CHAPTER 1

Computer Abstractions and Technology

Chia-Heng Tu

Dept. of Computer Science and Information Engineering

National Cheng Kung University

國立成功大學
National Cheng Kung University

Outline

- Introduction
- Seven great ideas
- Below your program
- Under the covers
- Technologies for building processors/memory
- Performance
- The power wall
- Switch from uniprocessors to multiprocessors
- Real stuff
- Going faster
- Concluding Remarks

The Computer Revolution

- Progress in computer technology
  - Underpinned by domain-specific accelerators
- Makes novel applications feasible
  - Computers in automobiles
  - Cell phones
  - Human genome project
  - World Wide Web
  - Search Engines
- Computers are pervasive

Classes of Computers

- Personal computers
  - ➢General purpose, variety of software
  - ➢Subject to cost/performance tradeoff
- Embedded computers
  - ➢Hidden as components of systems
  - ➢Stringent power/performance/cost constraints
- Server computers
  - ➢Network based
  - ➢High capacity, performance, reliability
  - ➢Range from small servers to building sized
- Supercomputers
  - ➢Type of server
  - ➢High-end scientific and engineering calculations
  - ➢Highest capability but represent a small fraction of the overall computer market

The PostPC Era

- Personal Mobile Device (PMD)
  - ➢Battery operated
  - ➢Connects to the Internet
  - ➢Hundreds of dollars
  - ➢Smart phones, tablets, electronic glasses
- Cloud computing
  - ➢Warehouse Scale Computers (WSC)
  - ➢Software as a Service (SaaS)
  - ➢Portion of software run on a PMD and a portion run in the Cloud
  - ➢Major cloud service provider
    - ❖E.g., Amazon and Google

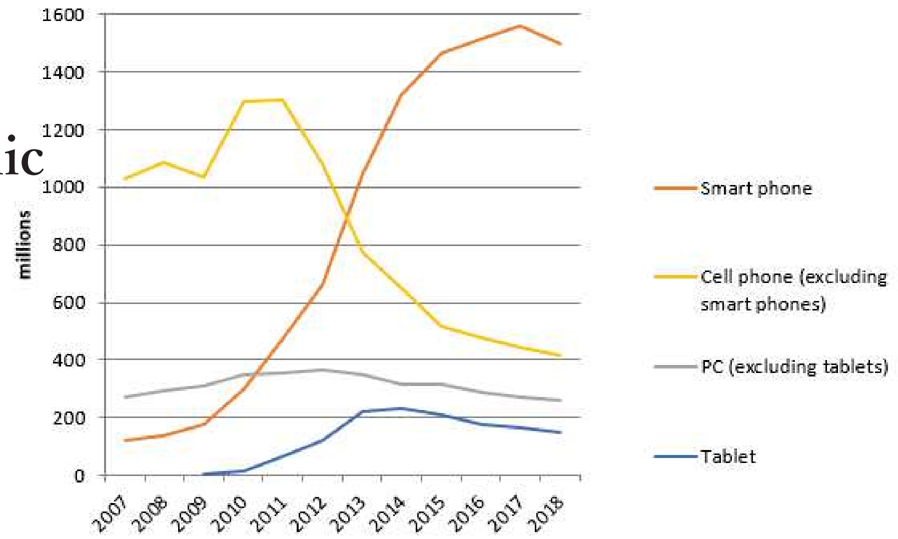


FIGURE 1.2 The number manufactured per year of tablets and smart phones, which reflect the post-PC era, versus personal computers and traditional cell phones. Smart phones represent the recent growth in the cell phone industry, and they passed PCs in 2011. PCs, tablets, and traditional cell phone categories are declining. The peak volume years are 2011 for cell phones, 2013 for PCs, and 2014 for tablets. PCs fell from 20% of total units shipped in 2007 to 10% in 2018.

What You Will Learn in This Class

- How programs are translated into the machine language
  - ➢and how the hardware executes them
- The hardware/software interface
- What determines program performance
  - ➢And how it can be improved
- How hardware designers improve performance
- Parallel processing concept

Understanding Performance

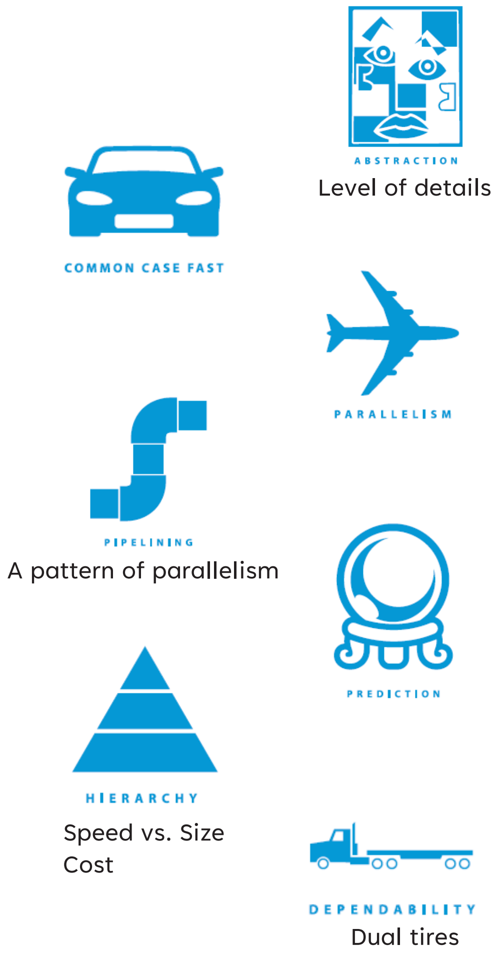
- The performance of a program depends on
  - a combination of the effectiveness of the algorithms used in the program,
  - the software systems used to create and translate the program into machine instructions, and
  - the *effectiveness* of the computer in executing those instructions,
  - which may include input/output (I/O) operations

Understanding Performance II

- Algorithm
  - ➢Determines number of computation/I/O operations executed
  - ➢It is covered in other classes
- Programming language, compiler, architecture
  - ➢Determine number of machine instructions executed per operation
  - ➢Chap. 2 and 3
- Processor and memory system
  - ➢Determine how fast instructions are executed
  - ➢Chap. 4, 5, and 6
- I/O system (including OS)
  - ➢Determines how fast I/O operations are executed
  - ➢Chap. 4, 5, and 6

Seven Great Ideas

- Use *abstraction* to simplify design
- Make the *common case fast*
- Performance *via parallelism*
- Performance *via pipelining*
- Performance *via prediction*
- *Hierarchy* of memories
- *Dependability via* redundancy



Below your program

- Application software
  - ➢Written in high-level language
- System software
  - ➢Compiler: translates HLL code to machine code
  - ➢Operating System: service code
  - ➢Handling input/output
  - ➢Managing memory and storage
  - ➢Scheduling tasks & sharing resources
- Hardware
  - ➢Processor, memory, I/O controllers

