PARALLELISM & CONCURRENCY: INTRODUCTION

LECTURE 01-1

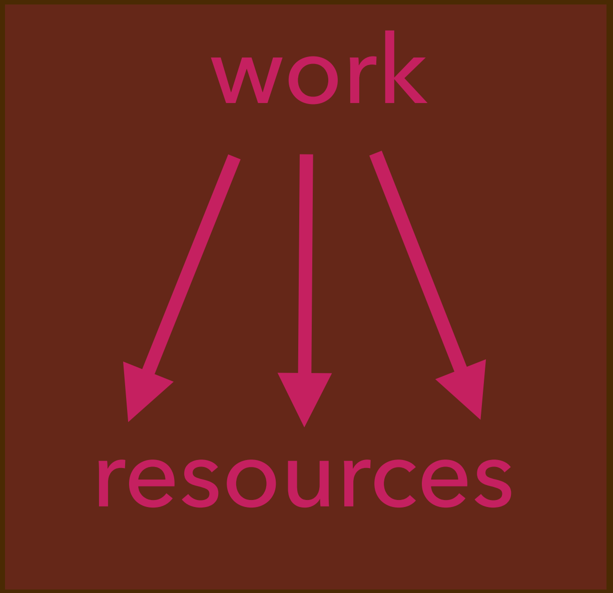
JIM FIX, REED COLLEGE CSCI 361

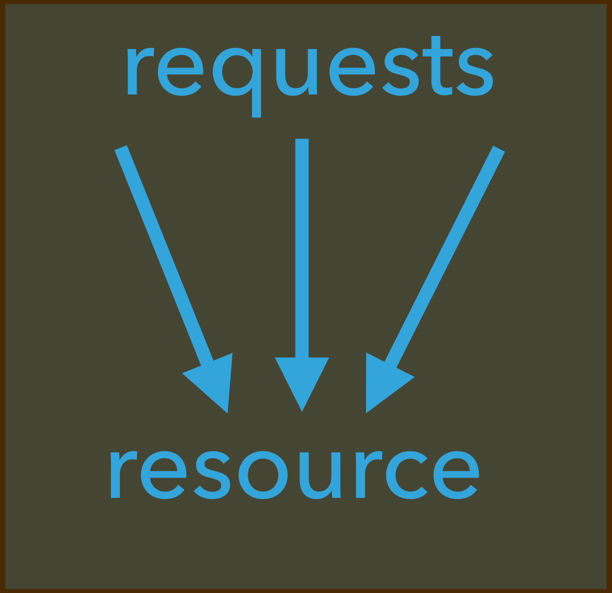
TODAY

- What is parallelism? What is concurrency?
- Why learn parallel programming and concurrency mechanisms?
  - Driven by trends in hardware and system design, deployment.
- Example parallel algorithm: merge sort
  - design and pseudocode
  - implementation in the Go language
- Brief overview of course and covered topics.
  - course web page: `https://jimfix.github.io/csci361/`

PARALLELISM VERSUS CONCURRENCY

- The two concepts are often confused; equated/conflated





parallelism - use several computational resources to solve a problem faster

concurrence - manage access to a shared resource (correctly and efficiently)

[From D. Grossman (UW)]

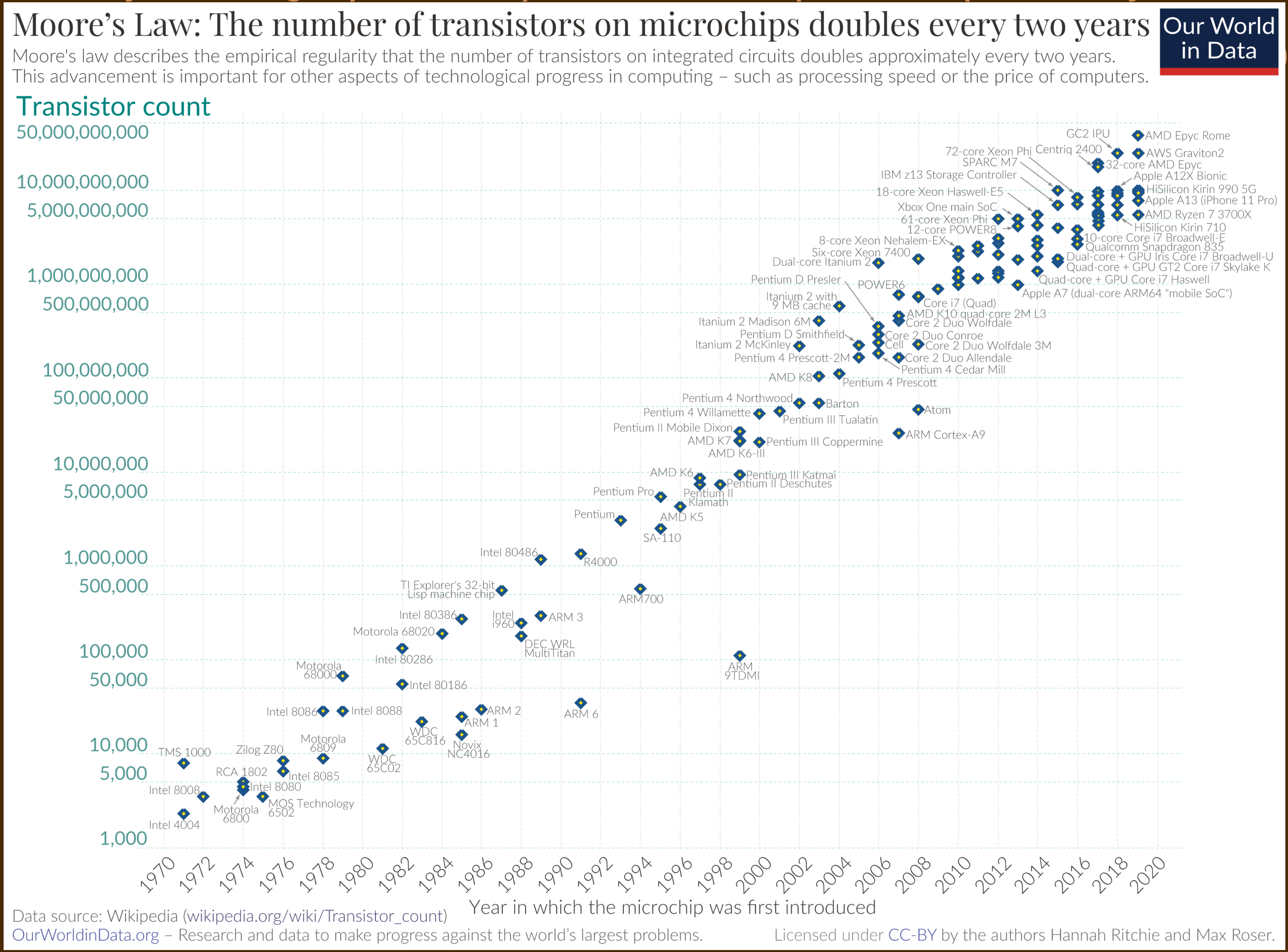
HISTORY: MOORE'S LAW AND SINGLE PROCESSOR PERFORMANCE

- For years, single processor performance improved exponentially.
    - ➡ Moore's Law: chip features (e.g. wires, transistors) can continually be made smaller
    - ➡ performance doubled (roughly) every 2.5 years.

HISTORY: MOORE'S LAW AND SINGLE PROCESSOR PERFORMANCE

- For years, single processor performance improved exponentially.

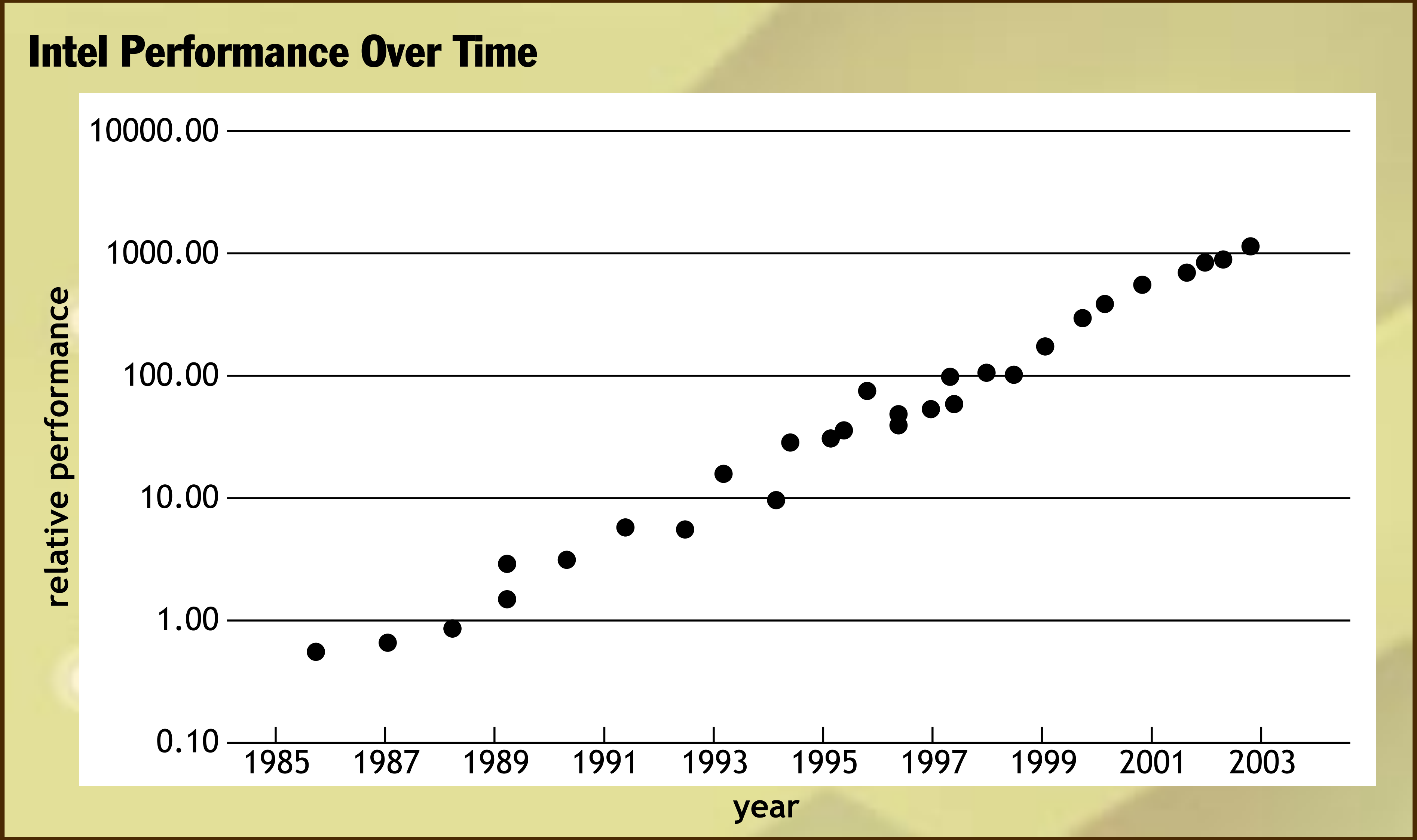
e made smaller



[Source: Wikimedia "Moore's Law Transistor Count 1979-2020"]

HISTORY: MOORE'S LAW AND SINGLE PROCESSOR PERFORMANCE

- For years, single processor performance improved exponentially.
  - ➡ Moore's Law: chip features (e.g. wires, transistors) can continually be made smaller
  - ➡ performance doubled (roughly) every 2.5 years.



