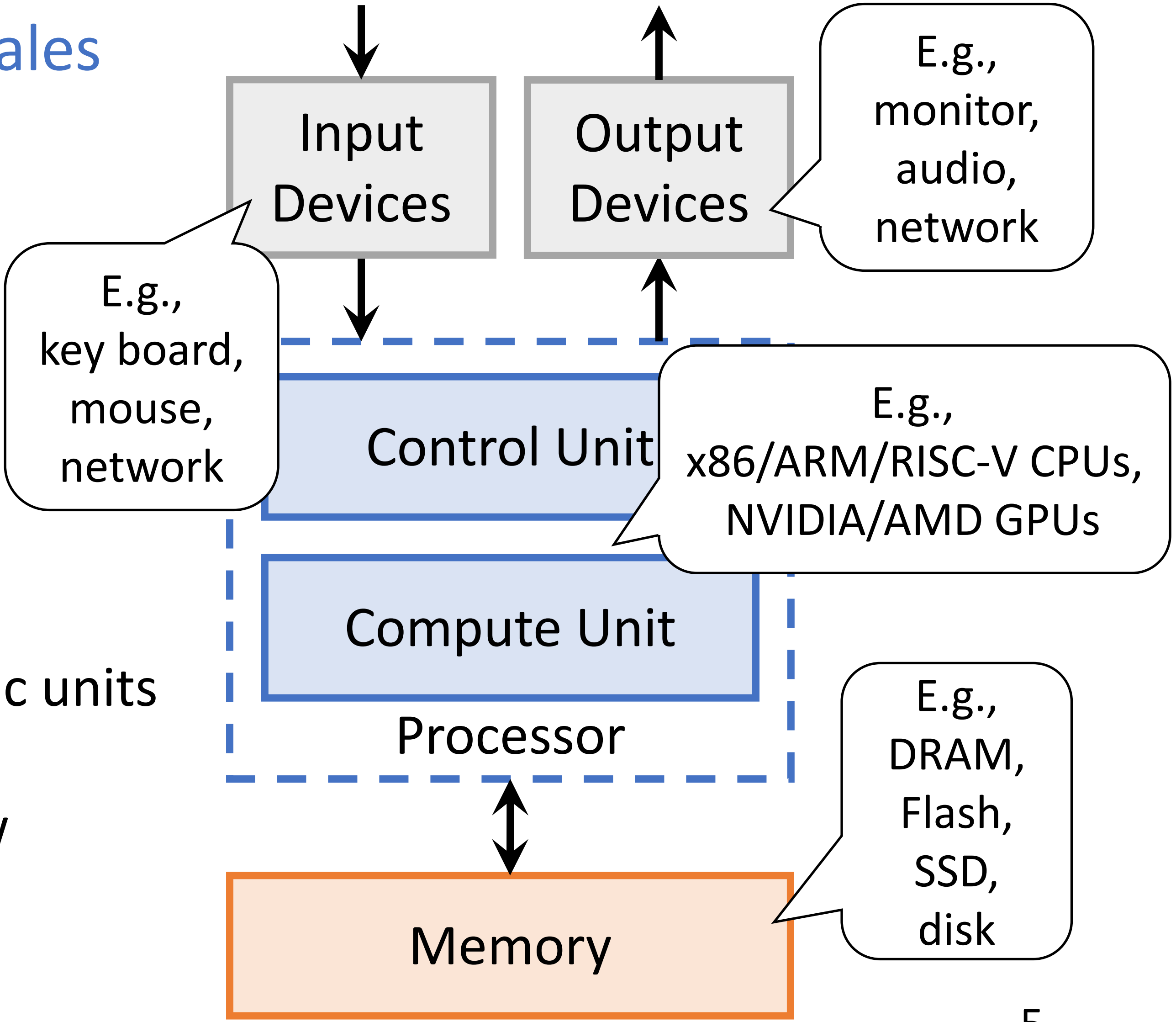
- Moore’s Law 摩尔定律
- Integrated Circuits 集成电路
- Transistor 晶体管
- Parallelism 并行
- Pipelining 流水线执行
- Superscalar 超标量
- Out-of-order 乱序执行Vector 矢量
- Multi-thread 多线程
- Bandwidth 带宽

Computer System Basics

How a computer works, in a *good* way

Typical Computer Systems Today

- Computer systems have diverse scales and abstraction levels
  - Mobile phones, laptops, desktop PCs, distributed clusters, datacenters, …
  - We study all of them
- Von Neumann Architecture
  - Memory: for both data and programs
  - Compute unit: include arithmetic/logic units and registers
  - Control unit: coordinate program flow
  - Input/output devices

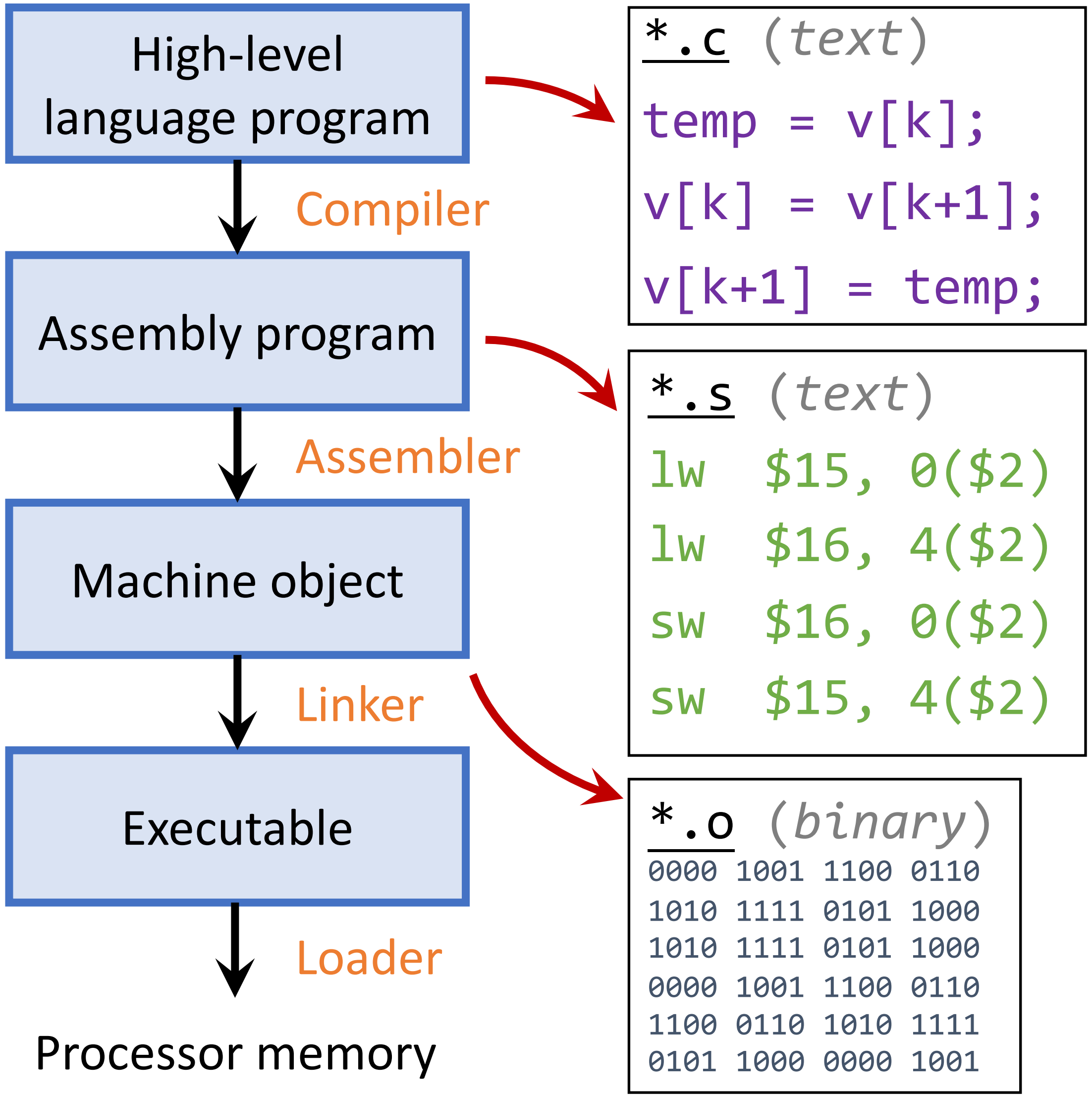

Input Devices
Output Devices
E.g., monitor, audio, network
E.g., key board, mouse, network
Control Unit
E.g., x86/ARM/RISC-V CPUs, NVIDIA/AMD GPUs
Compute Unit
Processor
Memory
E.g., DRAM, Flash, SSD, disk

How Computer Systems Work

- Computers work with binary signals, i.e., bits (0 and 1)
  - Everything is expressed as sequences of bits