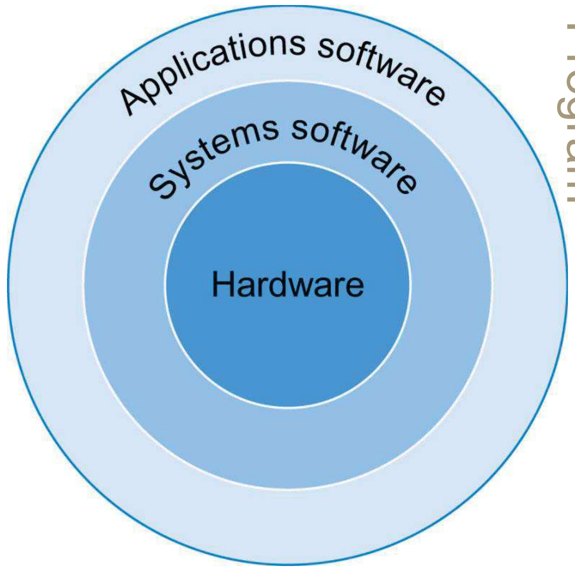
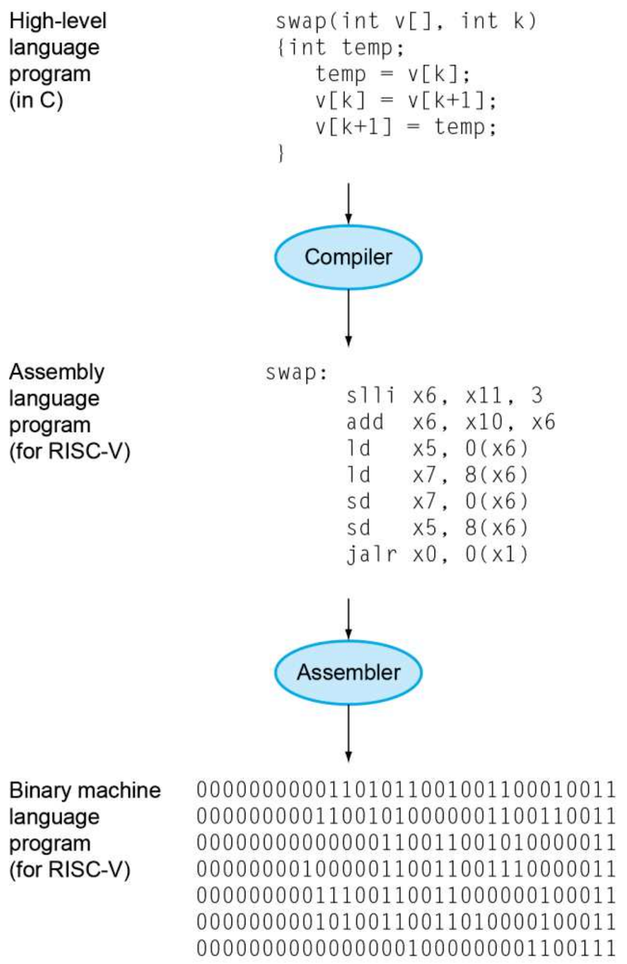


FIGURE 1.3 A simplified view of hardware and software as hierarchical layers, shown as concentric circles with hardware in the center and applications software outermost. In complex applications, there are often multiple layers of application software as well. For example, a database system may run on top of the systems software hosting an application, which in turn runs on top of the database.

Levels of Program Code

- High-level language
  - ➢Level of abstraction closer to problem domain
  - ➢Provides for productivity and portability
- Assembly language
  - ➢Textual representation (symbolic notation) of instructions
- Machine language
  - ➢Binary digits (bits)
  - ➢Encoded instructions and data


High-level language program (in C)
Compiler
Assembly language program (for RISC-V)
Assembler
Binary machine language program (for RISC-V)

```
swap(int v[], int k)
{int temp;
   temp = v[k];
   v[k] = v[k+1];
   v[k+1] = temp;
}
```

```
swap:
      slli x6, x11, 3
      add  x6, x10, x6
      ld   x5, 0(x6)
      ld   x7, 8(x6)
      sd   x7, 0(x6)
      sd   x5, 8(x6)
      jalr x0, 0(x1)
```

```
00000000001101011001001100010011
00000000011001010000001100110011
00000000000000110011001010000011
00000000100000110011001110000011
00000000011100110011000000100011
00000000010100110011010000100011
00000000000000001000000001100111
```

FIGURE 1.4 C program compiled into assembly language and then assembled into binary machine language. Although the translation from high-level language to binary machine language is shown in two steps, some compilers cut out the middleman and produce binary machine language directly. These languages and this program are examined in more detail in Chapter 2.

Components of a Computer

- Five classic components of a computer as shown in the figure
- Same components for all kinds of computer
  - ➢Desktop, server, embedded
- Input/output includes
  - ➢User-interface devices
    - ❖Display, keyboard, mouse
  - ➢Storage devices
    - ❖Hard disk, CD/DVD, flash
  - ➢Network adapters
    - ❖For communicating with other computers

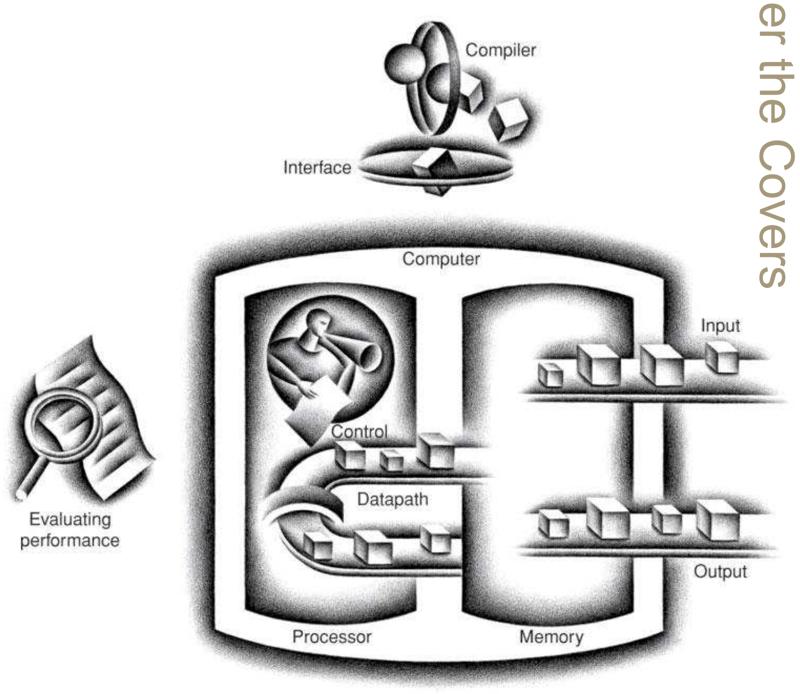


FIGURE 1.5 The organization of a computer, showing the **five** classic components. The processor gets instructions and data from memory. Input writes data to memory, and output reads data from memory. Control sends the signals that determine the operations of the datapath, memory, input, and output.