[Source: Figure 1 of "The Future of Multiprocessors", K. Olukotun, L. Hammond *ACM Queue*, 2005]

HISTORY: MOORE'S LAW AND SINGLE PROCESSOR PERFORMANCE

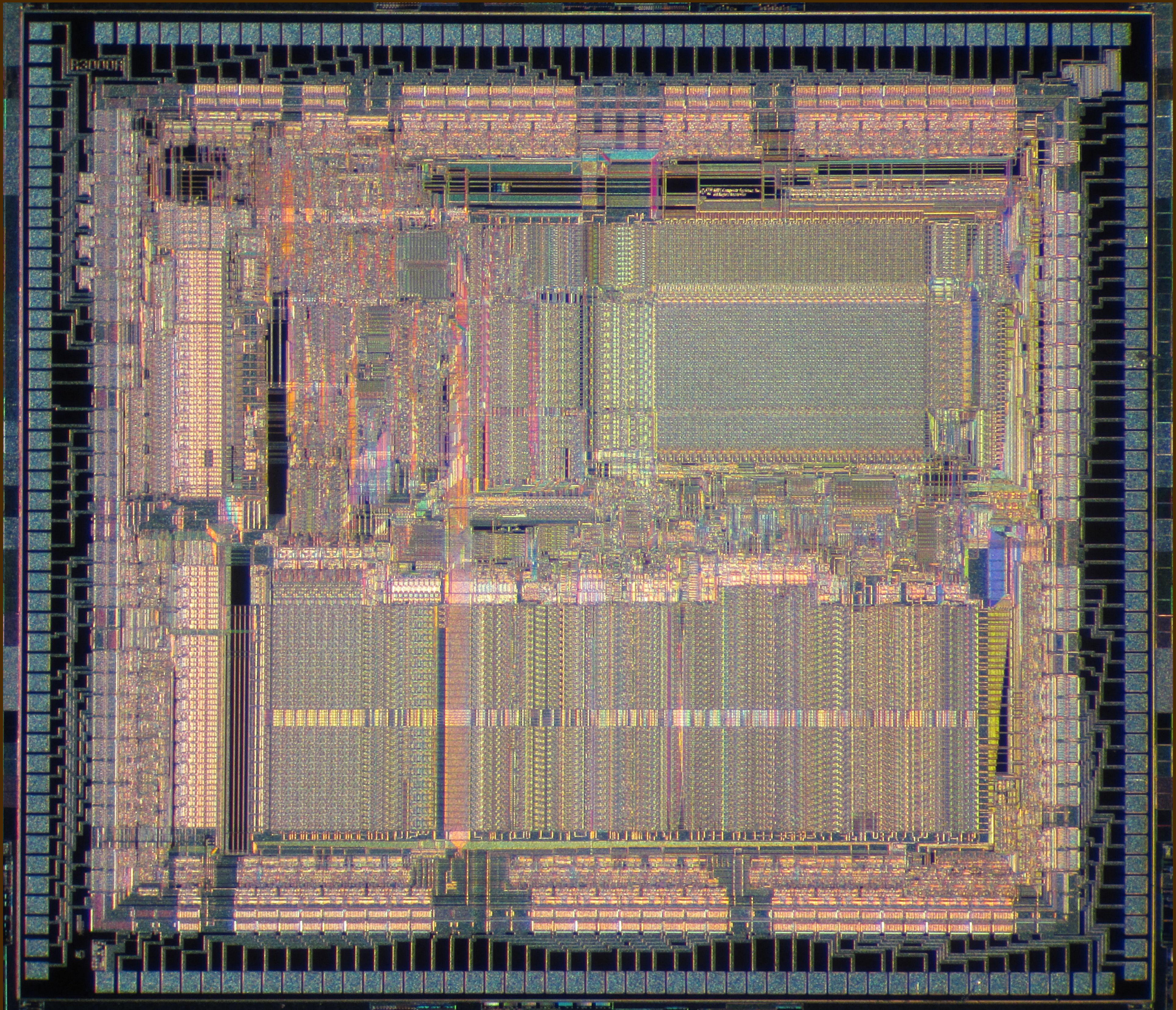
- Because of chip improvements, clock speed could be increased.
- And also processor could do more with all the extra transistors:
  - memory caches
  - pipelining
  - superscalar designs
  - out-of-order execution
  - speculative execution
  - vector, VLIW designs

HISTORY: MOORE'S LAW AND SINGLE PROCESSOR PERFORMANCE

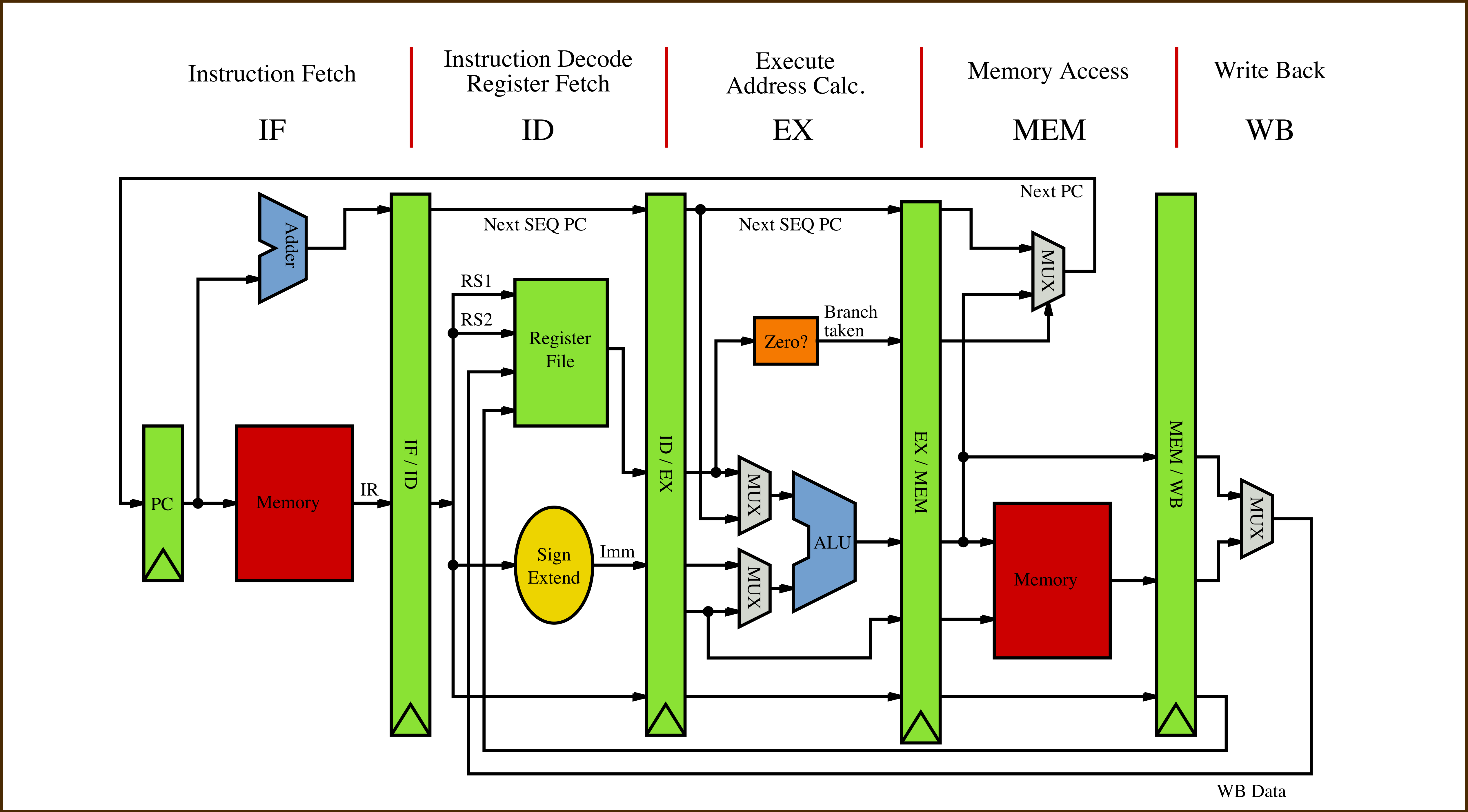
- Because of chip improvements, clock speed could be increased.
- And also processor could do more with all the extra transistors:
  - memory caches
  - **pipelining**
  - **superscalar designs**
  - **out-of-order execution**
  - **speculative execution**
  - **vector, VLIW designs**

⇐ these are all forms of parallelism

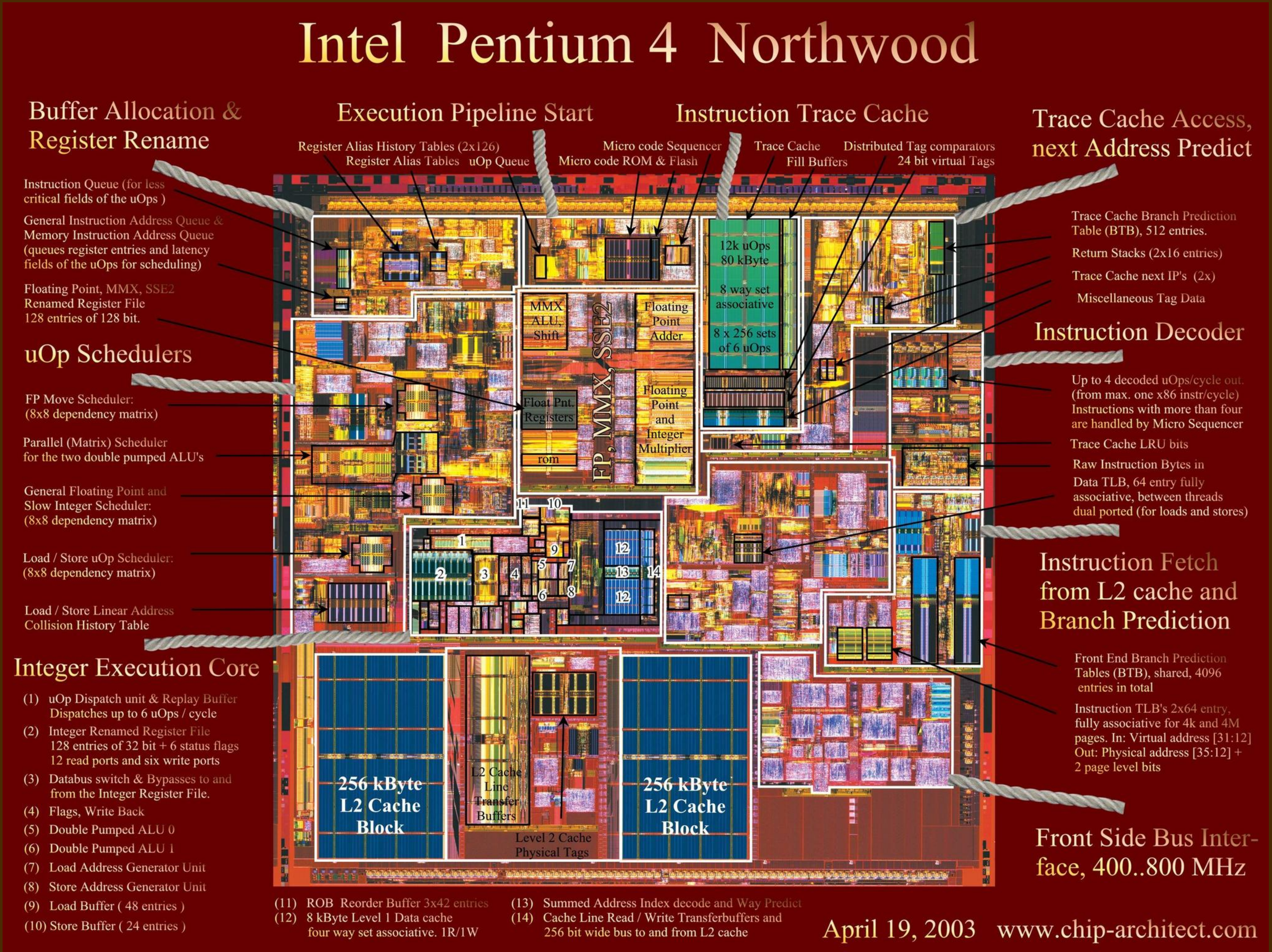
MIPS R3000A (1988)