- Program: sequence of instructions, encoded as strings of bits
  - ISA (instruction set architecture)
  - Example ISAs: x86, x86-64, ARM, RISC-V, MIPS, …
- Data storage (i.e., memory): cells preserve bits over time
  - Flip-flops, registers, SRAM, DRAM, …
- Data processing (i.e., compute & control): logic gates operate on bits
  - AND, OR, NOT, MUX, add, mult, …

Code Translation



High-level language program

Compiler

Assembly program

Assembler

Machine object

Linker

Executable

Loader

Processor memory

```
*.c (text)
temp = v[k];
v[k] = v[k+1];
v[k+1] = temp;
```

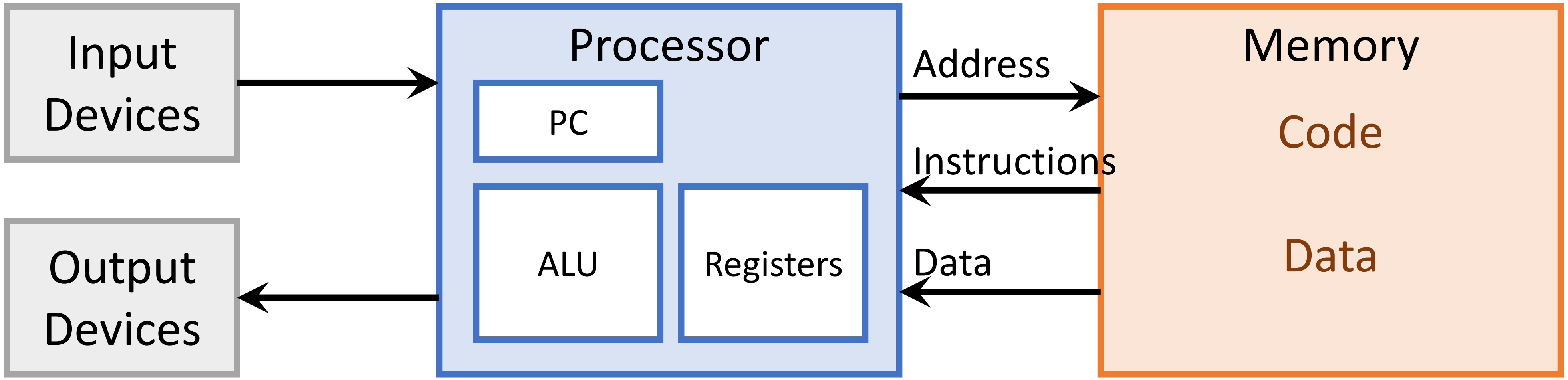
```
*.s (text)
lw  $15, 0($2)
lw  $16, 4($2)
sw  $16, 0($2)
sw  $15, 4($2)
```

```
*.o (binary)
0000 1001 1100 0110
1010 1111 0101 1000
1010 1111 0101 1000
0000 1001 1100 0110
1100 0110 1010 1111
0101 1000 0000 1001
```

- Machine code: the byte sequence that encodes program instructions
  - Actual 0's and 1's stored in the memory
- Assembly code: text representation of machine code
  - Human-readable instruction sequence
- Assembler
  - Binary encode each instruction
- Linker
  - Resolve inter-file references
  - Combine with static libraries
    - Some libraries are dynamically linked

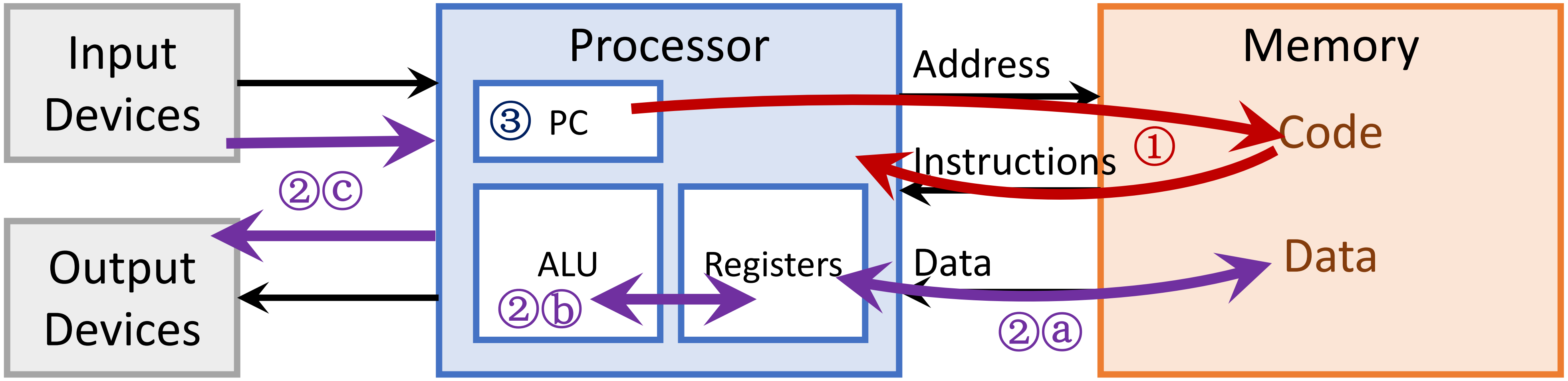
Assembly’s (Simplified) View of The System

- State: PC, registers, memory
  - Program counter (PC): points to the next instruction
    - Where we are in the middle of the instruction sequence
  - Registers (a.k.a., register file): local, heavily used data in the processor
  - Memory: code and data, both stored as blocks of bytes
- Arithmetic/logic unit (ALU): digital logic to do computation



How Instructions Execute

1) PC used as address to fetch an instruction from memory
2) The instruction indicates how to compute
   a) Transfer data between memory and processor registers
   b) Compute on register values and store results to registers
   c) Special ways to interact with outside world through I/O
3) Update PC to the next instruction, go back to 1)



What We Always Want

- All computer systems should work fast & efficiently

- High performance
  - Theorists invent algorithms by considering asymptotic behaviors (e.g., O(*nlogn*))
  - Architects set the constant factors
- Low cost, in terms of money, area, power, energy, design complexity, …
  - It is not hard to have high performance, but it is hard to do so with low cost

How did we achieve this goal before?

What could we do next to continue?

THE OLD GOOD DAYS

... when we have everything *scale* well

Moore’s Law

- Gordon E. Moore
  - Co-founder of Fairchild, later became CEO of Intel

- The original statement (1965)

  The complexity for minimum component costs has increased at a rate of roughly a factor of two per year

- The more well-known version

  The number of transistors per square inch on integrated circuits is doubling every year

  - In 1975, revised to *doubling every two years*

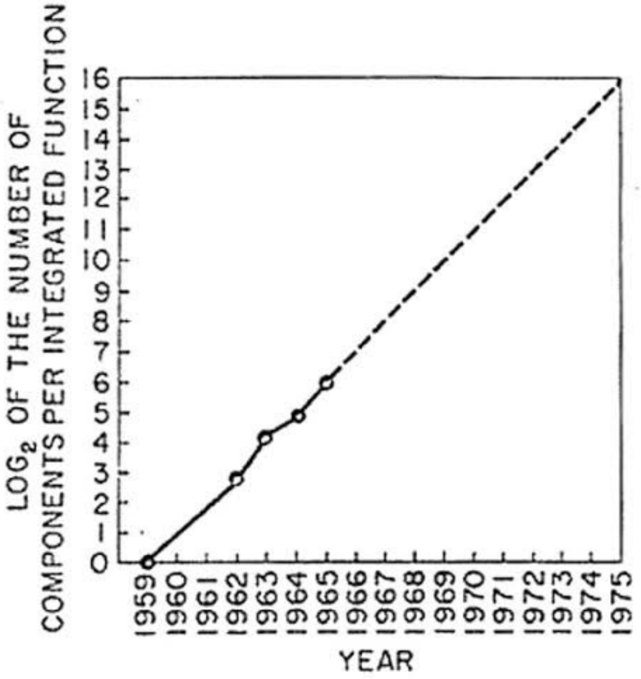


Fig. 2 Number of components per integrated function for minimum cost per component extrapolated vs time.