Input and Output Examples

INPUT

- PostPC devices
- Supersedes keyboard and mouse
- Resistive and Capacitive types
  - ➢Most tablets, smart phones use capacitive
  - ➢Capacitive allows multiple touches simultaneously

OUTPUT

- LCD screen: picture elements (pixels)
  - ➢Mirrors content of frame buffer memory

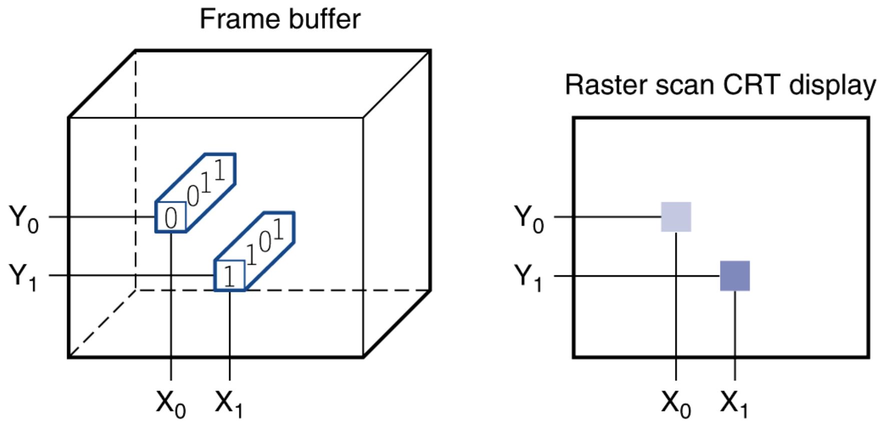


FIGURE 1.6 Each coordinate in the frame buffer on the left determines the shade of the corresponding coordinate for the raster scan CRT display on the right. Pixel (X0, Y0) contains the bit pattern 0011, which is a lighter shade on the screen than the bit pattern 1101 in pixel (X1, Y1).

Key Components

- I/O dominates the device (iPhone XS Max)
  - ➢LCD display, cameras, microphone, speakers, etc.
- Apple A12 package
  - ➢contains 2 large and 4 little ARM processors (for performing computations)
  - ➢with 4GB LPDDDR4x SDRAM (for keeping instructions and data of a program)

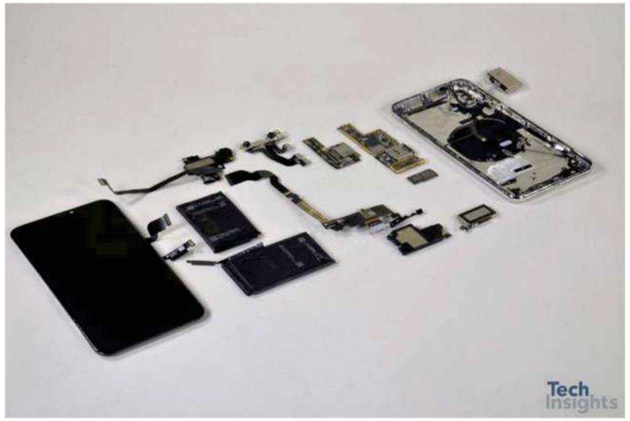


FIGURE 1.7 Components of the Apple iPhone XS Max cell phone. ...

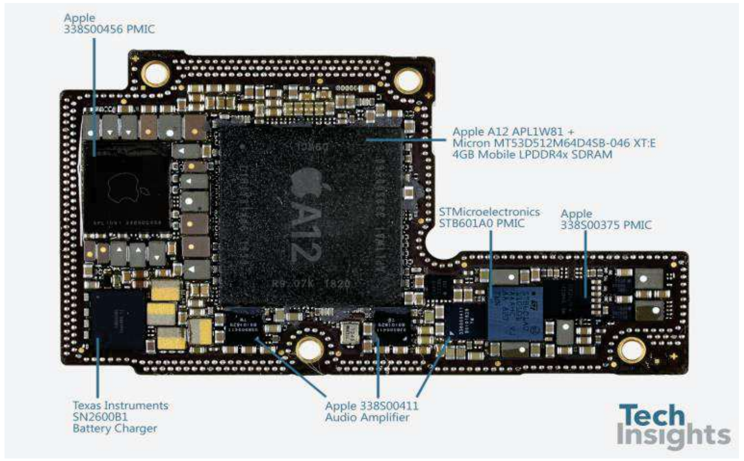

Apple 338S00456 PMIC
Apple A12 APL1W81 + Micron MT53D512M64D4SB-046 XT:E 4GB Mobile LPDDR4x SDRAM
STMicroelectronics STB601A0 PMIC
Apple 338S00375 PMIC
Texas Instruments SN2600B1 Battery Charger
Apple 338S00411 Audio Amplifier
A12
Tech Insights

FIGURE 1.8 The logic board of Apple iPhone XS Max in Figure 1.7. The large integrated circuit in the middle is the Apple A12 chip, which contains two large and four small ARM processor cores that run at 2.5 GHz, as well as 2 GiB of main memory inside the package. Figure 1.9 shows a photograph of the processor chip inside the A12 package. A similar-sized chip on a symmetric board that attaches to the back is a 64 GiB flash memory chip for nonvolatile storage. The other chips on the board include the power management integrated controller and audio amplifier chips.

The Processor Internal

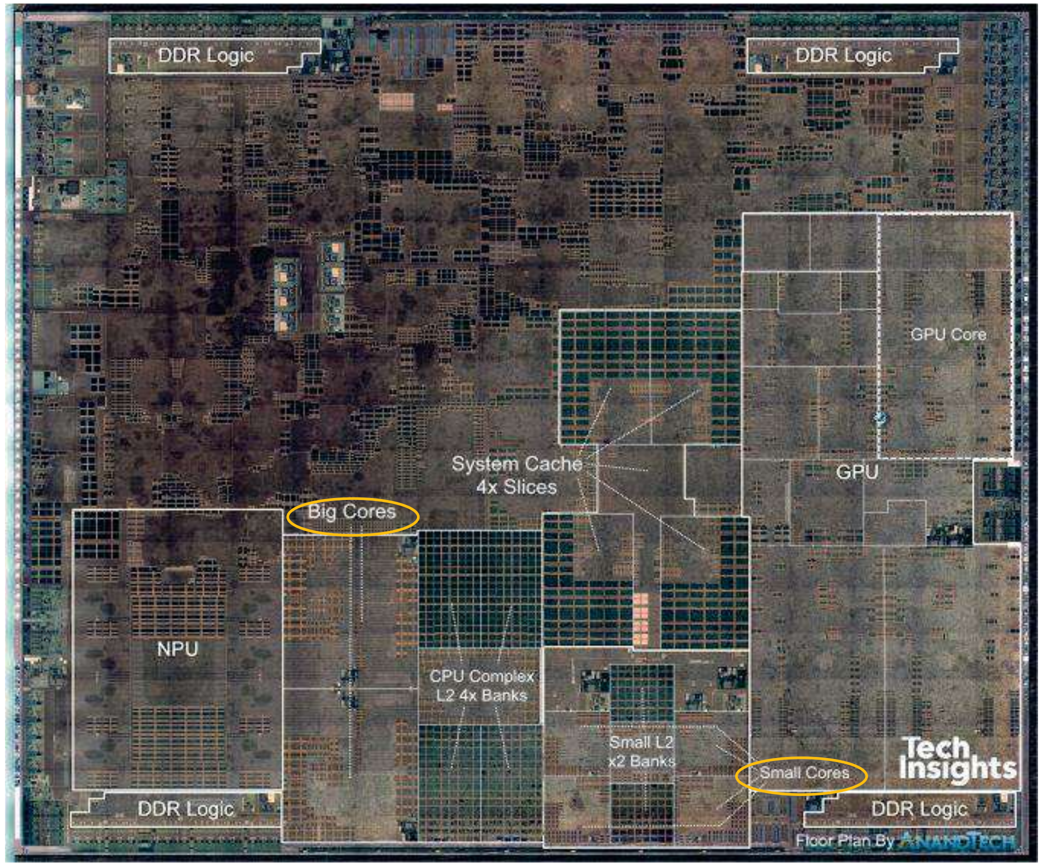


FIGURE 1.9 The processor integrated circuit inside the A12 package. The size of chip is 8.4 by 9.91 mm, and it was manufactured originally in a 7-nm process (see Section 1.5). It has two identical ARM processors or cores in the lower middle of the chip, four small cores on the lower right of the chip, a graphics processing unit (GPU) on the far right (see Section 6.6), and a domain-specific accelerator for neural networks (see Section 6.7) called the NPU on the far left. In the middle are second-level cache memory (L2) banks for the big and small cores (see Chapter 5). At the top and bottom of the chip are interfaces to the main memory (DDR DRAM).