PARALLELISM: PIPELINING



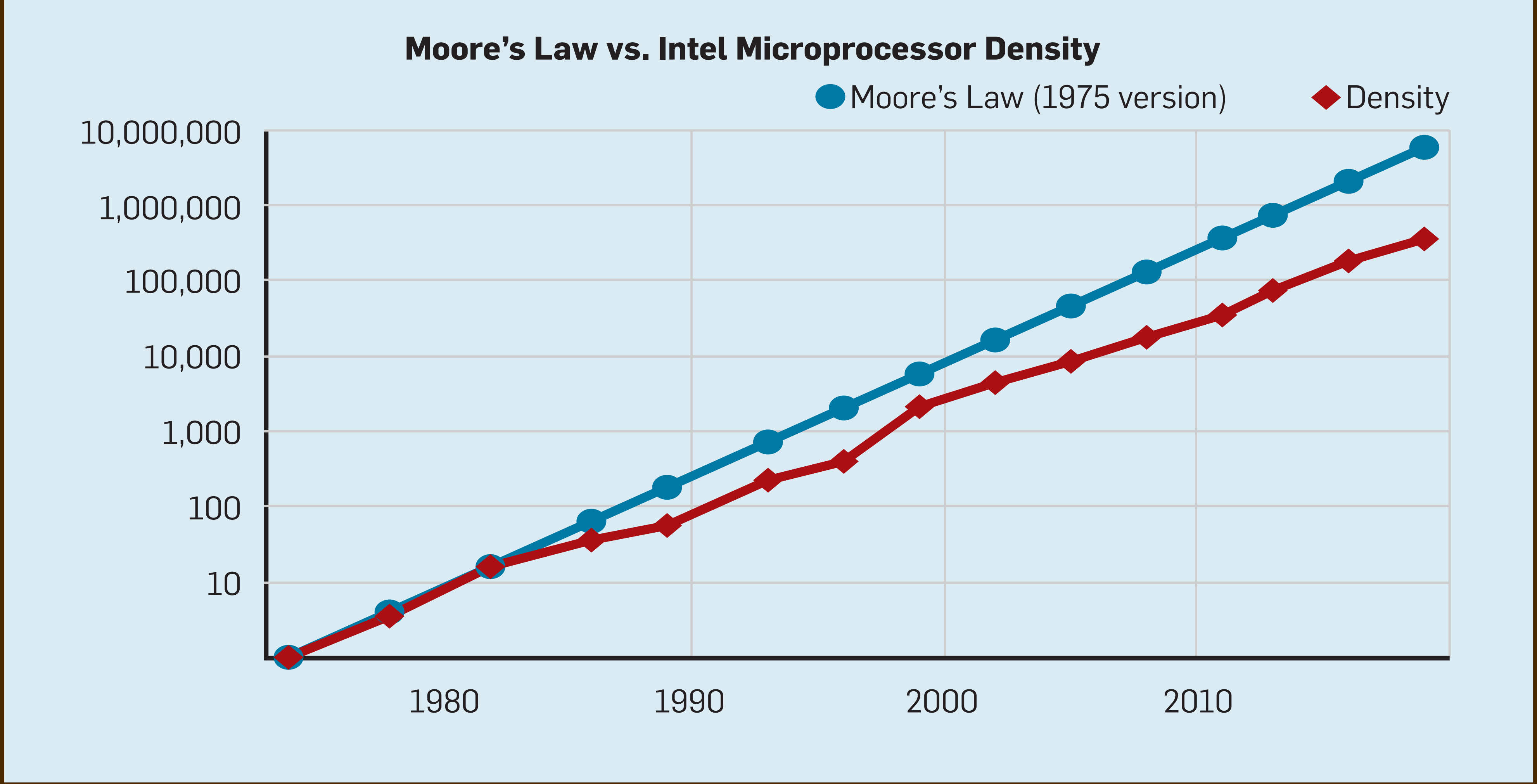
PENTIUM 4 (2003)


Intel Pentium 4 Northwood
Buffer Allocation & Register Rename
Execution Pipeline Start
Instruction Trace Cache
Trace Cache Access, next Address Predict
Register Alias History Tables (2x126)
Register Alias Tables
uOp Queue
Micro code Sequencer
Micro code ROM & Flash
Trace Cache Fill Buffers
Distributed Tag comparators 24 bit virtual Tags
Instruction Queue (for less critical fields of the uOps)
General Instruction Address Queue & Memory Instruction Address Queue (queues register entries and latency fields of the uOps for scheduling)
Floating Point, MMX, SSE2 Renamed Register File 128 entries of 128 bit.
uOp Schedulers
FP Move Scheduler: (8x8 dependency matrix)
Parallel (Matrix) Scheduler for the two double pumped ALU's
General Floating Point and Slow Integer Scheduler: (8x8 dependency matrix)
Load / Store uOp Scheduler: (8x8 dependency matrix)
Load / Store Linear Address Collision History Table
Integer Execution Core
(1) uOp Dispatch unit & Replay Buffer Dispatches up to 6 uOps / cycle
(2) Integer Renamed Register File 128 entries of 32 bit + 6 status flags 12 read ports and six write ports
(3) Databus switch & Bypasses to and from the Integer Register File.
(4) Flags, Write Back
(5) Double Pumped ALU 0
(6) Double Pumped ALU 1
(7) Load Address Generator Unit
(8) Store Address Generator Unit
(9) Load Buffer (48 entries)
(10) Store Buffer (24 entries)
(11) ROB Reorder Buffer 3x42 entries
(12) 8 kByte Level 1 Data cache four way set associative. 1R/1W
(13) Summed Address Index decode and Way Predict
(14) Cache Line Read / Write Transferbuffers and 256 bit wide bus to and from L2 cache
MMX ALU, Shift
FP, MMX, SSE2
Floating Point Adder
Float Pnt. Registers
rom
Floating Point and Integer Multiplier
12k uOps 80 kByte
8 way set associative
8 x 256 sets of 6 uOps
1 2 3 4 5 6 7 8 9 10 11 12 13 14
256 kByte L2 Cache Block
L2 Cache Line Transfer Buffers
Level 2 Cache Physical Tags
256 kByte L2 Cache Block
Trace Cache Branch Prediction Table (BTB), 512 entries.
Return Stacks (2x16 entries)
Trace Cache next IP's (2x)
Miscellaneous Tag Data
Instruction Decoder
Up to 4 decoded uOps/cycle out. (from max. one x86 instr/cycle) Instructions with more than four are handled by Micro Sequencer
Trace Cache LRU bits
Raw Instruction Bytes in
Data TLB, 64 entry fully associative, between threads dual ported (for loads and stores)
Instruction Fetch from L2 cache and Branch Prediction
Front End Branch Prediction Tables (BTB), shared, 4096 entries in total
Instruction TLB's 2x64 entry, fully associative for 4k and 4M pages. In: Virtual address [31:12] Out: Physical address [35:12] + 2 page level bits
Front Side Bus Interface, 400..800 MHz
April 19, 2003
www.chip-architect.com

HISTORY: MOORE'S LAW AND SINGLE PROCESSOR PERFORMANCE

- For years, single processor performance improved exponentially.
  - ➡ Moore's Law: chip features (e.g. wires, transistors) can continually be made smaller
  - ➡ performance doubled (roughly) every 2.5 years.

Figure 2. Transistors per chip of Intel microprocessors vs. Moore's Law.

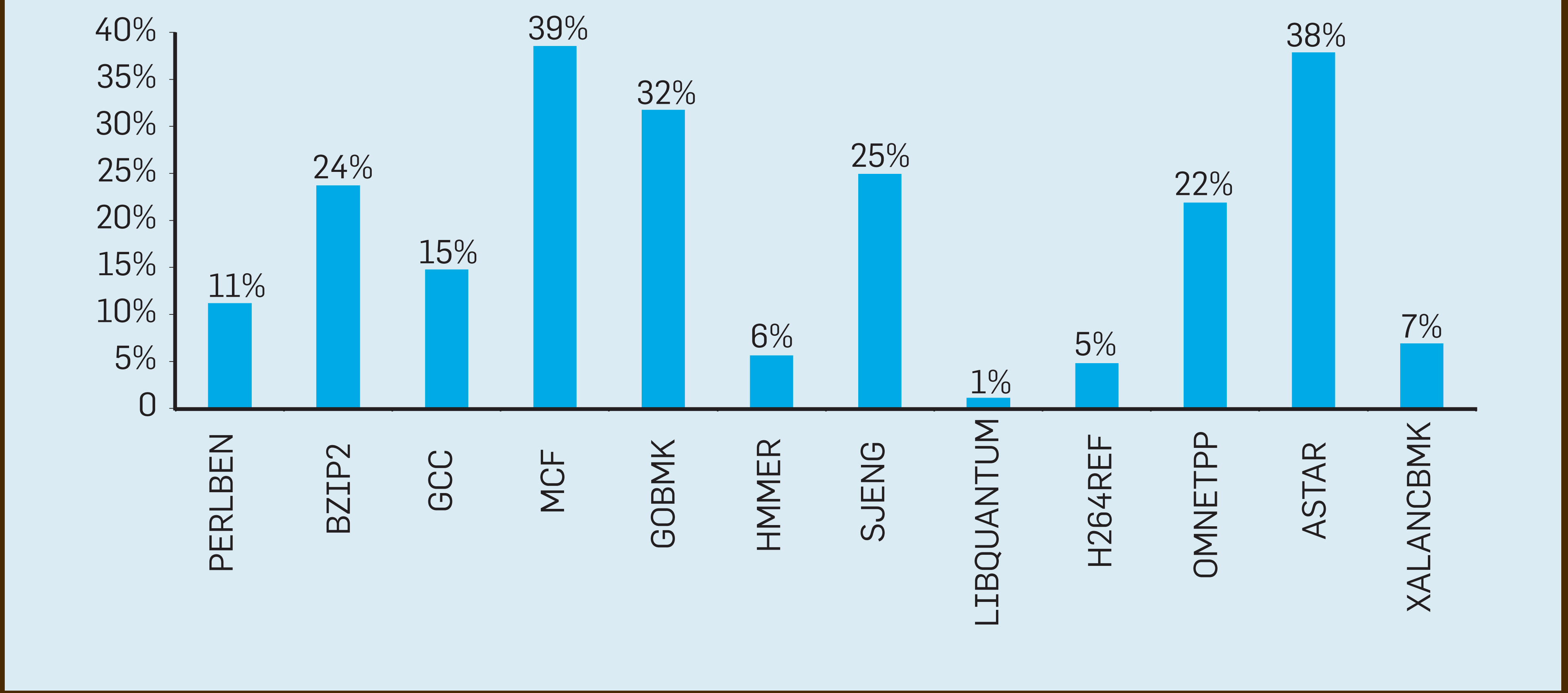


[Source: Figure 2 of "A New Golden Age for Computer Architecture", J. Hennesy, D. Patterson *Comm. of the ACM,* Feb 2019]

HISTORY: LIMITS TO SINGLE PROCESSOR PERFORMANCE

- In 2000s, computer architects hit real limits improving single-threaded performance .