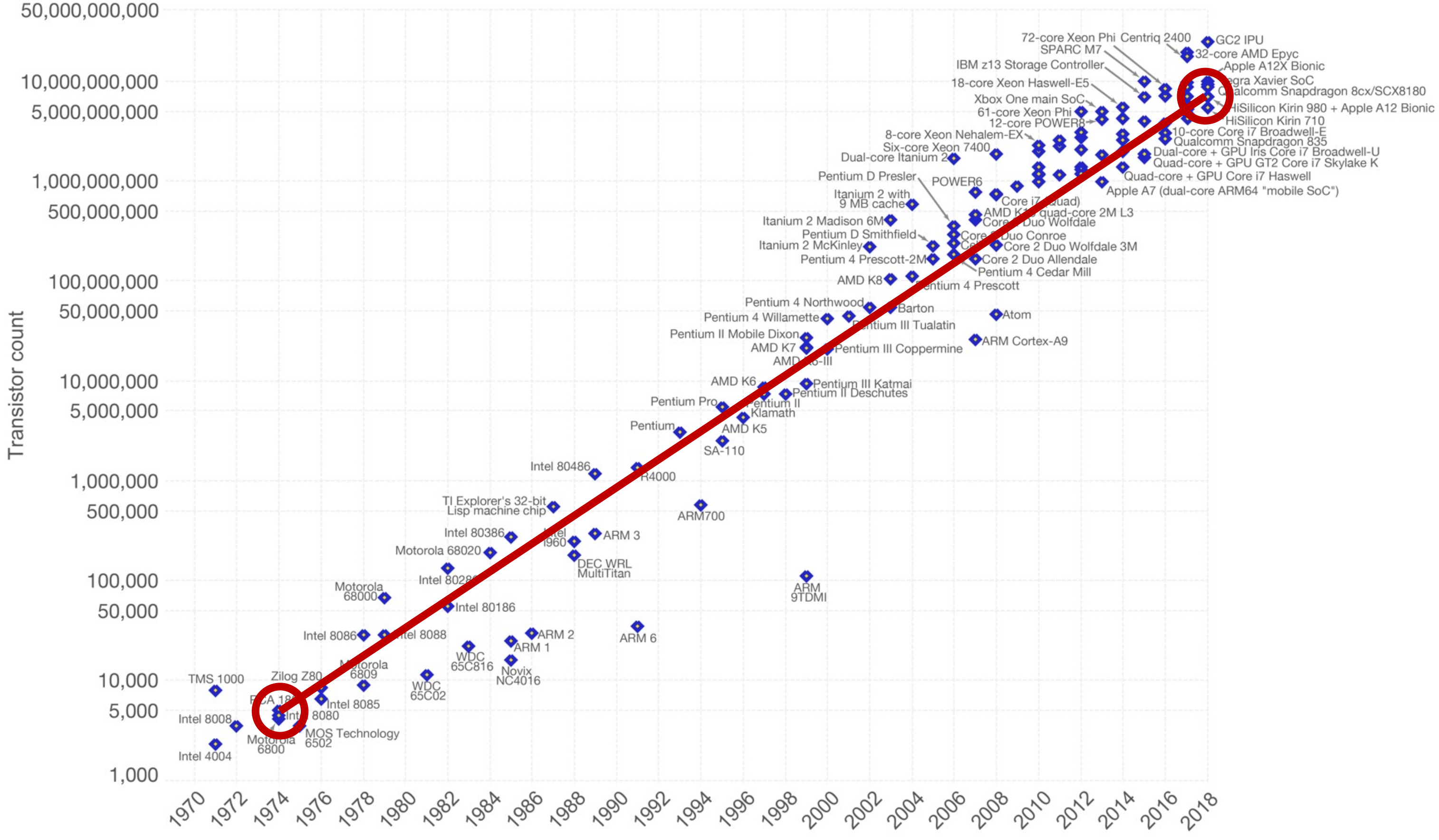
G. E. Moore. *Cramming More Components onto Integrated Circuits*. Electronics, Apr 1965.

How Is Moore’s Law Doing?

Moore’s Law – The number of transistors on integrated circuit chips (1971-2018)

Moore's law describes the empirical regularity that the number of transistors on integrated circuits doubles approximately every two years. This advancement is important as other aspects of technological progress – such as processing speed or the price of electronic products – are linked to Moore's law.

Our World in Data



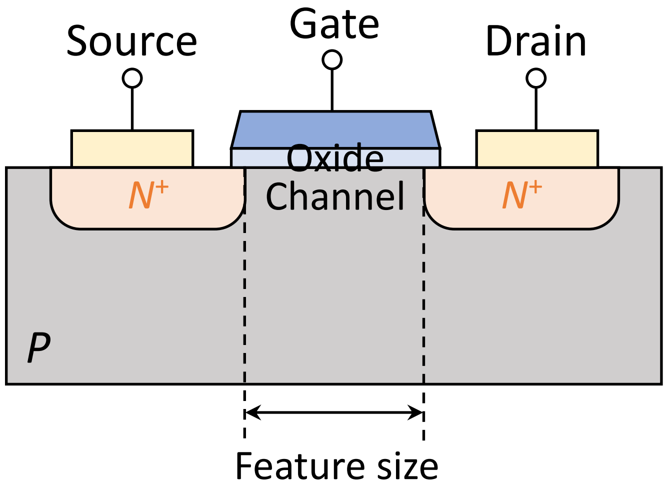
Data source: Wikipedia (https://en.wikipedia.org/wiki/Transistor_count)
The data visualization is available at OurWorldinData.org. There you find more visualizations and research on this topic.

- ❑ 1974, Intel 8080
  - ○ 6,000 transistors
  - ○ 20 mm^2
- ❑ 2018, HiSilicon Kirin 980
  - ○ 6,900,000,000 transistors
  - ○ 74.13 mm^2
- ❑ 310,000 × in 44 years
  - ○ 1.33 × per year
  - ○ About 1.7 × per two years
- ❑ Moore’s Law is still alive!

https://en.wikipedia.org/wiki/Moore%27s_law; https://en.wikipedia.org/wiki/Transistor_count

What is Happening Under the Hood

- Transistor
  - MOS: Metal-Oxide-Semiconductor
  - Digital switch: G controls the current from S to D
  - Use these 0/1 switches to build binary circuits



- Feature size: length of the transistor channel
  - Feature size shrinks in each technology generation (roughly 1.4× reduction)
    - 0.18 um → 0.13 um → 90 nm → 65 nm → 45 nm → 32 nm → 22 nm → 14 nm → 10 nm → 7 nm → ……
- Other dimensions shrink along with feature size with roughly same ratio
  - For planar area, 2× reduction per generation, enabling Moore's Law

Dennard Scaling

- Moore's Law is only about chip density scaling
  - If the transistor feature size scales by $1/S$ ($S \approx 1.4$), the chip density scales by S^2
- Performance and cost: Dennard Scaling (Robert H. Dennard, 1974)
  - If the feature size scales by $1/S$, the supply voltage and current can scale by $1/S$
- The free lunch in semiconductor industry!
  - 2.8x capability (2x transistors & 1.4x speed) with same area and same power
  - E.g., Intel Tick-Tock model

Feature Size	Area A	Capacitance C	Voltage V	Current I	Freq $f = I/CV$	Energy $E = CV^2$	Power $P = CV^2 f$	Pwr Density P/A
$1/S$	$1/S^2$	$1/S$	$1/S$	$1/S$	S	$1/S^3$	$1/S^2$	1

R. H. Dennard, et al. *Design of Ion-Implanted MOSFET's with Very Small Physical Dimensions*. JSSC, Oct 1974.

From More Transistors to Higher Performance

- Processor architectures have been evolving to utilize the increasing amount of transistors to offer higher performance
- By leveraging different levels of parallelism

- Bit-level parallelism
- Instruction-level parallelism
- Data-level parallelism
- Thread-level parallelism

Bit-Level Parallelism

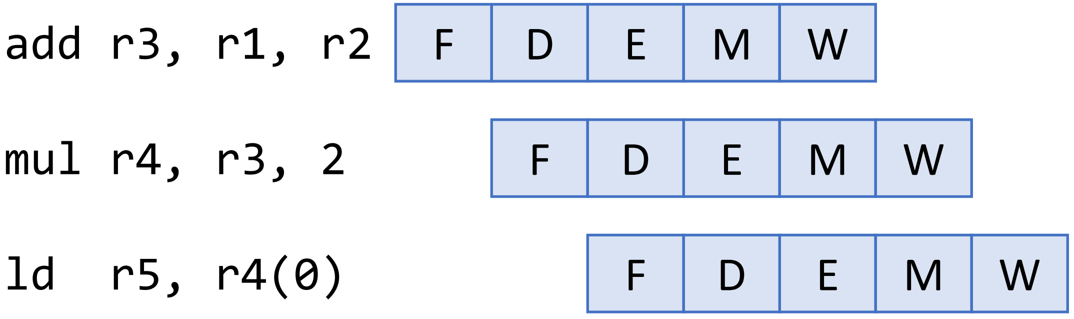
- From 1970s to mid 1980s, increase datapath bit width
  - 8-bit: 0 to 255, or −128 to 127
  - 16-bit: 0 to 65536, or −32768 to 32767
  - 32-bit: 0 to 4G, or −2G to 2G, large enough in most cases, except for memory
  - 64-bit: little benefit to further increase

Processor	Year	# Transistors	Bit width
Intel 4004	1971	2,250	4-bit
Intel 8008	1972	3,500	8-bit
Intel 8086	1978	29,000	16-bit
Intel 80386	1985	275,000	32-bit