Abstractions

The BIG Picture

Both hardware and software consist of **hierarchical layers** using *abstraction*, with each lower layer hiding details from the level above. One key interface between the levels of abstraction is the instruction set architecture—the interface between the hardware and low-level software. This abstract interface enables many implementations of varying cost and performance to run identical software.

- Abstraction helps us deal with complexity
  - ➢Hide lower-level detail
- Instruction set architecture (ISA)
  - ➢The hardware/software interface
  - ➢E.g., RISC-V instructions
- Application Binary Interface (ABI)
  - ➢The ISA plus system software interface
  - ➢Binary portability across computers
  - ➢E.g., RISC-V instructions (user mode) + Linux system calls
- Other *interfaces*
  - ➢The details hidden in underlying, various *implementations*

A safe place for data

- Volatile main memory
  - ➢Loses instructions and data when power off

- Non-volatile secondary memory
  - ➢Flash memory
  - ➢Magnetic disk
  - ➢Optical disk (CDROM, DVD)

- Memory hierarchy
  - ➢Cache memory (SRAM)
  - ➢Primary memory (DRAM)
  - ➢Secondary memory (Flash memory)
  - ➢Each has a different size, speed, and cost

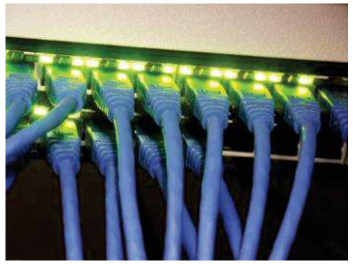
Courtesy of SSD IMG

Communications w/ Other Computers

- Communication
  - ➢Information is exchanged between computers at high speeds
- Resource sharing
  - ➢Computers on the network can share I/O devices to some other computers
- Nonlocal access
  - ➢By connecting computers over long distances, users need not be near the computer they are using

Networks

- Local area network (LAN)
  - ➢Ethernet
- Wide area network (WAN)
  - ➢The Internet
- Wireless network
  - ➢WiFi, Bluetooth



Technology Trends

- Electronics technology continues to evolve
  - ➢Increased capacity and performance
  - ➢Reduced cost

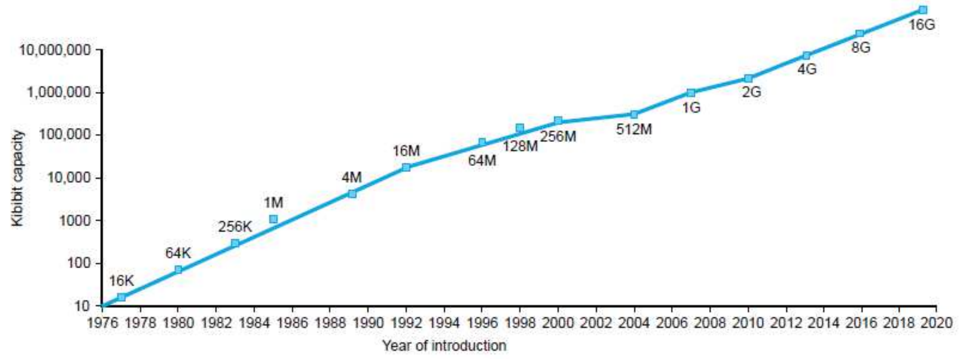


FIGURE 1.11 Growth of capacity per DRAM chip over time. The y-axis is measured in kibibits (210 bits). The DRAM industry quadrupled capacity almost every three years, a 60% increase per year, for 20 years. In recent years, the rate has slowed down and is somewhat closer to doubling every three years. With the slowing of Moore's Law and difficulties in reliable manufacturing of smaller DRAM cells given the challenging aspect ratios of their three-dimensional structure.

Year	Technology	Relative performance/cost
1951	Vacuum tube	1
1965	Transistor	35
1975	Integrated circuit (IC)	900
1995	Very large scale IC (VLSI)	2,400,000
2013	Ultra large scale IC	250,000,000,000
2020	Ultra large-scale IC	500,000,000,000

FIGURE 1.10 Relative performance per unit cost of technologies used in computers over time. Source: Computer Museum, Boston, with 2013 extrapolated by the authors. See Section 1.13.

Semiconductor Technology for Manufacturing ICs

- Silicon semiconductor
- Add materials to transform properties
  - ➢Conductors
  - ➢Insulators
  - ➢Switch

- *Yield* refers to the proportion of working dies per wafer
  - ❖A microscopic flaw in a wafer or in patterning steps can result in that area of the wafer failing (called defect)
  - ❖Dice the patterned wafers into dies to deal with the *defects*

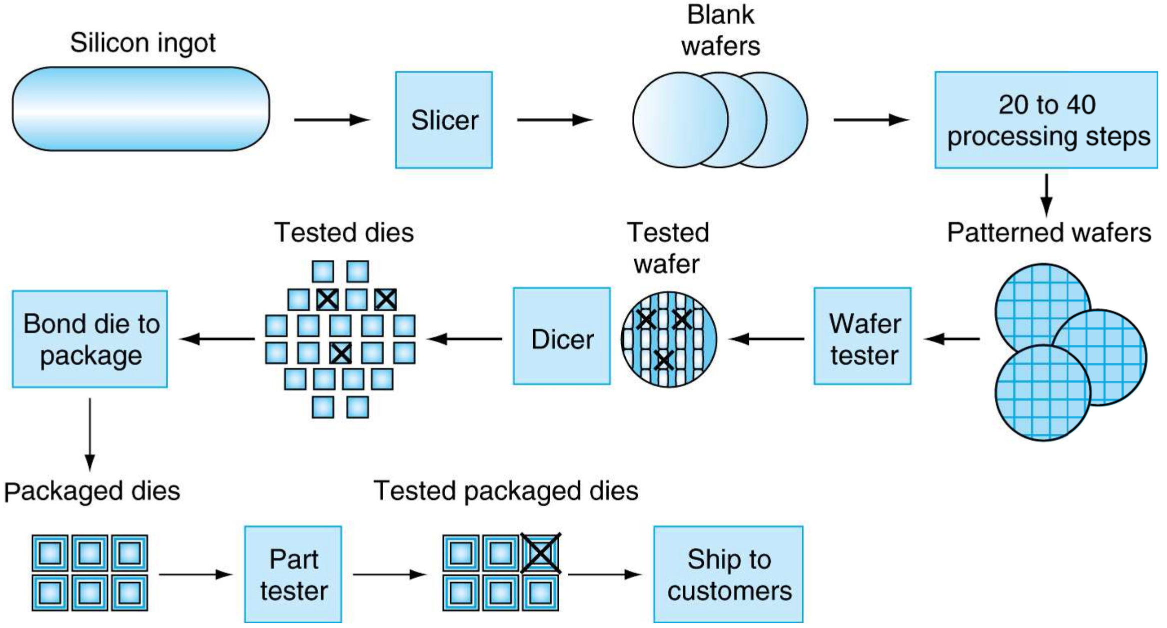


FIGURE 1.12 The chip manufacturing process. After being sliced from the silicon ingot, blank wafers are put through 20 to 40 steps to create patterned wafers (see Figure 1.13). These patterned wafers are then tested with a wafer tester, and a map of the good parts is made. Next, the wafers are diced into dies (see Figure 1.9). In this figure, one wafer produced 20 dies, of which 17 passed testing. (X means the die is bad.) The ***yield*** of good dies in this case was 17/20, or 85%. These good dies are then bonded into packages and tested one more time before shipping the packaged parts to customers. One bad packaged part was found in this final test.