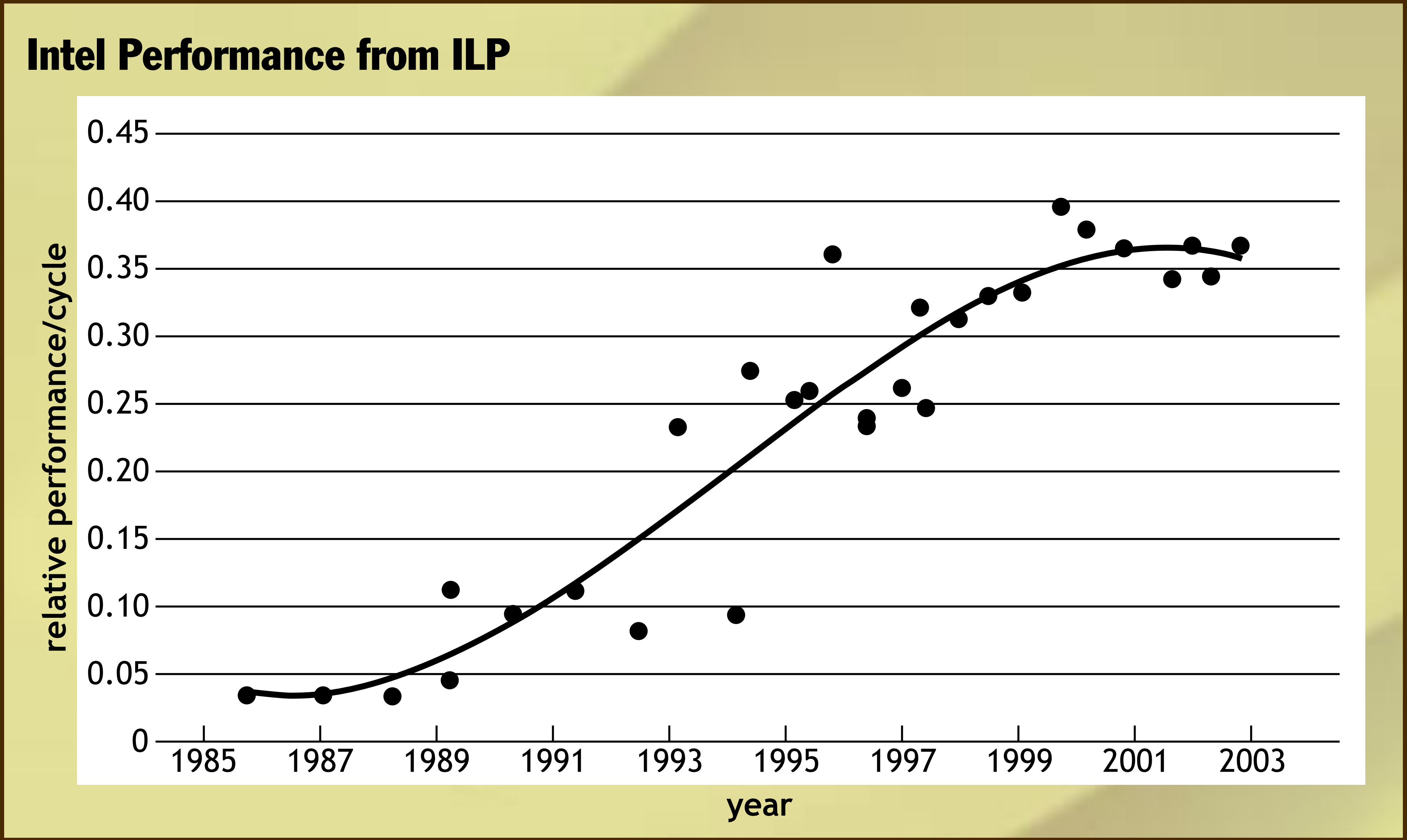
Figure 4. Wasted instructions as a percentage of all instructions completed on an Intel Core i7 for a variety of SPEC integer benchmarks.



[Source: Figure 4 of "A New Golden Age for Computer Architecture",
J. Hennesy, D. Patterson *Comm. of the ACM,* Feb 2019]

HISTORY: LIMITS TO SINGLE PROCESSOR PERFORMANCE

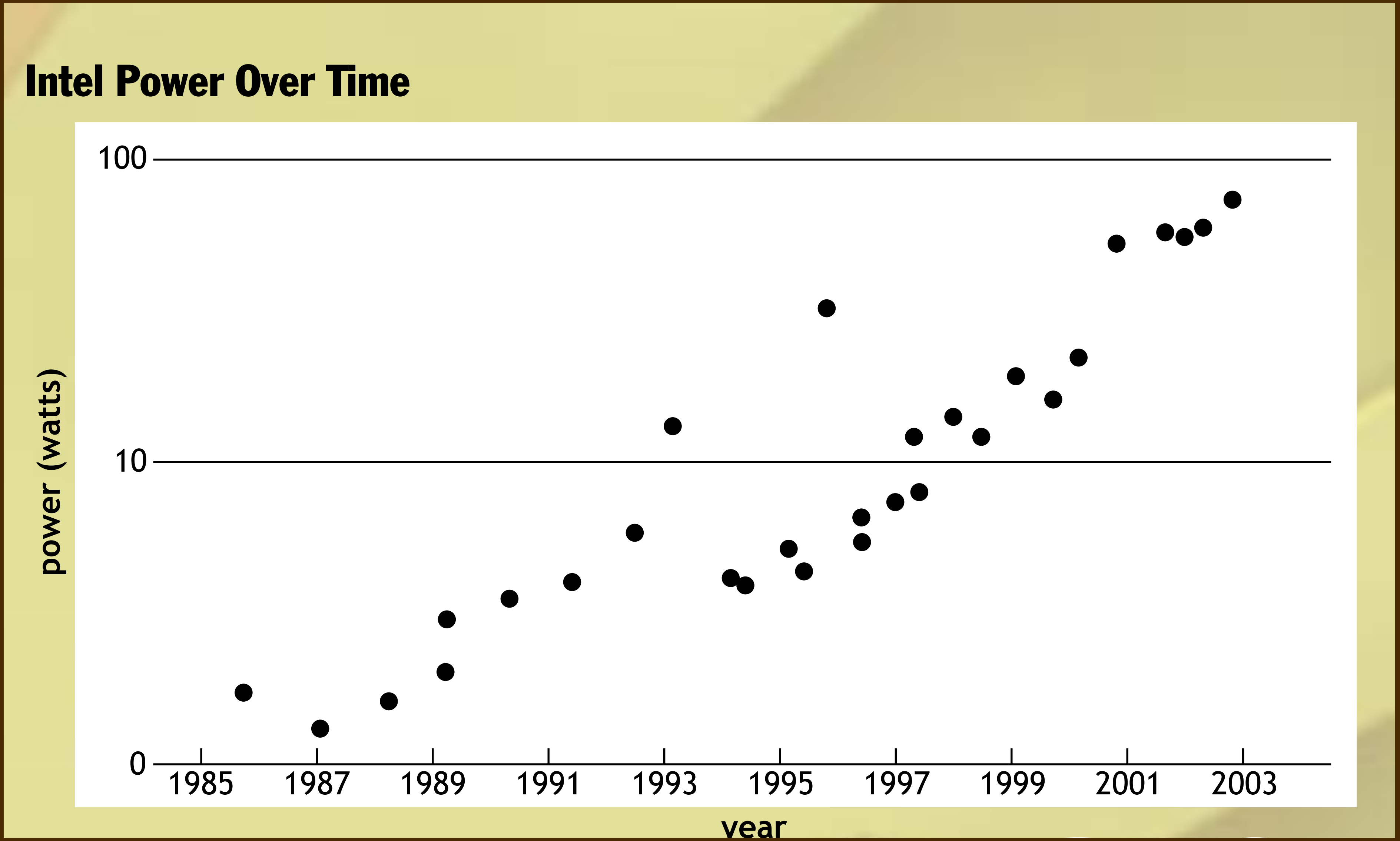
- In 2000s, computer architects hit real limits improving single-threaded performance .



[Source: Figure 2 of "The Future of Multiprocessors", K. Olukotun, L. Hammond *ACM Queue*, 2005]

HISTORY: LIMITS TO SINGLE PROCESSOR PERFORMANCE

- In 2000s, computer architects hit real limits improving single-threaded performance .

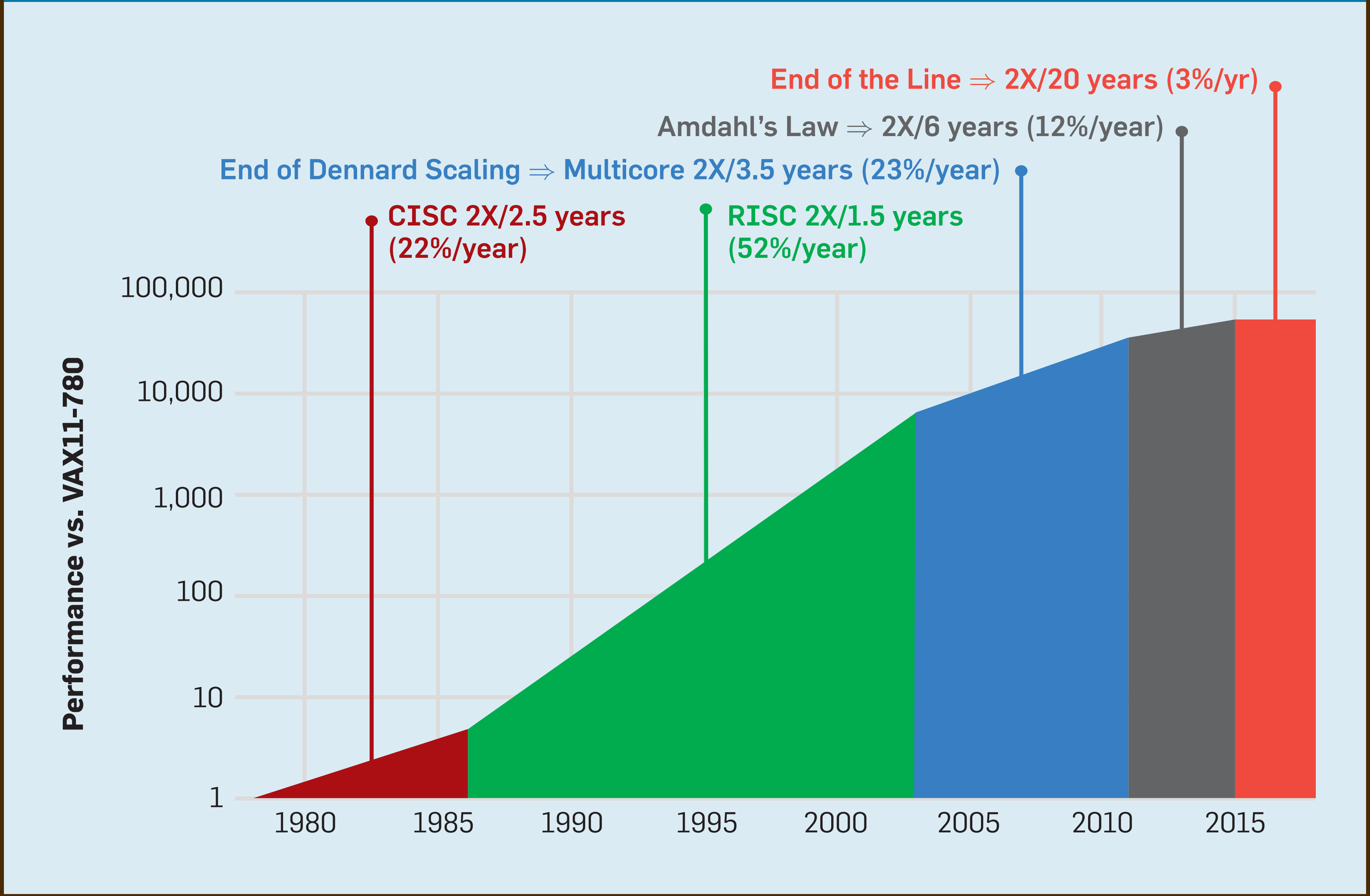


[Source: Figure 3 of "The Future of Multiprocessors", K. Olukotun, L. Hammond *ACM Queue*, 2005]

HISTORY: LIMITS TO SINGLE PROCESSOR PERFORMANCE

- In 2000s, computer architects hit real limits improving single-threaded performance .

Figure 6. Growth of computer performance using integer programs (SPECintCPU).