- Reduce number of cycles required to perform arithmetic operations
  - E.g., multiply two 32-bit numbers on a 16-bit processor
    - Four multiplies ($A_H \times B_H$, $A_H \times B_L$, $A_L \times B_H$, $A_L \times B_L$), followed by several adds

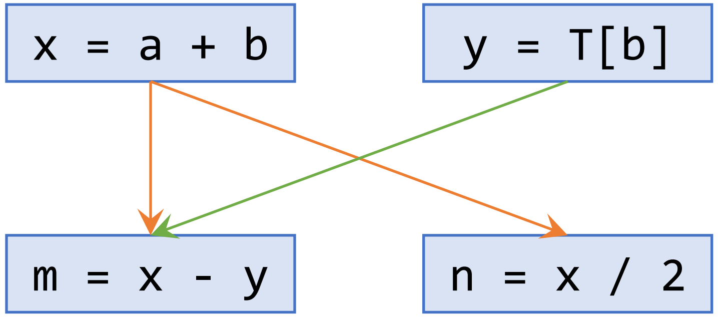
Instruction-Level Parallelism

- From mid 1980s to 2000, increase instruction throughput
  - Pipelining: partially overlap execution of instructions
  - Superscalar & out-of-order: executing multiple instructions together
- Limitations
  - Dependencies (data or control) between instructions
  - Hardware complexity and cost



```
add r3, r1, r2  F D E M W
mul r4, r3, 2     F D E M W
ld  r5, r4(0)       F D E M W
```

Pipelining



```
x = a + b      y = T[b]
m = x - y      n = x / 2
```

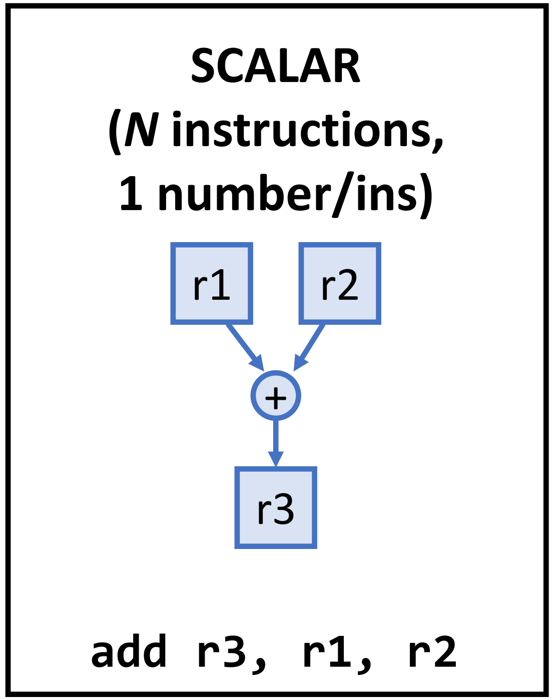
Superscalar & OoO

Data-Level Parallelism

- Scalar processors execute each instruction on single numbers (scalars)
- Vector processors may execute one instruction on vectors of numbers
- SIMD (single-instruction, multiple-data)
  - Increase processing throughput
  - Amortize instruction cost
  - E.g., Intel SSE, AVX, …

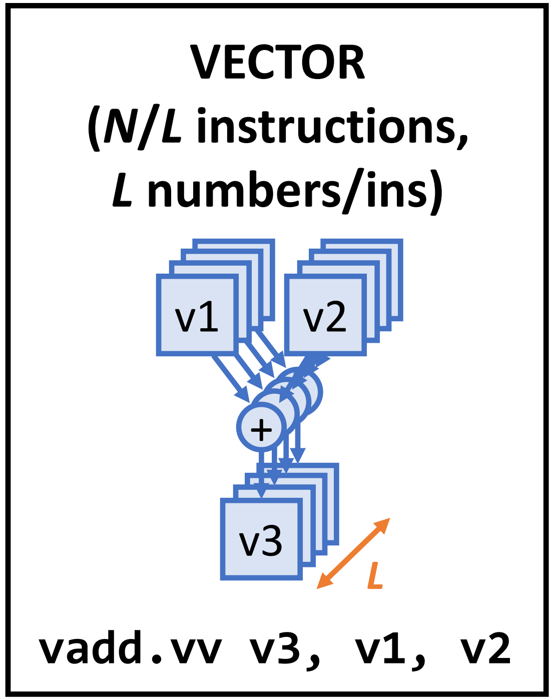
```
for i: 0 to N
  c[i] = a[i] + b[i]
```

Q: what domains of algorithms would SIMD be most beneficial?



SCALAR
**(*N* instructions,
1 number/ins)**

`add r3, r1, r2`

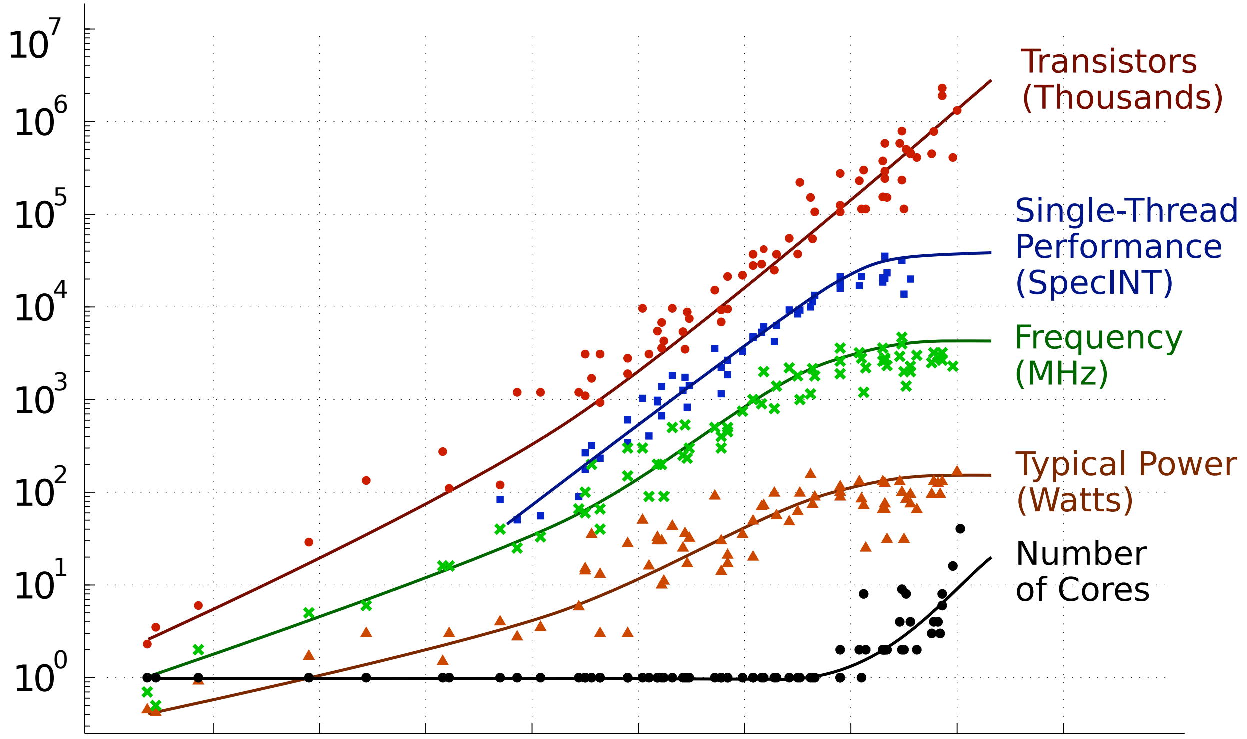


VECTOR
**(*N*/*L* instructions,
L numbers/ins)**

`vadd.vv v3, v1, v2`

Thread-Level Parallelism

- After 2000, chip multiprocessors (CMPs), i.e., multi-core
- Single-thread performance is more and more difficult to optimize
- Explicitly extract task parallelism and specify as multi-thread programs



Data collected by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, C. Batten

THE WALLS

... which block us from further improving performance *now*

Post-Dennard Scaling Era

- Moore's Law *without* Dennard Scaling
  - No more voltage and current scaling/reduction!! Why?
- When transistor size is so small (a few nanometers) …
  - Transistor threshold voltage and gate oxide thickness are set by leakage
  - Further reducing voltage cannot effectively turn off
  - Leakage power may exceed switch power
- Implication: 1.4x chip capability per generation if with constant power

Feature Size	Area A	Capacitance C	Voltage V	Current I	Freq $f = I/CV$	Energy $E = CV^2$	Power $P = CV^2f$	Pwr Density P/A
$1/S$	$1/S^2$	$1/S$	~ 1	~ 1	$< S$	$1/S$	< 1	$< S^2$