Wafer of Intel Core Processor

- A 300mm wafer
  - ➢506 chips
  - ➢10nm technology
- Each chip is 11.4 x 10.7 mm

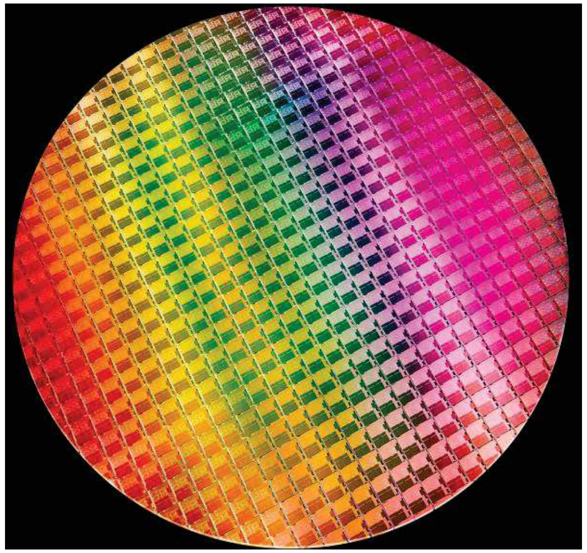


FIGURE 1.13 A 12-inch (300mm) wafer this 10nm wafer contains 10th Gen Intel® Core processors, code-named “Ice Lake” (Courtesy Intel). The number of dies on this 300 mm (12 inch) wafer at 100% yield is 506. According to AnandTech, each Ice Lake die is 11.4 by 10.7 mm. The several dozen partially rounded chips at the boundaries of the wafer are useless; they are included because it’s easier to create the masks used to pattern the silicon. This die uses a 10-nanometer technology, which means that the smallest features are approximately 10 nm in size, although they are typically somewhat smaller than the actual feature size, which refers to the size of the transistors as “drawn” versus the final manufactured size.

Integrated Circuit Cost

- Nonlinear relation to area and defect rate
  - ➢Wafer cost and area are fixed
  - ➢Defect rate determined by manufacturing process
  - ➢Die area determined by architecture and circuit design

- The first equation is straightforward to derive
- The second is an approximation, since it does not subtract the area near the border of the round wafer
- The final equation is based on empirical observations of yields at integrated circuit factories, with the exponent related to the number of critical processing steps
- Hence, depending on the defect rate and the size of the die and wafer, costs are generally not linear in the die area

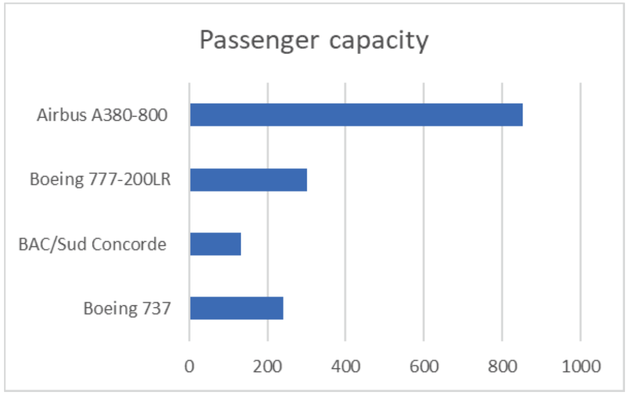
$$\text{Cost per die} = \frac{\text{Cost per wafer}}{\text{Dies per wafer} \times \text{yield}}$$

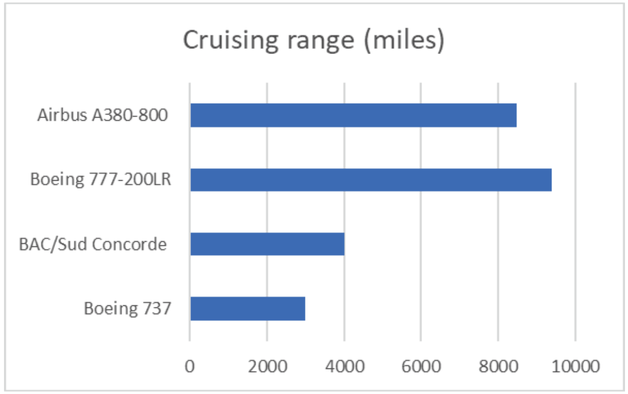
$$\text{Dies per wafer} \approx \frac{\text{Wafer area}}{\text{Die area}}$$

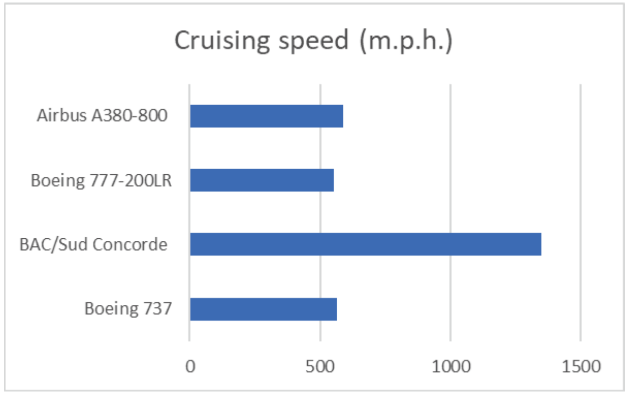
$$\text{Yield} = \frac{1}{(1 + (\text{Defects per area} \times \text{Die area}))^N}$$

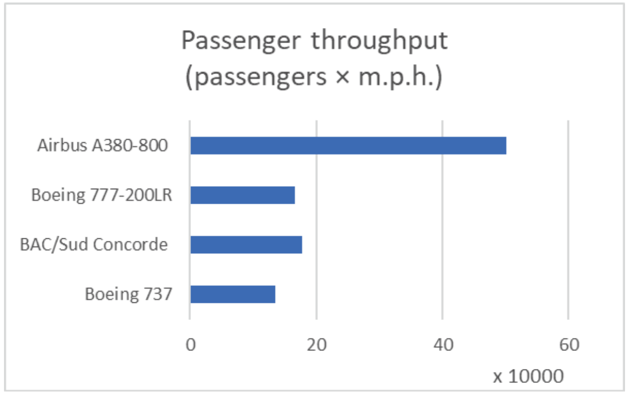
Performance

- Which airplane has the best performance?
  - Four airplanes: A380-800, 777-200LR, Concorde, and 737
- The charts provide their performance in terms of:
  - Capacity, cruising range, speed, and throughput









Performance Goal

- Response time
  - ➢How long it takes to do a task
  - ➢E.g., 10 seconds to finish the task
  - ➢Also called execution time, the total time required for the computer to complete a task, including disk accesses, memory accesses, I/O activities, operating system overhead, CPU execution time, and so on

- Throughput (or bandwidth)
  - ➢Total work done per unit time
  - ➢E.g., completed tasks (transactions) per hour

- How are response time and throughput affected by
  - ➢Replacing the processor with a faster version?
  - ➢Adding more processors?