[Source: Figure 8 of "A New Golden Age for Computer Architecture", J. Hennesy, D. Patterson *Comm. of the ACM,* Feb 2019]

HISTORY: LIMITS TO SINGLE PROCESSOR PERFORMANCE

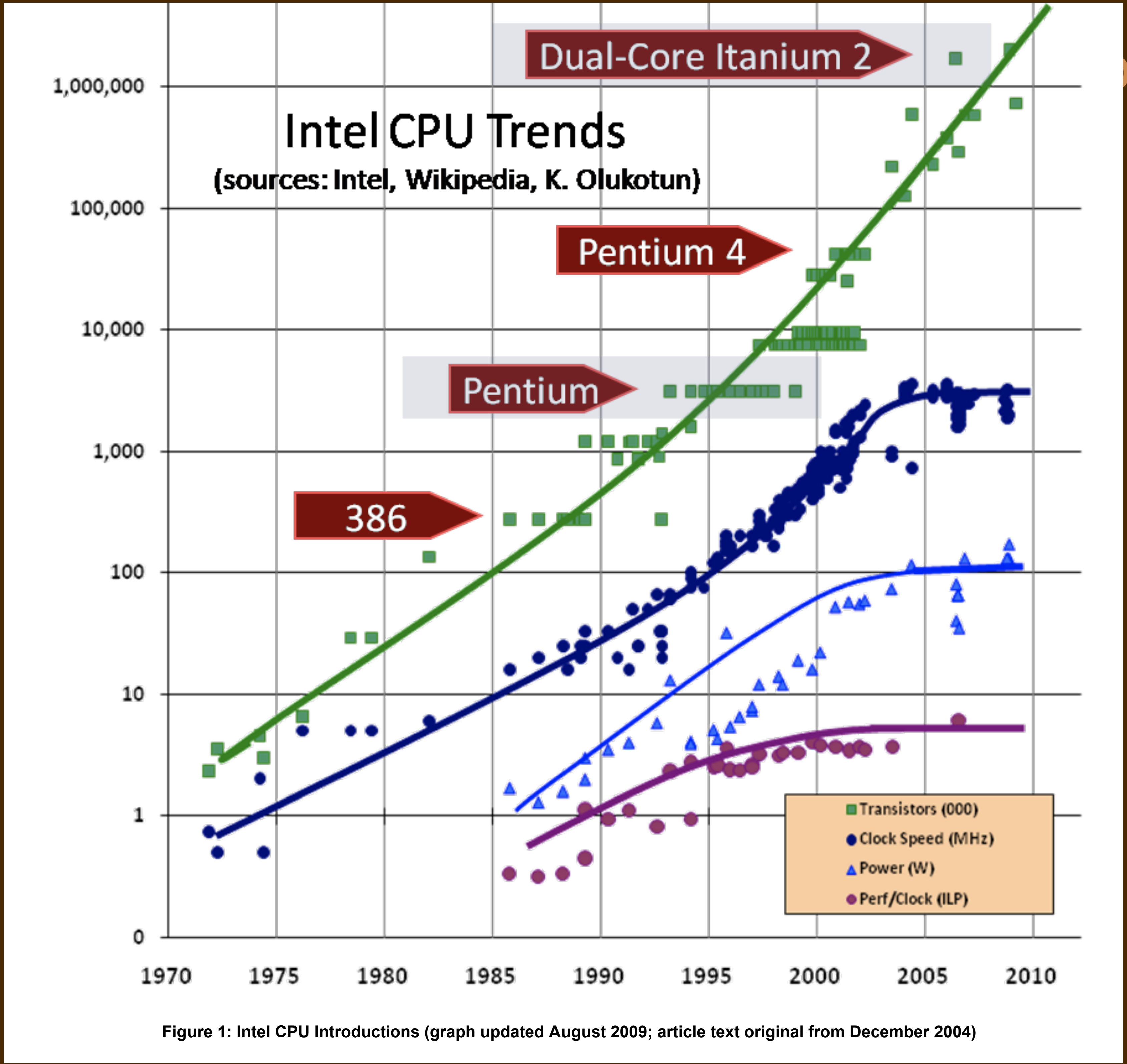


Figure 1: Intel CPU Introductions (graph updated August 2009; article text original from December 2004)

ving single-threaded

[Source:
"The Free Lunch Is Over"
H. Sutter, 2009]

HISTORY: MOORE'S LAW AND SINGLE PROCESSOR PERFORMANCE

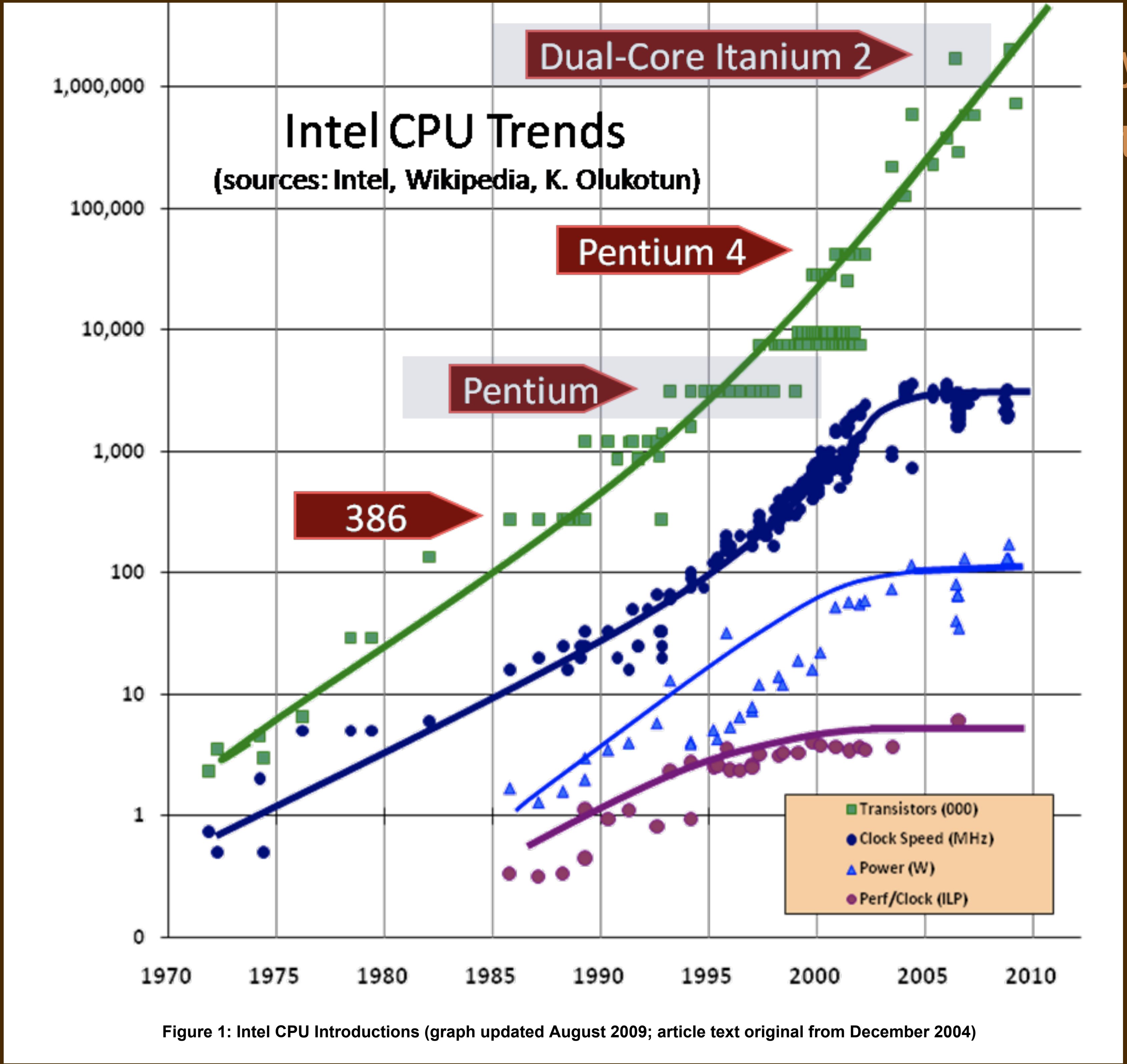


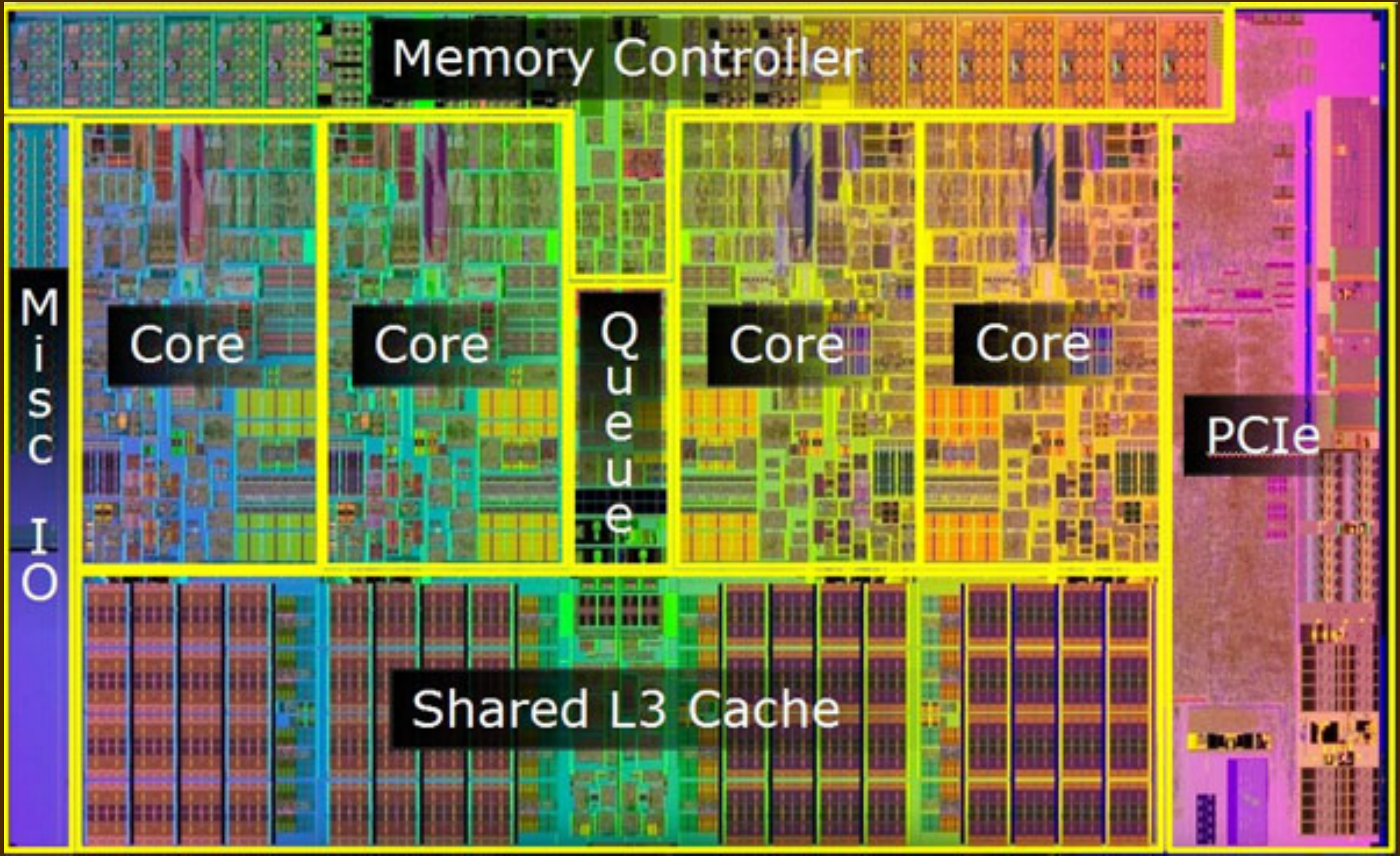
Figure 1: Intel CPU Introductions (graph updated August 2009; article text original from December 2004)

y.