The Power Wall

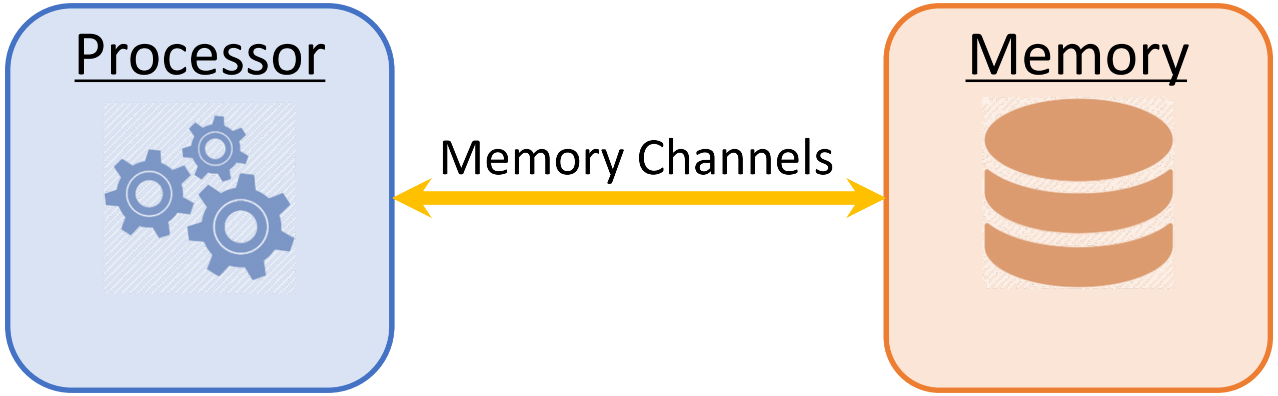
- Power Wall: all computers are power limited
  - From cellphones to datacenters
  - Power density would increase at S^2, need to cap it at 1

- Slow down the frequency scaling
  - Basically frequency has almost stopped scaling (see previous slide): $S \rightarrow 1$
- Reduce chip utilization (*Dark Silicon*: dark area of chip not turned on)
  - S^2 transistors available, but can only utilize S transistors simultaneously
  - Save another S factor

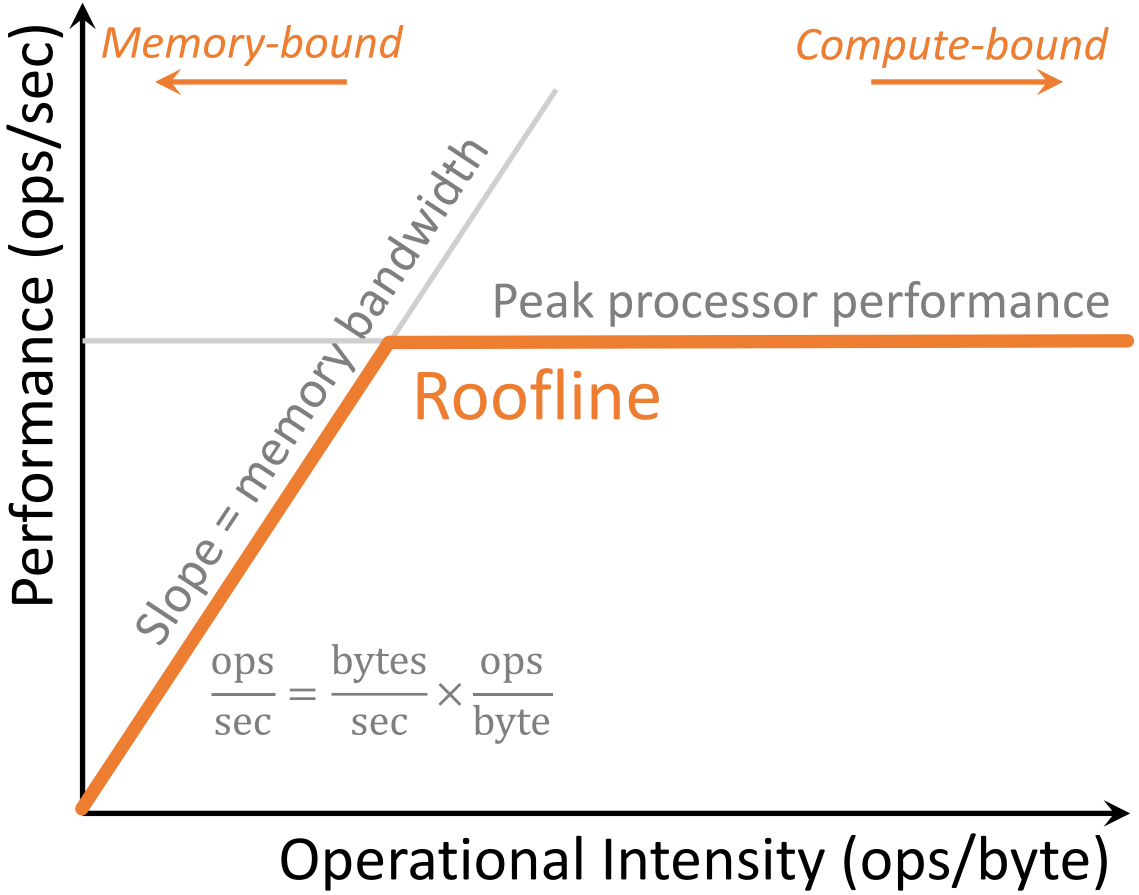
H. Esmaeilzadeh, et al. *Dark Silicon and the End of Multicore Scaling*. ISCA, 2011.

Recall: Von Neumann Architecture

- Separated processor (compute chip) and memory (data chip)
  - Fetch data from memory into processor
  - Compute in processor (processor has limited local data store)
  - Write back data from processor to memory
  - Continue with next data …
- Overall performance is determined by both processor and memory
  - How fast processor computes
  - How fast memory delivers data



The Roofline Model

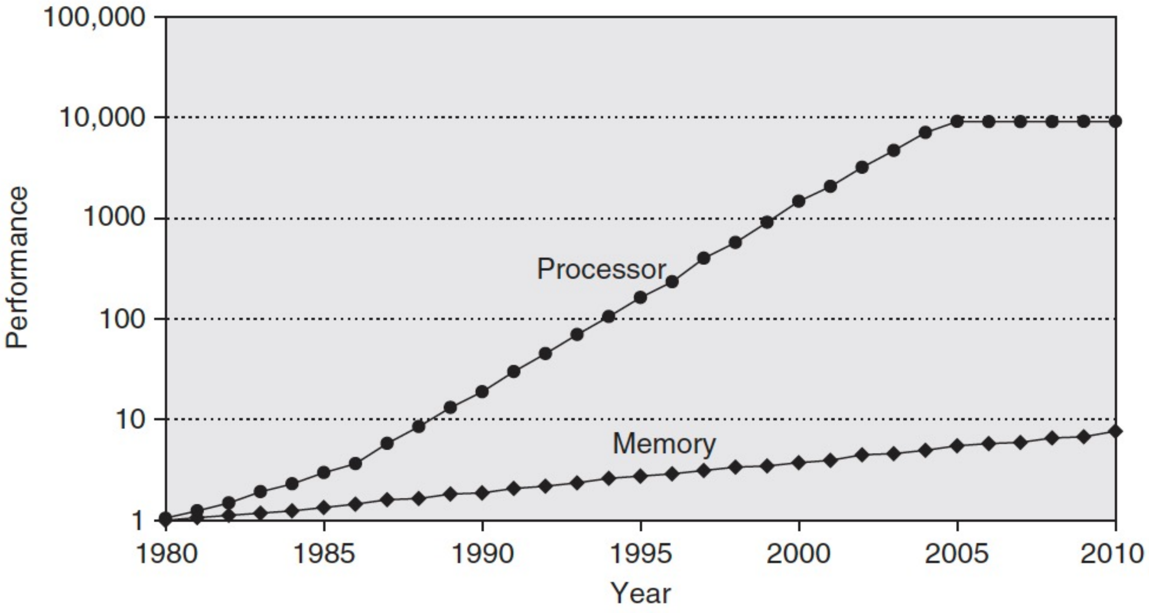


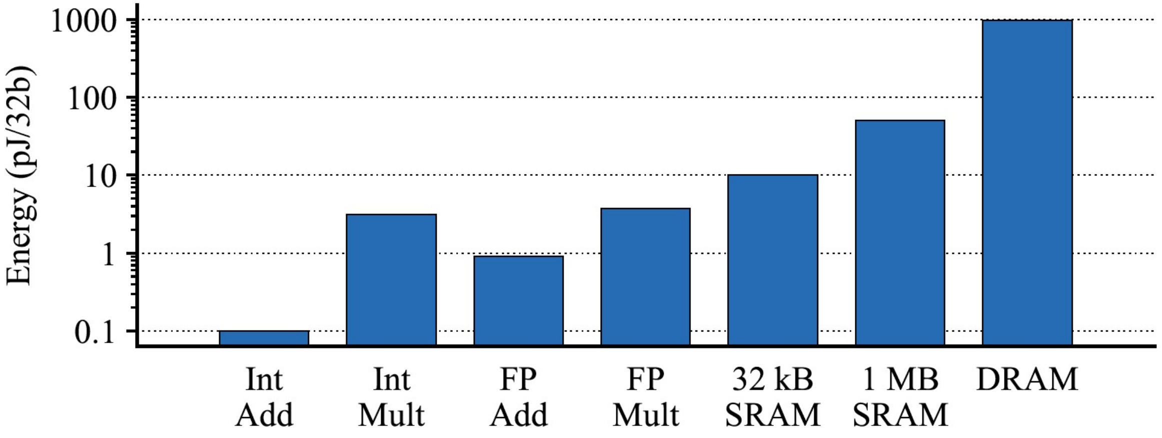
- Performance: operations per second
- Operational Intensity
  - How many ops to perform for each byte
  - A characteristic of program
  - Denote the ratio of compute to data
- Overall performance = min(
  processor performance,
  memory performance)
  - High OI: compute-bound
  - Low OI: memory-bound