- Focus on response time in the following slides

Relative Performance

- What is Performance?

  Performance = 1/Execution Time

- "X is n time faster than Y"

  $$\text{Performance}_X / \text{Performance}_Y$$
  $$= \text{Execution time}_Y / \text{Execution time}_X = n$$

- Example: time taken to run a program
  - 10s on A, 15s on B
  - Execution Time_B / Execution Time_A = 15s / 10s = 1.5
  - Thus, A is 1.5 times faster than B

Execution Time

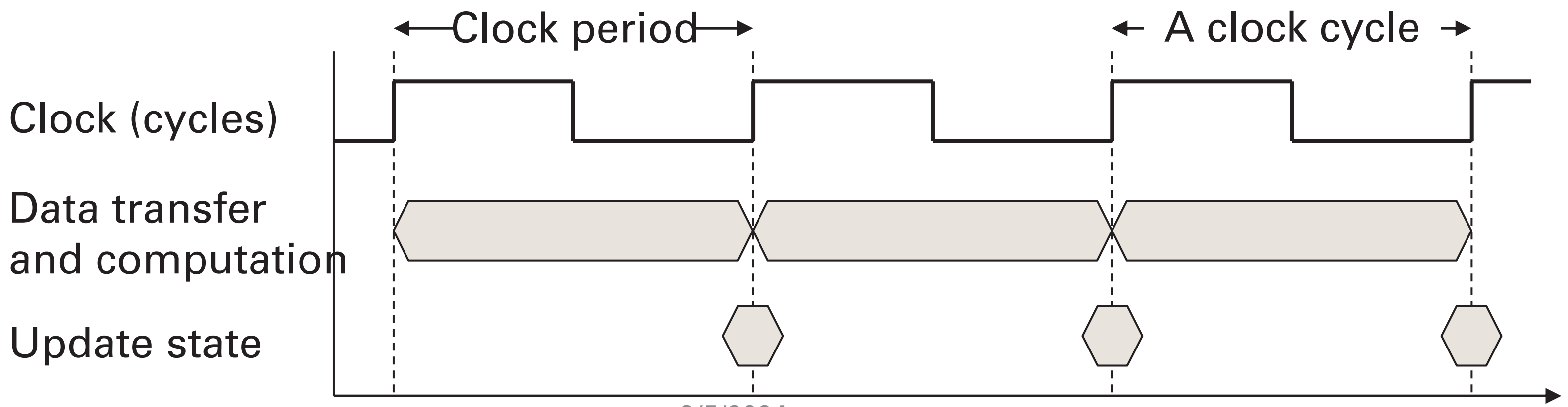
- Elapsed time
  - ➢Total response time includes *all* aspects: CPU processing, I/O, OS overhead, and idle time
  - ➢Determines system performance
  - ➢I.e., <u>the time between the start and completion of a task</u>
  - ➢E.g., the time you tap the touch screen and the time that the screen displays the result you want
  - ➢Total response time is usually measured by wall clock time

- CPU (execution) time
  - ➢Time spent on CPU for processing a given job
  - ➢Comprises user CPU time and system CPU time
  - ➢Discounts I/O time, other jobs' shares
  - ➢Different programs are affected differently by CPU & system performance

CPU time can be further divided into the CPU time spent in the program, called user CPU time, and the CPU time spent in the operating system performing tasks on behalf of the program, called system CPU time.

CPU Clocks

- CPU has a constant-rate *clock* that determines when events take place in the hardware
  - ➢The discrete time intervals are called clock cycles (or ticks, clock ticks, clock periods, clocks, cycles)
- The length of a clock period can be referred to as:
  - ➢the time for a complete clock cycle
  - ➢E.g., 250ps (picoseconds) = 0.25ns = 250×10^{-12}s
  - ➢the clock rate (frequency), which is the inverse of the clock period; cycles per second
  - ➢E.g., 4.0GHz (gigahertz) = 4,000MHz = 4.0×10^{9}Hz



CPU Time

- CPU time formula suggests that performance can be improved by
  - ➢reducing number of clock cycles required by a program or
  - ➢increasing clock rate (reducing the length of the CPU clock cycle)

- Hardware designer must often trade off <u>clock rate</u> against <u>cycle count</u> (needed for a program)

$$\text{CPU Time} = \text{CPU Clock Cycles} \times \text{Clock Cycle Time}$$
$$= \frac{\text{CPU Clock Cycles}}{\text{Clock Rate}}$$

Improving Performance Example

- It takes 10s (CPU time) to run a program on Computer A @ 2GHz
- Goal is to build Computer B, which runs the program in 6s
  - ➢The designer has determined that a substantial increase in the clock rate is possible, but this increase will affect the rest of the CPU design, causing computer B to require 1.2 times as many clock cycles as computer A for this program
- How fast must Computer B clock be?

$$\text{Clock Rate}_B = \frac{\text{Clock Cycles}_B}{\text{CPU Time}_B} = \frac{1.2 \times \text{Clock Cycles}_A}{6\text{s}}$$

$$\text{Clock Cycles}_A = \text{CPU Time}_A \times \text{Clock Rate}_A$$

$$= 10\text{s} \times 2\text{GHz} = 20 \times 10^9$$

$$\text{Clock Rate}_B = \frac{1.2 \times 20 \times 10^9}{6\text{s}} = \frac{24 \times 10^9}{6\text{s}} = 4\text{GHz}$$

CPU Time and Instruction Count

- We relate instructions needed for a program to the CPU clock cycles
  - ➢since the compiler clearly generated instructions to execute, and the computer had to execute the instructions to run the program
- Instruction Count for a program
  - ➢Determined by program, ISA, and compiler
- Cycles per instruction (CPI)
  - ➢Determined by CPU hardware
  - ➢If different instructions have different CPI
  - ➢Average CPI affected by *instruction mix*

$$\text{Clock Cycles} = \text{Instruction Count} \times \text{Cycles per Instruction}$$

$$\text{CPU Time} = \text{Instruction Count} \times \text{CPI} \times \text{Clock Cycle Time}$$

$$= \frac{\text{Instruction Count} \times \text{CPI}}{\text{Clock Rate}}$$

Using Performance Equation Example

- Same ISA for Computers A and B
  - ➢Computer A: Cycle Time = 250ps, CPI = 2.0, for Prog X
  - ➢Computer B: Cycle Time = 500ps, CPI = 1.2, for Prog X

- Which is faster, and by how much?
  - ➢Recall CPU time = Inst. Count * CPI * clock cycle time

$$\text{CPU Time}_A = \text{Instruction Count} \times \text{CPI}_A \times \text{Cycle Time}_A$$

$$= \text{I} \times 2.0 \times 250\text{ps} = \text{I} \times 500\text{ps}$$

$$\text{CPU Time}_B = \text{Instruction Count} \times \text{CPI}_B \times \text{Cycle Time}_B$$

$$= \text{I} \times 1.2 \times 500\text{ps} = \text{I} \times 600\text{ps}$$

$$\frac{\text{CPU Time}_B}{\text{CPU Time}_A} = \frac{\text{I} \times 600\text{ps}}{\text{I} \times 500\text{ps}} = 1.2$$