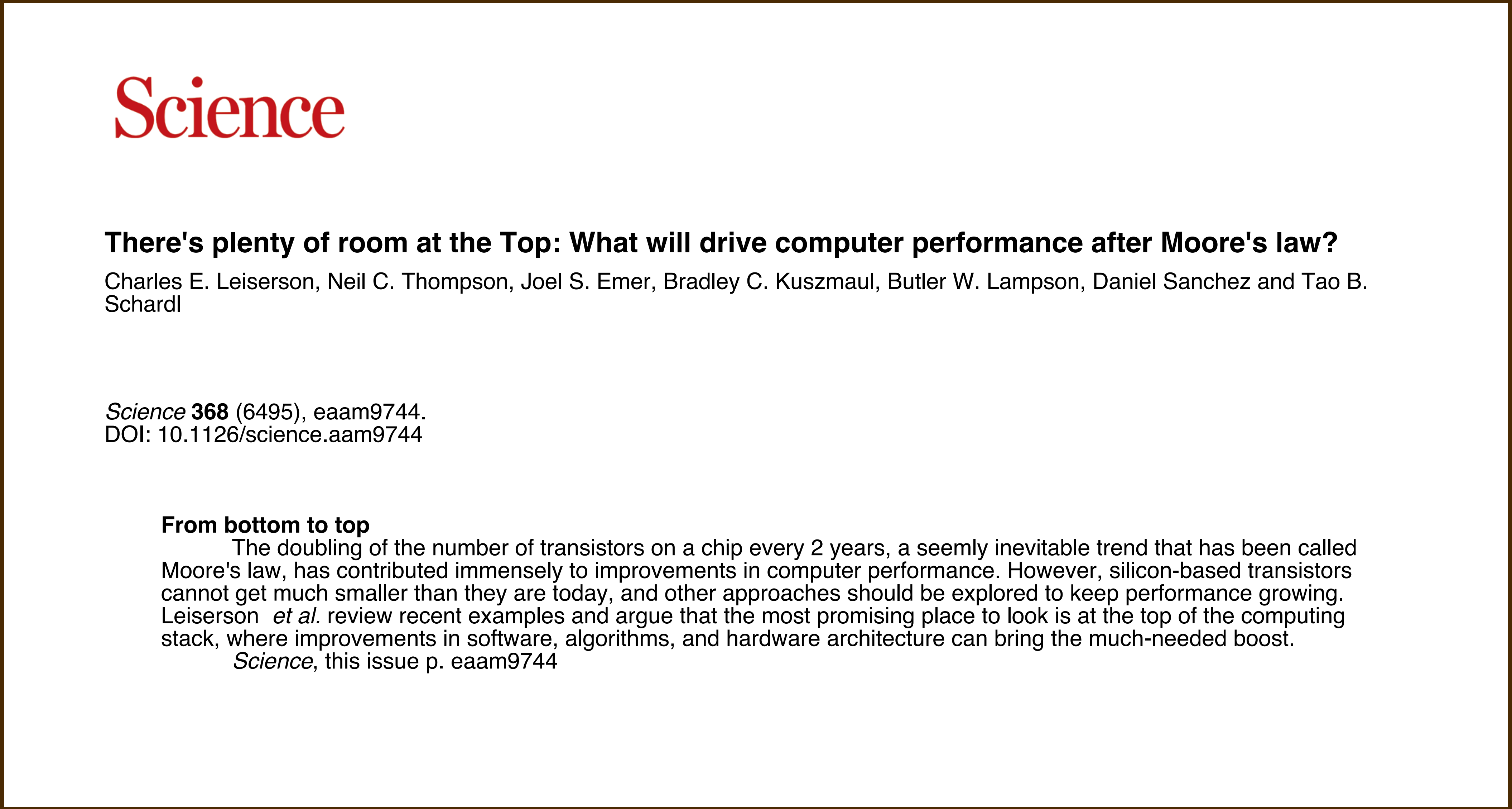
tinually be made smaller

[Source:
"The Free Lunch Is Over"
H. Sutter, 2009]

PARALLELISM: MULTICORE



OPPORTUNITIES FOR PERFORMANCE IMPROVEMENT



Science

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

Charles E. Leiserson, Neil C. Thompson, Joel S. Emer, Bradley C. Kuszmaul, Butler W. Lampson, Daniel Sanchez and Tao B. Schardl

Science **368** (6495), eaam9744.
DOI: 10.1126/science.aam9744

From bottom to top
The doubling of the number of transistors on a chip every 2 years, a seemly inevitable trend that has been called Moore's law, has contributed immensely to improvements in computer performance. However, silicon-based transistors cannot get much smaller than they are today, and other approaches should be explored to keep performance growing. Leiserson *et al.* review recent examples and argue that the most promising place to look is at the top of the computing stack, where improvements in software, algorithms, and hardware architecture can bring the much-needed boost.
Science, this issue p. eaam9744

OPPORTUNITIES FOR PERFORMANCE IMPROVEMENT

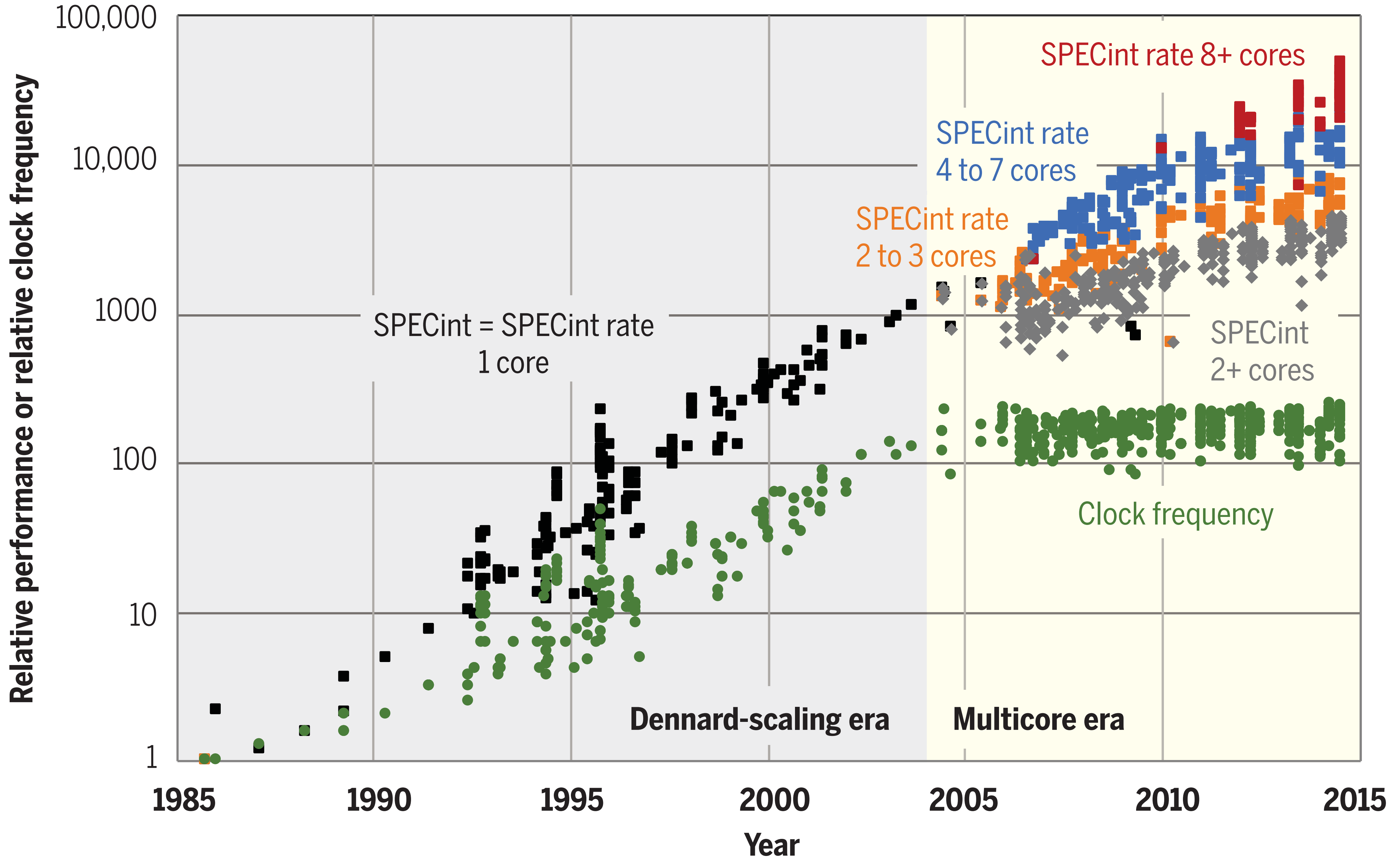
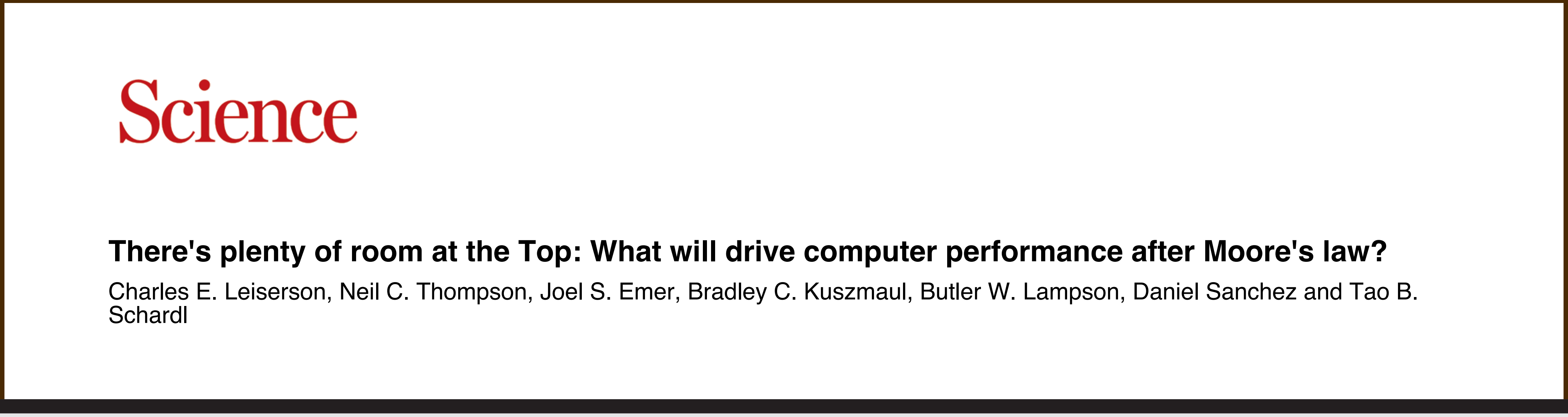


Fig. 2. SPECint (largely serial) performance, SPECint-rate (parallel) performance, and clock-frequency scaling for microprocessors from 1985 to 2015, normalized to the Intel 80386 DX microprocessor in 1985. Microprocessors and their clock frequencies were obtained from the Stanford CPU database (56).

OPPORTUNITIES FOR PERFORMANCE IMPROVEMENT



Science

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

Charles E. Leiserson, Neil C. Thompson, Joel S. Emer, Bradley C. Kuszmaul, Butler W. Lampson, Daniel Sanchez and Tao B. Schardl