S. Williams, et al. *Roofline: An Insightful Visual Performance Model for Multicore Architectures*. CACM, April 2008.

The Memory Wall

- Memory Wall: memory performance and energy dominate systems
  - Memory (DRAM) technology scales more slowly than processor (MOS)
  - Today, memory accesses are two to three orders of magnitude more expensive than processor operations





J. L. Hennessy and D. A. Patterson. *Computer Architecture: A Quantitative Approach* (Fifth Edition).
M. Horowitz. *Computing's energy problem (and what we can do about it)*. ISSCC, 2014.

Make Things Great Again

Let’s break the walls!

Improving Power: Energy Efficiency

$$\text{Power} = \frac{\text{Energy}}{\text{Time}} = \frac{\text{Energy}}{\text{Op}} \times \frac{\text{Op}}{\text{Time}}$$

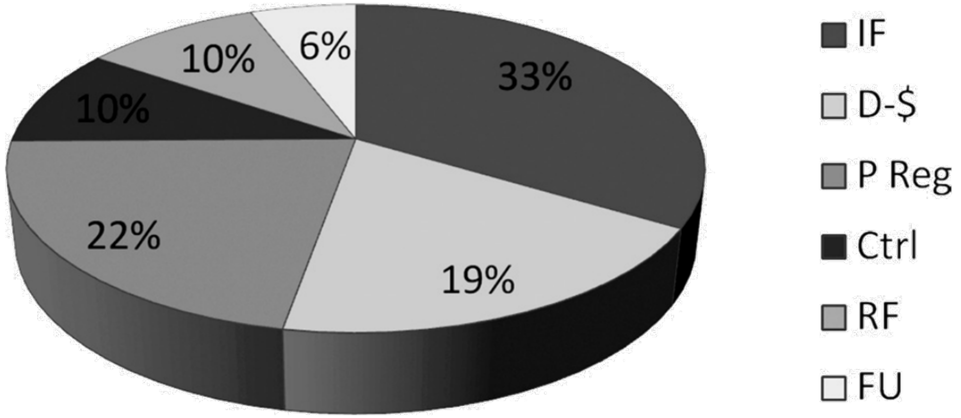
Energy Efficiency

Performance

- To improve performance we must improve energy efficiency
  - Otherwise, we cannot use the additional transistors

- Where does the energy go?

Figure 4. Processor energy breakdown for base implementation. IF is instruction fetch/decode. D-$ is data cache. P Reg includes the pipeline registers, buses, and clocking. Ctrl is miscellaneous control. RF is register file. FU is the functional units.



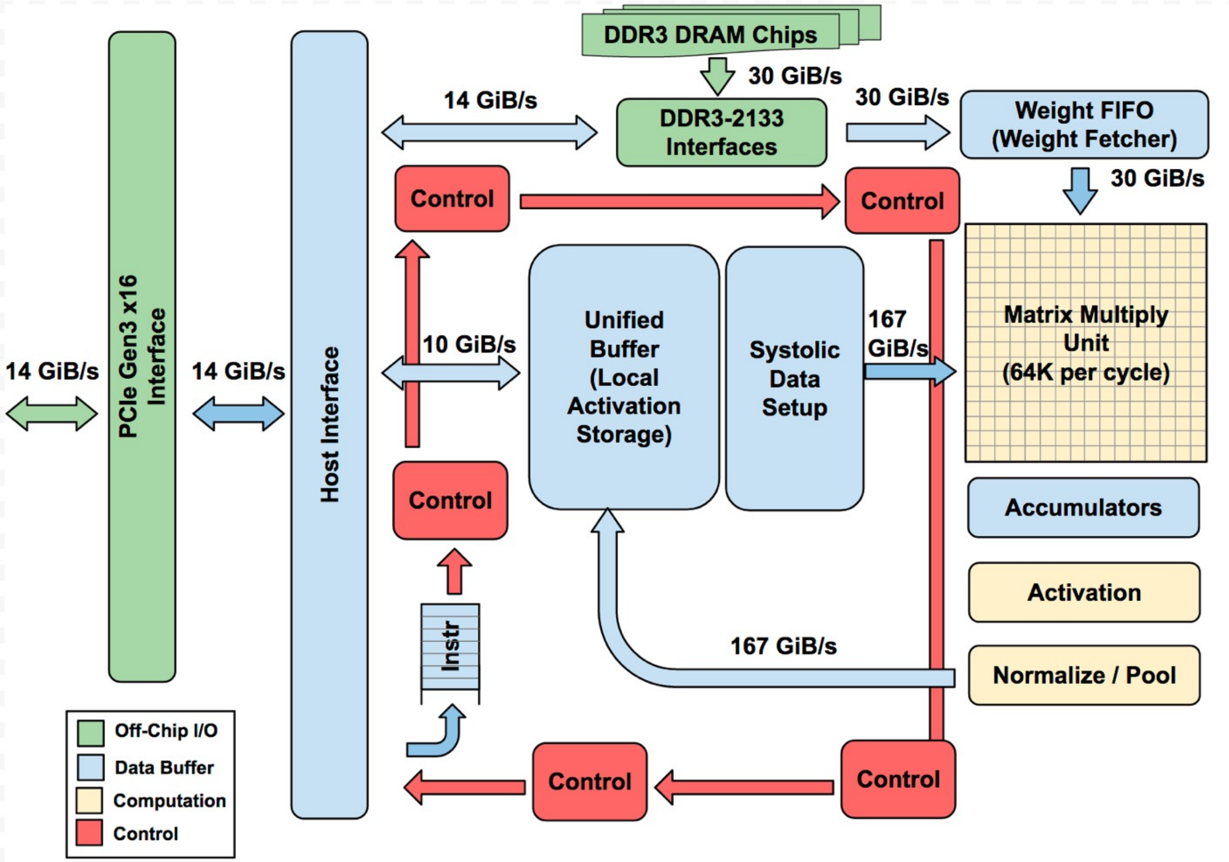
R. Hameed, et al. *Understanding Sources of Inefficiency in General-Purpose Chips*. CACM, October 2011.

Domain-Specific Architectures (DSAs)

- Achieve better energy efficiency by tailoring the architecture to characteristics of the domain
  - Not one application, but a domain of applications – different from strict ASIC
  - Requires more domain-specific knowledge then general-purpose processors
- DSA can …
  - Reduce overheads of general-purpose
    - Instruction fetch, instruction scheduling (pipelining/OoO), …
  - Use custom optimizations
    - Lower precision in certain domains, e.g., neural nets
  - Have more effective parallelism
    - SIMD, spatial arrays of simple processing elements, …
  - Access memory more efficiently
    - User-controlled on-chip buffer management