A is faster…

…by this much

More about Performance Equation

- In practice, a program would contain different classes of instructions (instruction mix)
- Different instruction classes take different numbers of cycles
  - ➢In such a case, the <u>total CPU clock cycles</u> for the program becomes:

$$\text{Clock Cycles} = \sum_{i=1}^{n} (\text{CPI}_i \times \text{Instruction Count}_i)$$

  - ➢Also, the <u>average CPI</u> for the program is:

$$\text{CPI} = \frac{\text{Clock Cycles}}{\text{Instruction Count}} = \sum_{i=1}^{n} \left(\text{CPI}_i \times \underbrace{\frac{\text{Instruction Count}_i}{\text{Instruction Count}}}_{\substack{\text{Relative frequency for} \\ i\text{-type instructions}}} \right)$$

Another CPU Time Example

- Compiler designer tries to decide between two code sequences for a computer
  - ➢The CPI of the computer and the instruction counts (ICs) for the two sequences are listed below

Class of instructions	A	B	C
CPI for class	1	2	3
IC in Sequence 1	2	1	2
IC in Sequence 2	4	1	1

- Sequence 1: IC = 5
  - Clock Cycles $= 2\times1 + 1\times2 + 2\times3 = 10$
  - Avg. CPI $= 10/5 = 2.0$
- Sequence 2: IC = 6
  - Clock Cycles $= 4\times1 + 1\times2 + 1\times3 = 9$
  - Avg. CPI $= 9/6 = 1.5$

- Questions
  - ➢Which code sequence executes the most instructions?
  - ➢Which will be faster?
  - ➢What is the CPI for each sequence?

Measuring Computer Performance

- Time is the only complete and reliable measure of computer performance

$$\text{CPU Time} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Clock cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Clock cycle}}$$

- Performance depends on many factors
  - ➢Algorithm: affects IC, possibly CPI
  - ➢Programming language: affects IC, CPI
  - ➢Compiler: affects IC, CPI
  - ➢Instruction set architecture: affects IC, CPI, clock rate

  - ➢Please refer to Sec. 1.5 of the textbook for more details

The Power Wall (in CMOS Technology)

- Clock rate and power increased rapidly for decades and then flattened/dropped off in the past few years
- The reason for their recent slowing is that we have run into the practical power limit for cooling commodity microprocessors (to reduce heat)
- How to reduce power consumption?

5V → 1V

$$\text{Power} = \text{Capacitive load} \times \text{Voltage}^2 \times \text{Frequency}$$

×288
(3,600/12.5)

×30

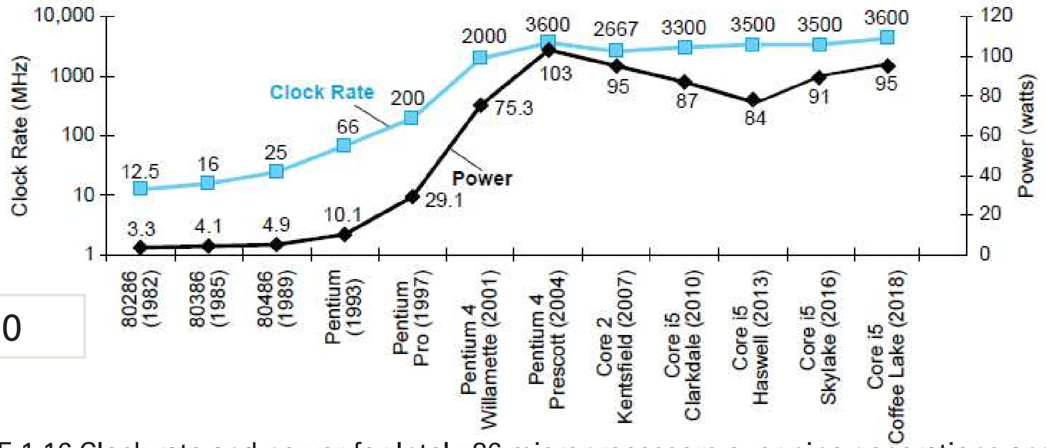


FIGURE 1.16 Clock rate and power for Intel x86 microprocessors over nine generations and 36 years. The Pentium 4 made a dramatic jump in clock rate and power but less so in performance. The Prescott thermal problems led to the abandonment of the Pentium 4 line. The Core 2 line reverts to a simpler pipeline with lower clock rates and multiple processors per chip. The Core i5 pipelines follow in its footsteps.

For CMOS, the primary source of energy consumption is so-called dynamic energy. Energy that is consumed when transistors switch states from 0 to 1 and vice versa.

Static energy consumption occurs because of leakage current that flows even when a transistor is off.
In servers, leakage is typically responsible for 40% of the energy consumption.
Increasing the number of transistors increases power dissipation, even if the transistors are always off.

Switch from Uniprocessors to Multiprocessors. Why?