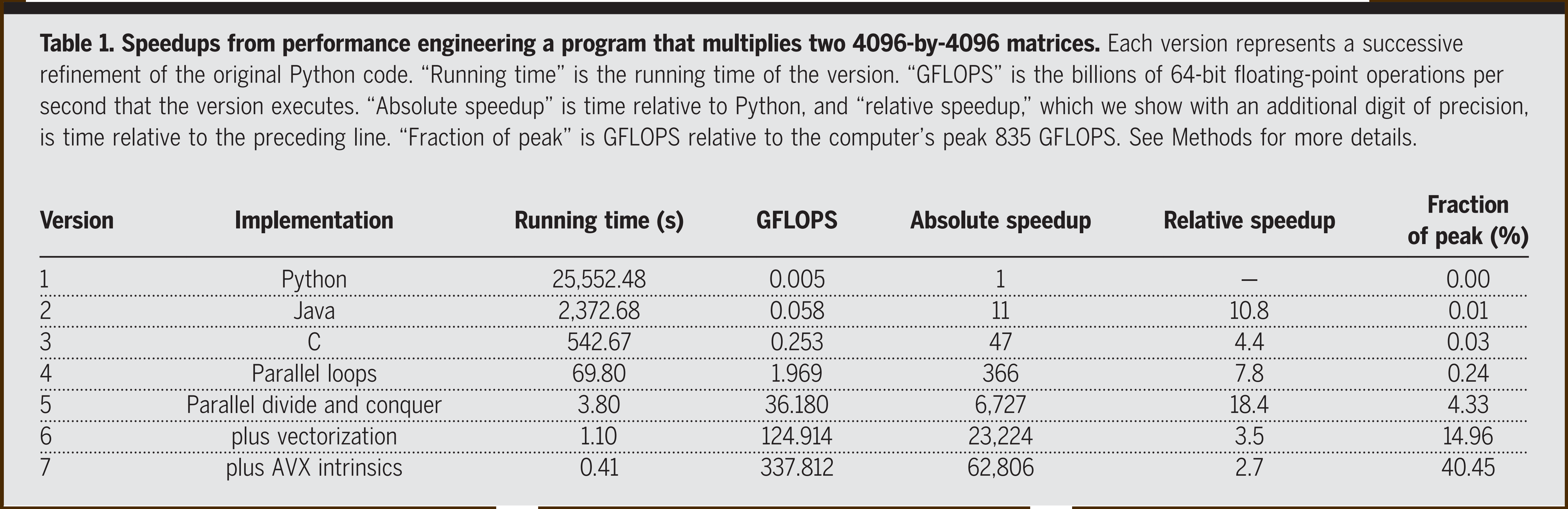
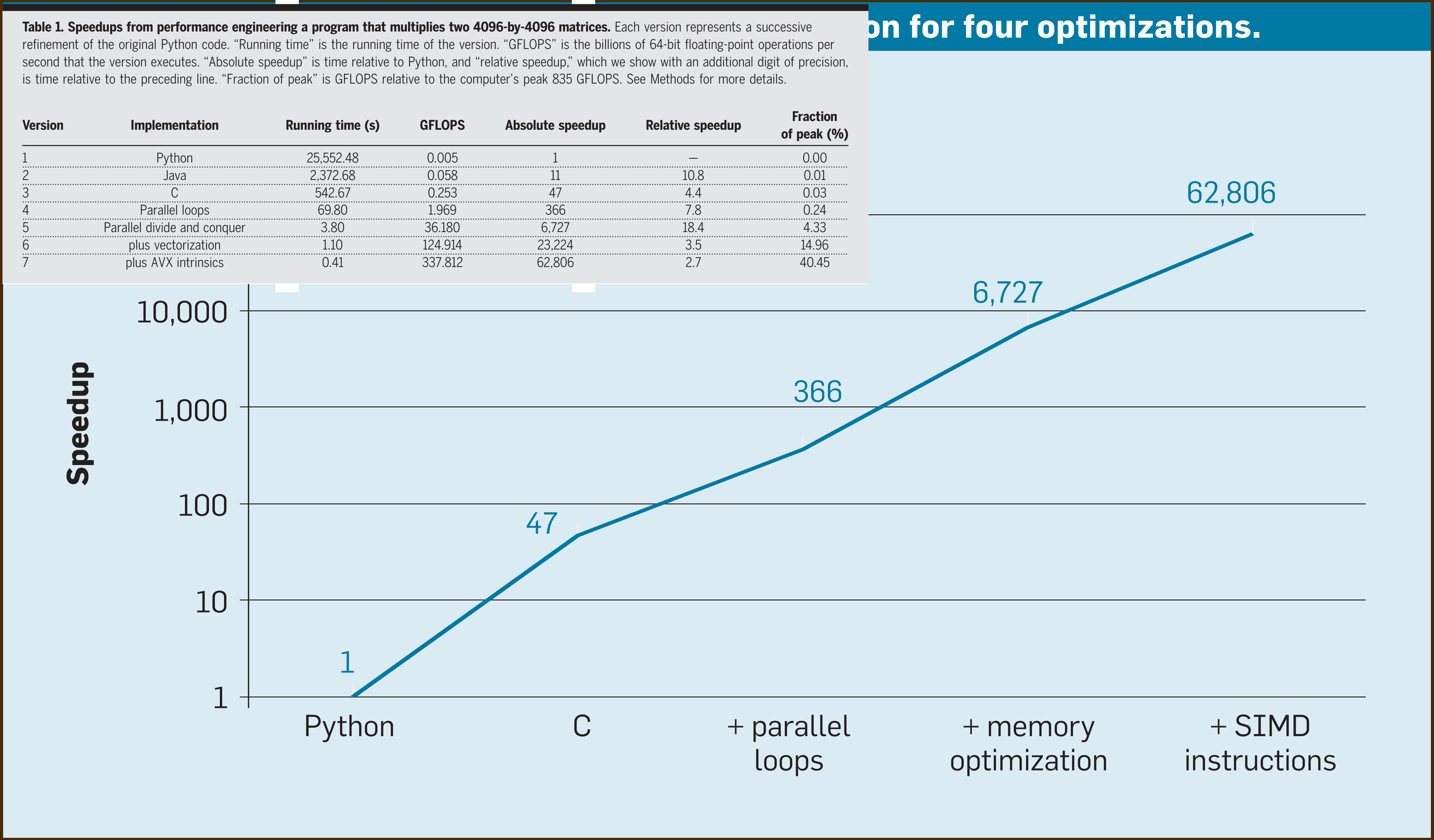


Table 1. Speedups from performance engineering a program that multiplies two 4096-by-4096 matrices. Each version represents a successive refinement of the original Python code. "Running time" is the running time of the version. "GFLOPS" is the billions of 64-bit floating-point operations per second that 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

[Source: Table 1 of "There's Plenty of Room at the Top:…",
C. Leiserson et al., *Science*, Jun 2020]

OPPORTUNITIES FOR PERFORMANCE IMPROVEMENT



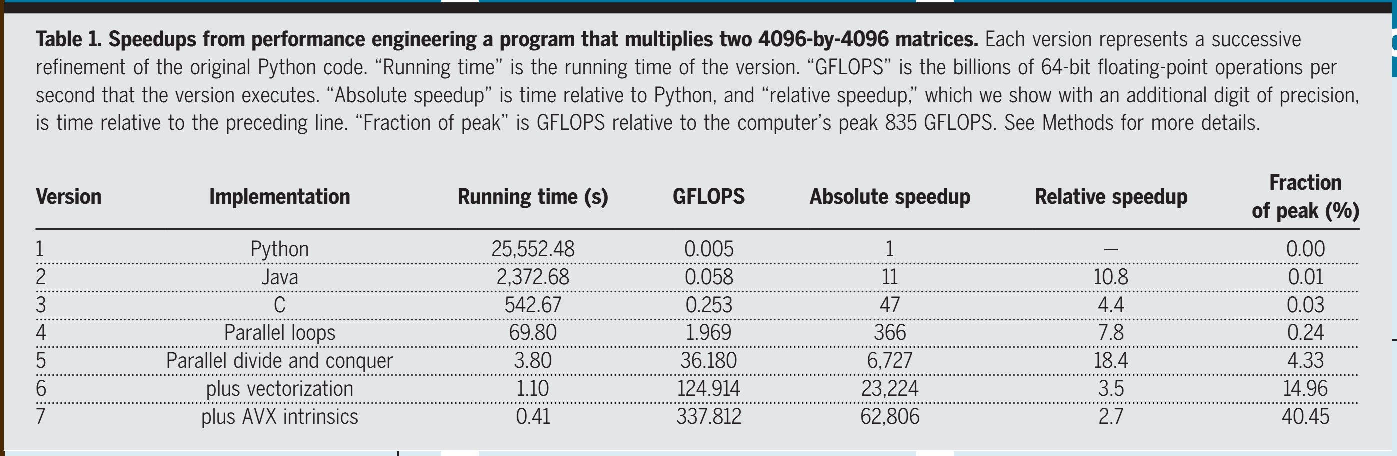


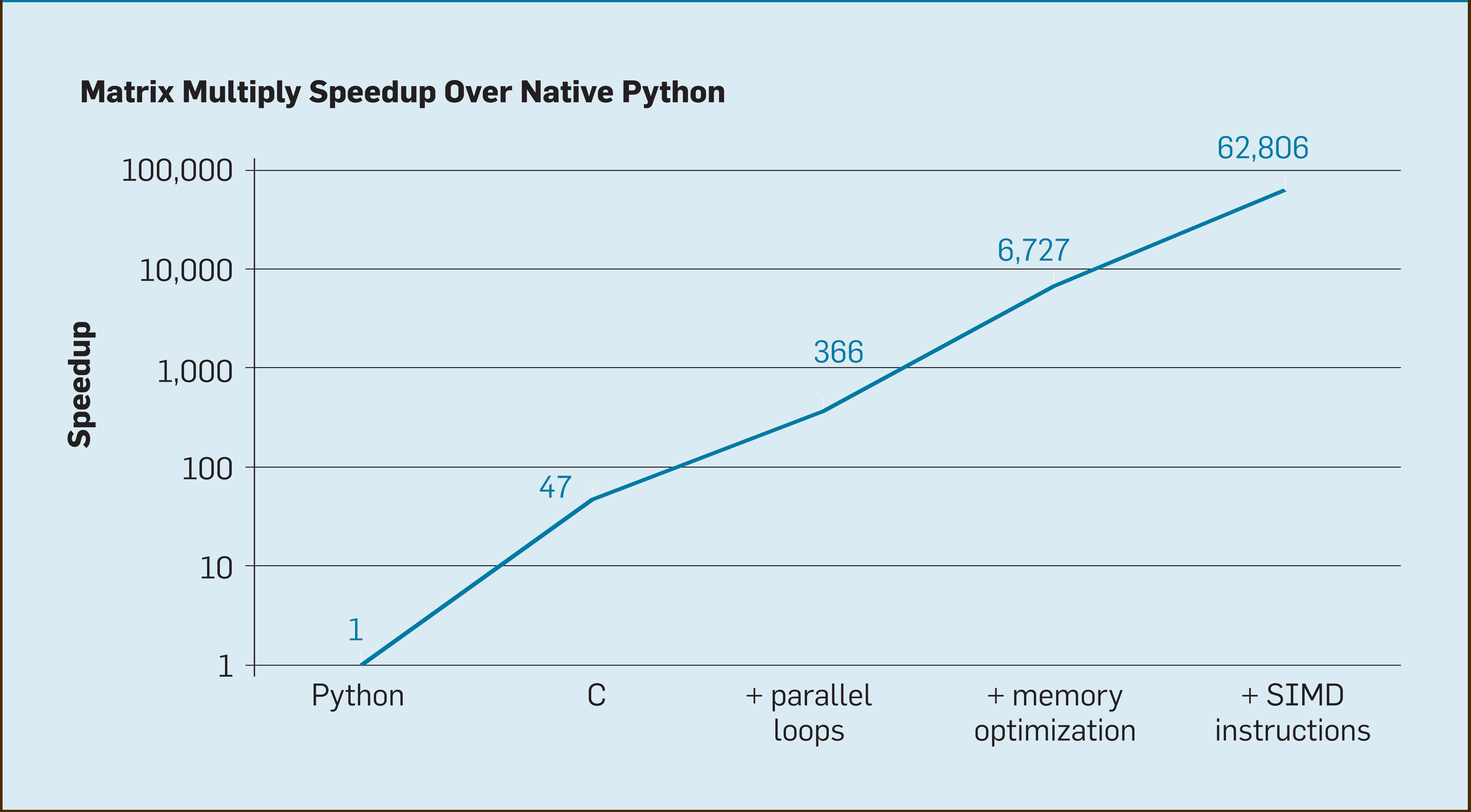
Table 1. Speedups from performance engineering a program that multiplies two 4096-by-4096 matrices. Each version represents a successive refinement of the original Python code. “Running time” is the running time of the version. “GFLOPS” is the billions of 64-bit floating-point operations per second that 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

[Source: Figure 7 of “A New Golden Age for Computer Architecture”, J. Hennesy, D. Patterson *Comm. of the ACM,* Feb 2019]