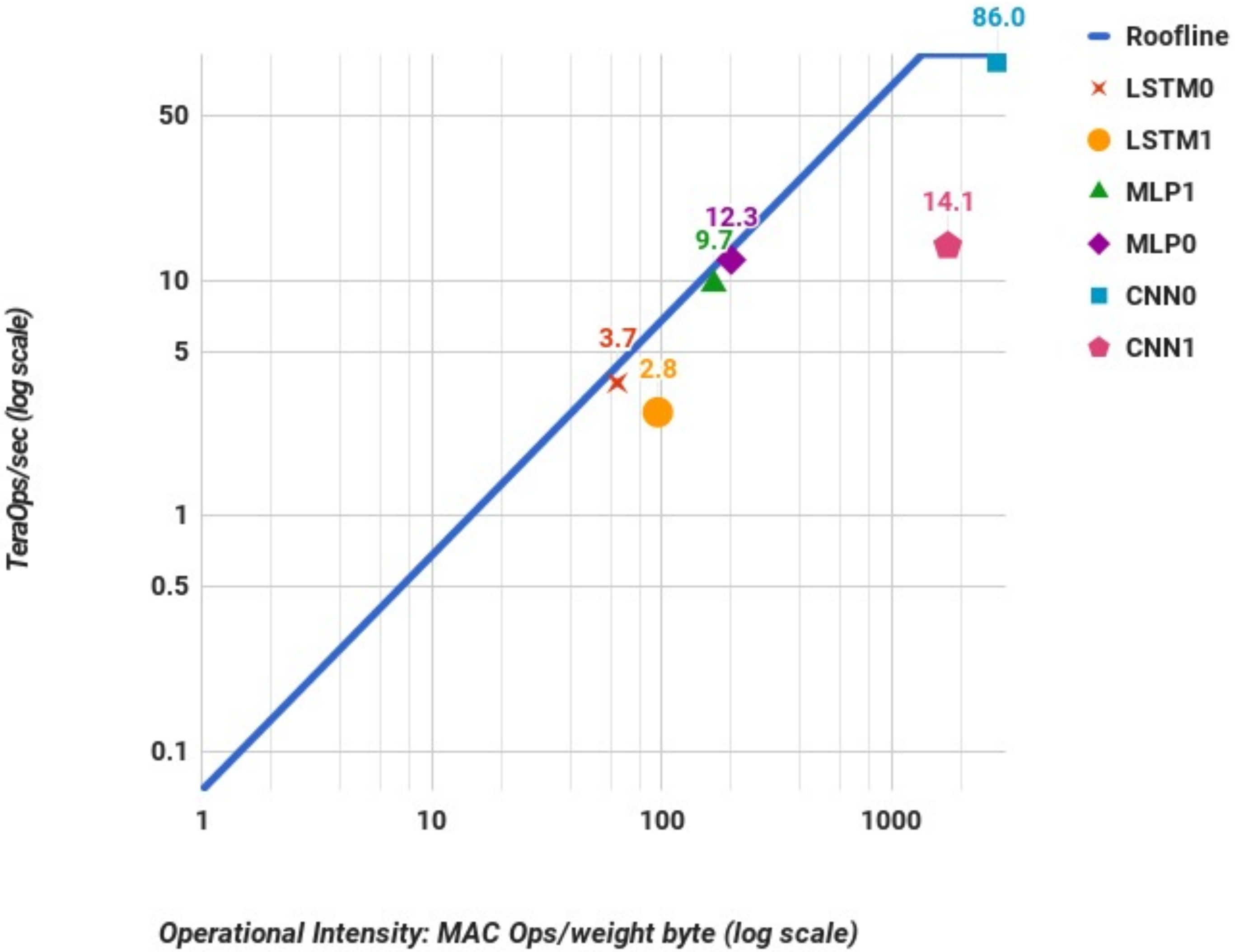
Example: Google TPU v1 for Neural Nets

- Matrix Multiply Unit
  - 256 x 256 8-bit multiply-accumulate
  - Peak: 92 Tops/sec (@ 700 MHz)
- On-chip memories
  - 4 MB accumulator memory
  - 24 MB unified buffer
- 25x MACs vs. GPU
- 3.5x on-chip memory vs. GPU
- 29x perf/Watt vs. GPU



N. Jouppi, et al. *In-Datacenter Performance Analysis of a Tensor Processing Unit*. ISCA, 2017.

TPU v1 Roofline



- What do you see from this plot?

N. Jouppi, et al. *In-Datacenter Performance Analysis of a Tensor Processing Unit*. ISCA, 2017.

Open Research Questions for DSA

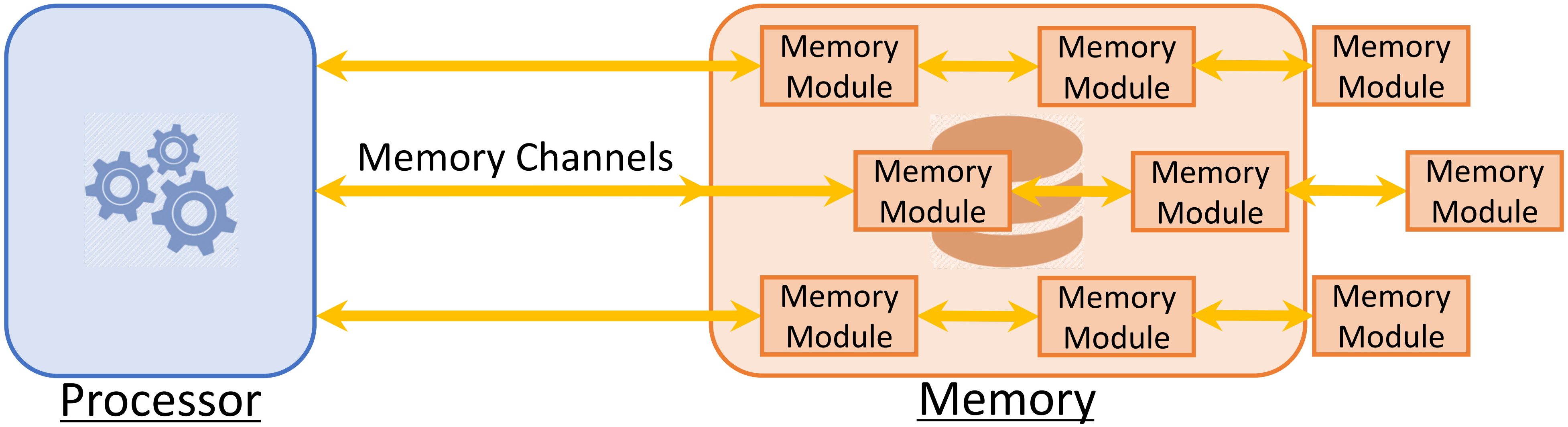
- Tradeoff between general-purpose and domain-specific
  - Generality, cost vs. performance, efficiency
  - Smart selection of domains

- Hardware development cost → agile hardware development
  - Debugging, testing, verification, …

- Programming on DSAs
  - Domain-specific languages: TensorFlow, OpenGL, P4, Halide, …
  - Compiler challenges: from DSL code to DSA binary
  - Co-design of new DSLs and DSAs

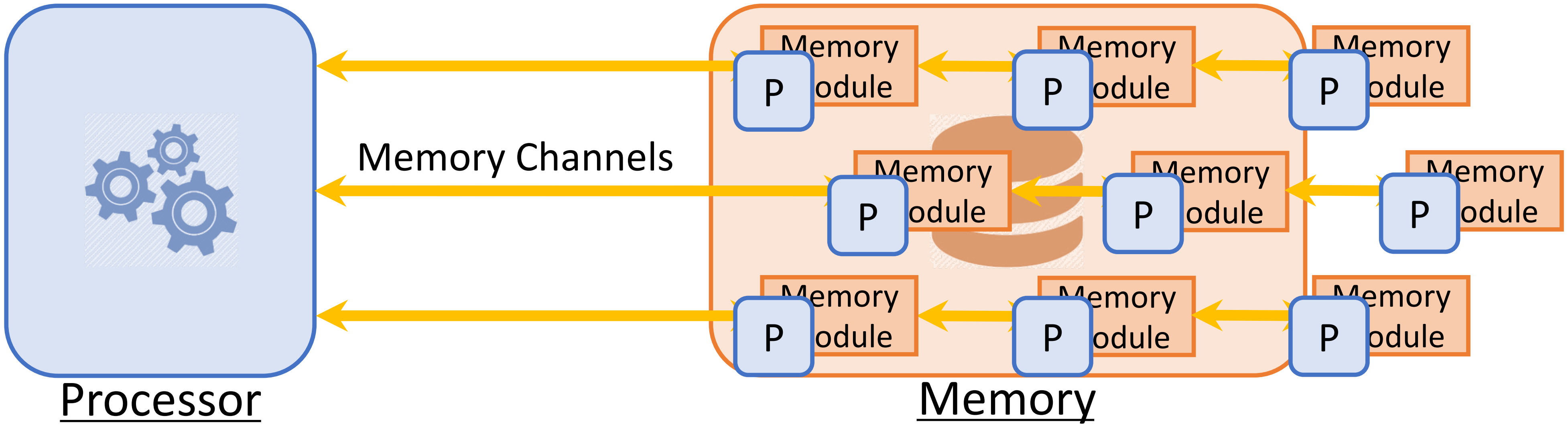
Improving Memory

- Memory capacity is not too hard to scale up
- Data bandwidth to processor is limited, restricting performance
  - Bandwidth (bytes/sec) = datawidth (bytes) x frequency (Hz)
  - Channel datawidth is limited by available chip pins, which is limited by chip area
  - Channel data frequency is limited by signal integrity
- Inter-chip data transfers (memory to processor) have high energy cost