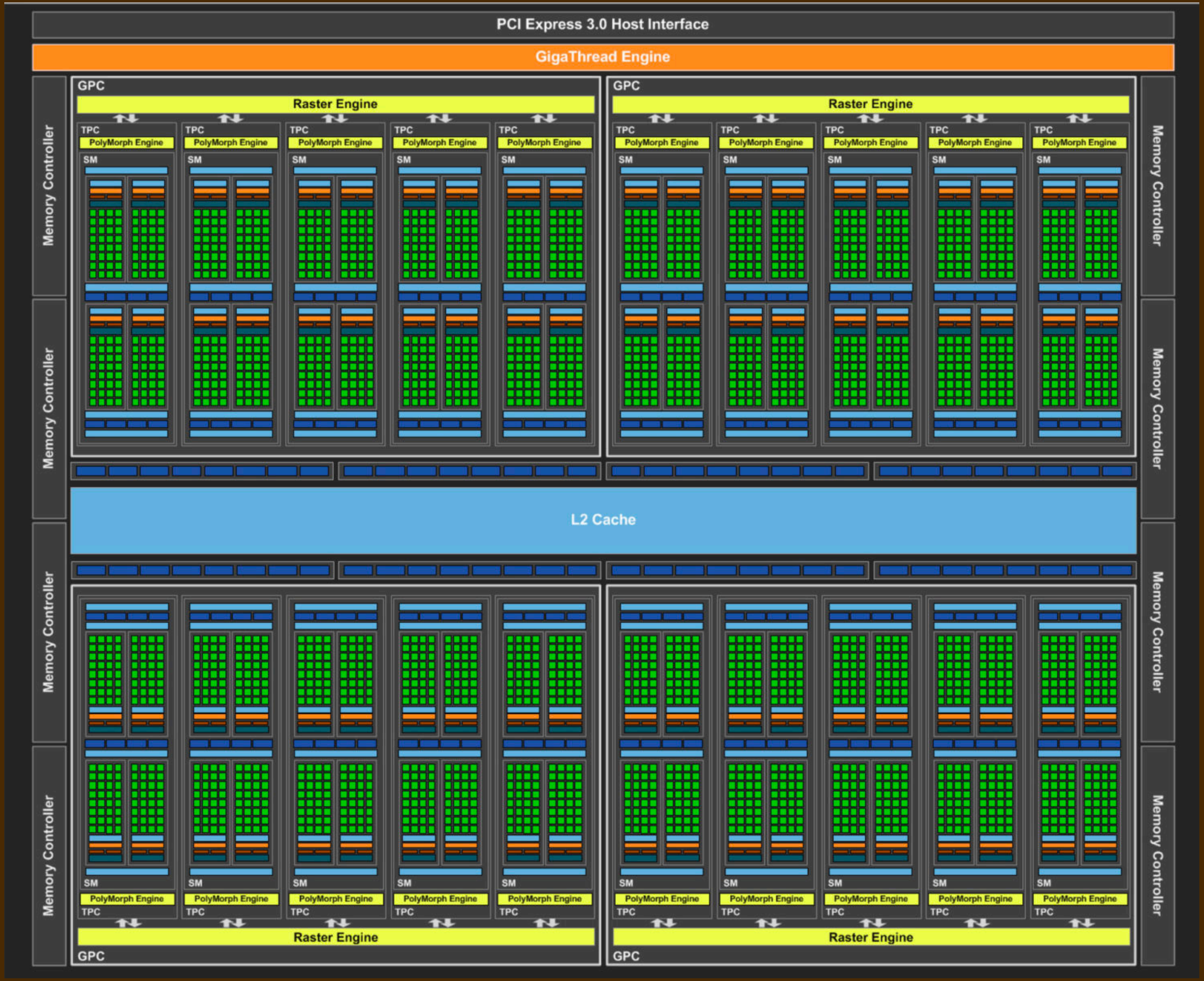
OPPORTUNITIES FOR PERFORMANCE IMPROVEMENT

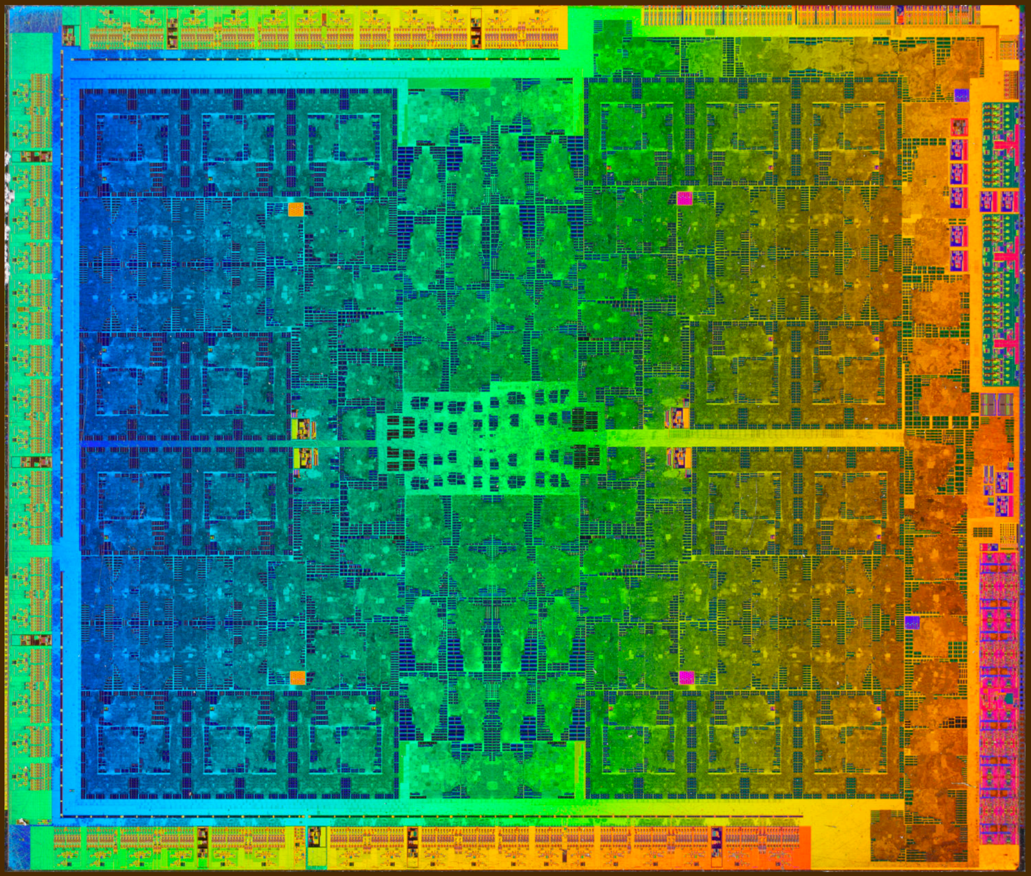
Figure 7. Potential speedup of matrix multiply in Python for four optimizations.



[Source: Figure 7 of "A New Golden Age for Computer Architecture",
J. Hennesy, D. Patterson *Comm. of the ACM,* Feb 2019; quoting C. Leiserson et al.

DOMAIN SPECIFIC PARALLELISM: GRAPHICS PROCESSOR



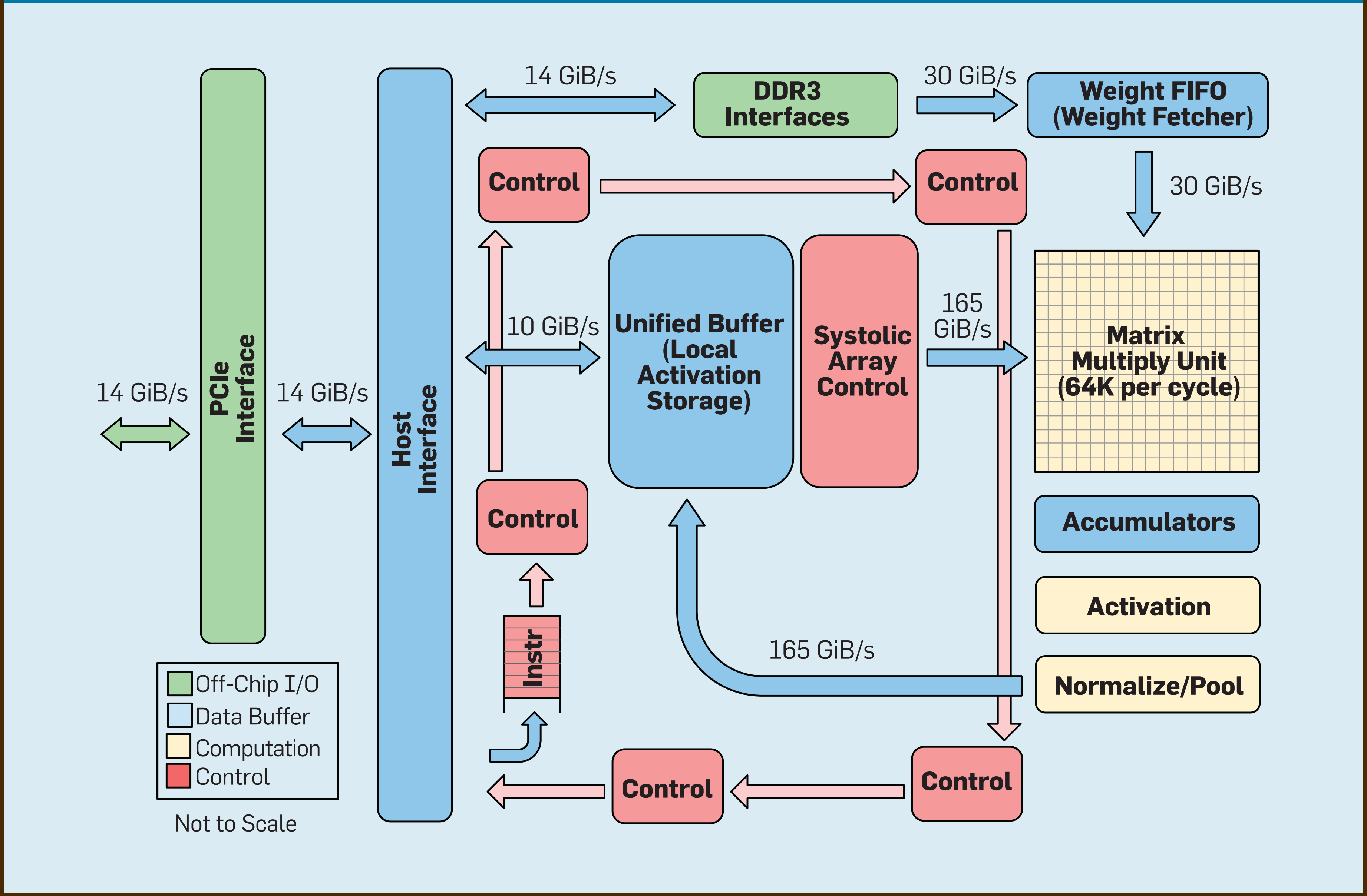


NVidia GTX 1080 GP104

- 2560 "CUDA" cores

DOMAIN-SPECIFIC PARALLELISM: GOOGLE'S TENSOR PROCESSOR

Figure 8. Functional organization of Google Tensor Processing Unit (TPU v1).



[Source: Figure 8 of "A New Golden Age for Computer Architecture",
J. Hennesy, D. Patterson *Comm. of the ACM,* Feb 2019]

PROGRAMMING FOR YOU BEFORE VERSUS NOW

CSCI 121 and CSCI 221 teach "*sequential programming*":

- Program does one thing at a time, in sequence.

In this course we start with *multithreaded programming*

- Structure computation using several threads of execution; coordinate them.
  - Seek to gain throughput, have parallel activity offer speedup.
  - Need to support concurrent access to data.

This creates interesting challenges and opportunities in program design.

EXAMPLE ALGORITHM: MERGE SORT

Let's "parallelize" a standard sorting algorithm…

[Reading: Chapter 27.3 of CLRS algorithms textbook]

SEMESTER TOPICS

- parallel merge sort, quick sort, radix sort
- parallel reduction/scan; map-reduce
- work-efficient parallel prefix
- fork-join model
- work and span analysis
- parallel RAM (PRAM) model
- algorithms on 1-D and 2-D arrays
- oblivious parallel sorting networks
- parallel graph algorithms
- parallel sequence analysis
- parallel task scheduling with work-stealing
- the Go language; "goroutines"; channels; synchronization
- pthreads C library; sychronization with mutexes and condition variables
- GPU programming and CUDA
- parallel complexity; Nick's class; P-completeness

RESPONSIBILITIES

- (roughly) bi-weekly homework assignment
  - ➡ written and coded
- final project and presentation

READING

- no text
- selected readings
  - papers and on-line materials

RESOURCES

- Can use your own computer to prototype
  - Go language, pthreads C library available on any system
- Should hopefully also gain access to patty.reed.edu and polly.reed.edu
  - Sitting in my office but maintained by **Cstar**
- Patty's specs:
  - 32 AMD Ryzen "threadripper" cores
  - NVidia GeForce RTX 2080 GPU
    - 3072 small processors organized as 192 stream multiprocessors