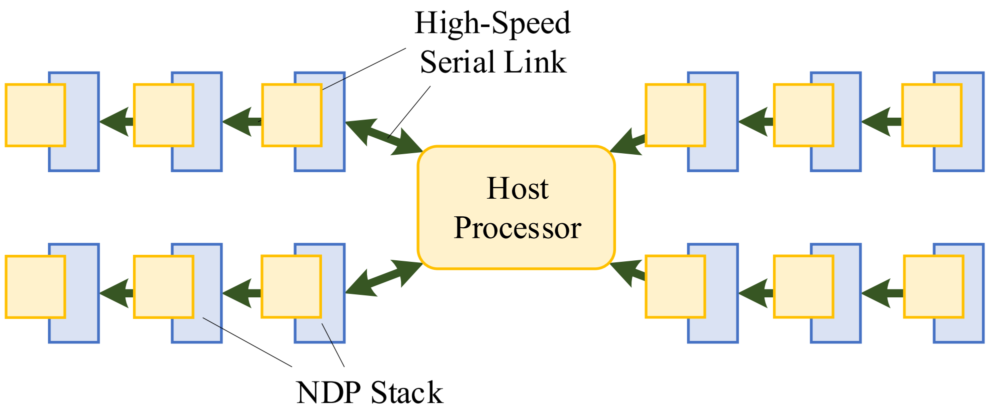


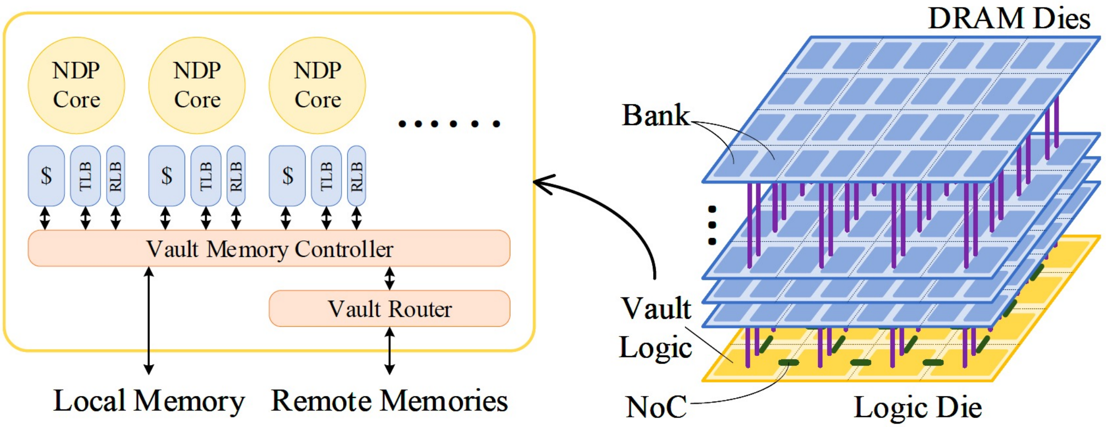
Near-Data Processing/Processing-In-Memory

- Key idea of NDP/PIM: execute computation closer to data
- Attach small processors inside or near memory modules
  - High local data bandwidth, low energy cost, low access latency, high parallelism
- Near-data compute capability is constrained by area & power budget
  - Compute-bound tasks: execute on host processor
  - Memory-bound tasks: execute on near-data processors



Example: Near-Data Processing Architecture

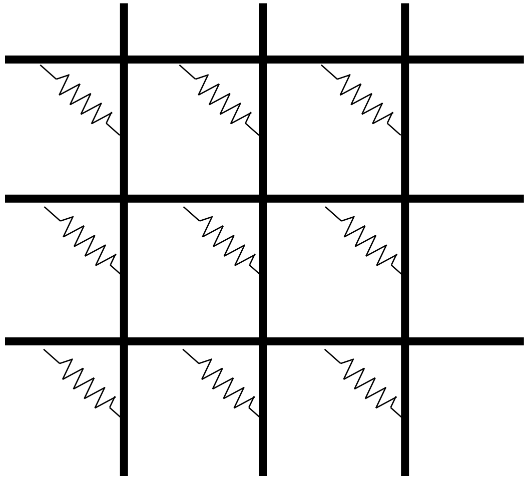


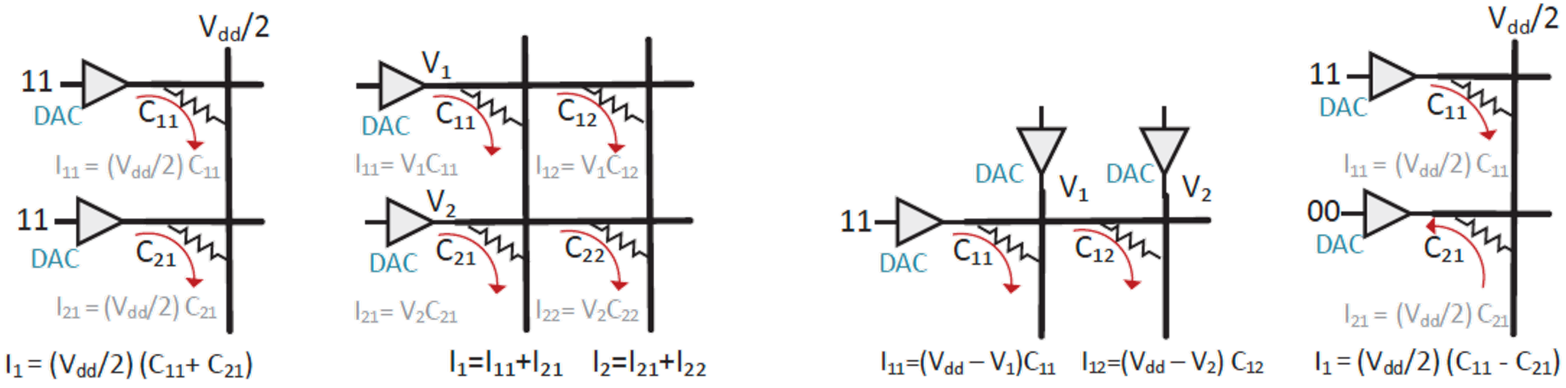


- 3D stacking integration for NDP: a logic die below multiple memory dies
  - Much more local vertical connections within a chip module
  - Memory bandwidth scale proportionally to memory capacity
  - Much more small cores utilize abundant memory bandwidth
- Implication: centralized → distributed
  - Making software programming more difficult

M. Gao, et al. *Practical Near-Data Processing for In-memory Analytics Frameworks*. PACT, 2015.

Example: In-Situ Analog Processing-in-Memory





(a) Addition **(b) Dot-product** **(c) Element-wise multiplication** **(d) Subtraction**

- Resistive RAM (memristor): cell resistance/conductance represent value
- Enable analog dot-product ($I = \sum V \cdot G$) and more
  - Computation within memory: one of the two operands stays inside memory
  - Require digital-to-analog and analog-to-digital conversions (DAC, ADC)
  - Well-suited for vector/matrix computation
- Challenge: how to enable more general, fine-grained operations?

D. Fujiki, et al. *In-Memory Data Parallel Processor*. ASPLOS, 2018.

Open Research Questions for NDP/PIM

- Impact on programming models
  - Heterogenous between host and NDP processors
  - Distributed among NDP processors
  - Work partition, data layout, coherence, consistency, synchronization, …

- Issues with analog computation
  - Overheads of digital-analog conversion
  - Physical device reliability, noise, interference, data precision, …

- More efficient computation
  - Fully utilize abundant memory bandwidth
  - Meet the tight constraints of area and power budget in memory modules

Summary

- Computer systems used to be great! – exponential perf increasing
  - Moore’s Law and Dennard Scaling
  - Parallelism in processor architectures

- They are not so “great” now ☹ – the challenges
  - *The Power Wall*
  - *The Memory Wall*

- How to make computer systems great again? – the research directions
  - Domain-specific architectures
  - Near-data processing/processing-in-memory
  - And many more!

The Scope of My Research Lab

Innovative Data-centric Efficient Architecture Lab (IDEAL)

- Data storage
  - Near-Data Processing/Processing-in-Memory architectures
  - Hybrid memory systems: DRAM + NVM
- Data processing
  - AI accelerators: hardware architectures + software scheduling
  - Reconfigurable architectures
- Data security
  - Isolated execution on accelerator hardware
  - Hardware enclaves + cryptographic algorithms

Extended Reading/Watching

- Turing Lecture by 2017 ACM A. M. Turing Award recipients – John L. Hennessy and David A. Patterson

A New Golden Age for Computer Architecture:
Domain-Specific Hardware/Software Co-Design,
Enhanced Security, Open Instruction Sets,
and Agile Chip Development

- https://dl.acm.org/citation.cfm?id=3282307
  - A video recording the Turing lecture (on YouTube)
  - An article on Communications of the ACM

THANKS!

Webpage: http://people.iiis.tsinghua.edu.cn/~gaomy/

Contact: gaomy@tsinghua.edu.cn