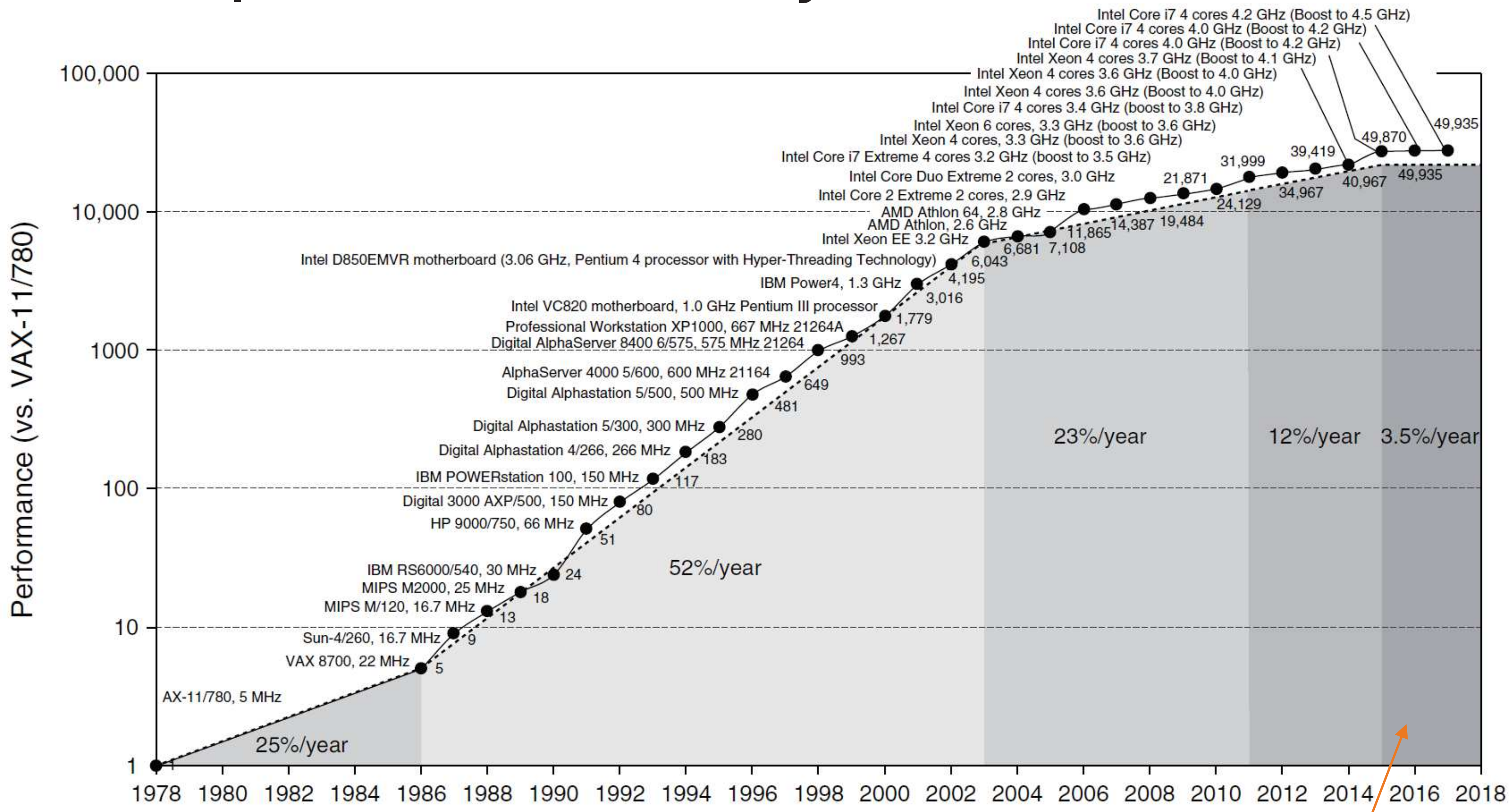


FIGURE 1.17 Growth in processor performance since the mid-1980s. This chart plots performance relative to the VAX 11/780 as measured by the SPECint benchmarks (see Section 1.11). Prior to the mid-1980s, processor performance growth was largely technology driven and averaged about 25% per year. The increase in growth to about 52% since then is attributable to more advanced architectural and organizational ideas. The higher annual performance improvement of 52% since the mid-1980s meant performance was about a factor of seven larger in 2002 than it would have been had it stayed at 25%. Since 2002, the limits of power, available instruction-level parallelism, and long memory latency have slowed uniprocessor performance recently, to about 3.5% per year.

Hard to reduce voltage (1V) and remove heat

Switch from Uniprocessors to Multiprocessors

- The power limit has forced a dramatic change in the design of microprocessors
  - ×Continuing to decrease the response time of one program running on the single processor
  - ✓ As of 2006 all desktop and server companies are shipping microprocessors with multiple processors per chip, e.g., multicore microprocessors
  - ✓ E.g., a quadcore microprocessor is a chip containing four processors (cores)

- For programmers to get significant improvement in response time
  - ➢they need to rewrite their programs to take advantage of multiple processors

Switch from Uniprocessors to Multiprocessors (Cont'd)

- It requires explicitly parallel programming
  - ➢Parallelism was often hidden before *the switch*

Before the switch

- Enjoy "instruction level parallelism"
  - ➢Hardware executes multiple instructions at once
  - ➢Hidden from the programmer

After the switch

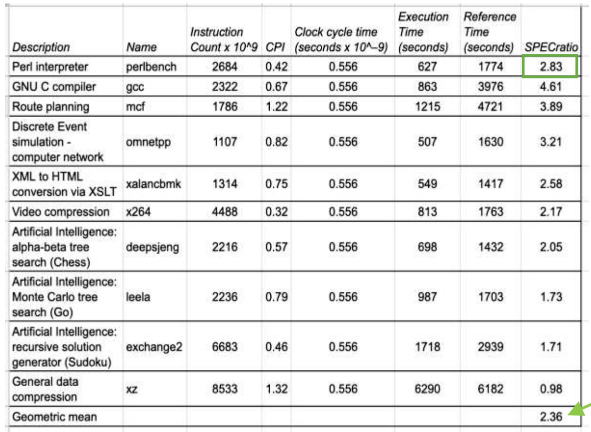
- It is hard to do
  - ➢Programming for performance (while being aware of parallel HW)
  - ➢Load balancing
  - ➢Optimizing communication and synchronization

Benchmark a Computer

- A set of programs (that a computer user usually uses) is used to evaluate the performance of a new computer
  - ➢The set of programs would form a workload
- To be representative, a set of benchmarks is often used for the performance evaluation
  - ➢Benchmarks are programs specifically chosen to measure performance
  - ➢Benchmarks are supposedly typical of actual workload
- Standard Performance Evaluation Corp (SPEC)
  - ➢is an effort funded and supported by a number of computer vendors to create standard sets of benchmarks for modern computer systems
  - ➢develops benchmarks for CPU, I/O, Web, etc.

Benchmark a Computer w/ SPEC CPU 2017

- SPEC CPU2017 is the latest version for evaluating processor performance
  - ➢Consisting of a set of 10 *integer* benchmarks (SPECspeed 2017 Integer) and 13 *floating-point* benchmarks (CFP2006)
  - ➢Elapsed time to execute a selection of programs
  - ➢Negligible I/O, so focuses on CPU performance
  - ➢Normalize relative to a *reference* machine
  - ➢Summarize as *geometric mean* of performance ratios $\sqrt[n]{\prod_{i=1}^{n} \text{Execution time ratio}_i}$



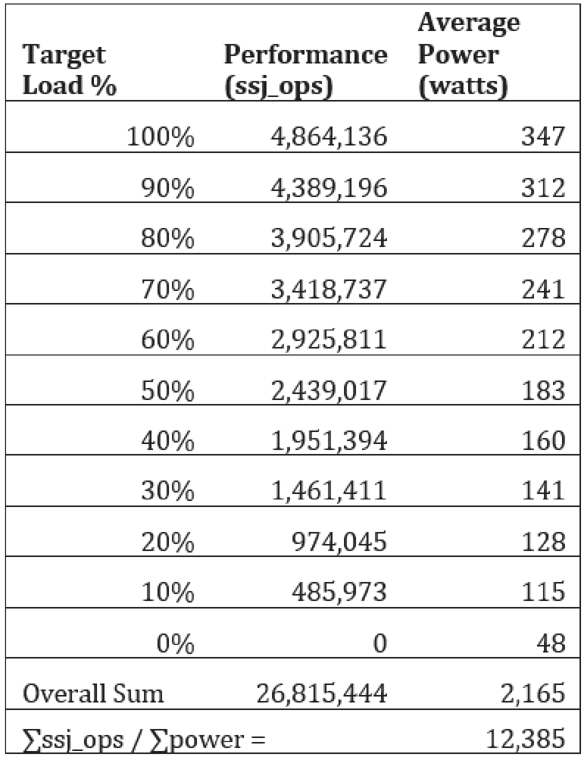
Description	*Name*	*Instruction Count x 10^9*	*CPI*	*Clock cycle time (seconds x 10^–9)*	*Execution Time (seconds)*	*Reference Time (seconds)*	*SPECratio*
Perl interpreter	perlbench	2684	0.42	0.556	627	1774	2.83
GNU C compiler	gcc	2322	0.67	0.556	863	3976	4.61
Route planning	mcf	1786	1.22	0.556	1215	4721	3.89
Discrete Event simulation - computer network	omnetpp	1107	0.82	0.556	507	1630	3.21
XML to HTML conversion via XSLT	xalancbmk	1314	0.75	0.556	549	1417	2.58
Video compression	x264	4488	0.32	0.556	813	1763	2.17
Artificial Intelligence: alpha-beta tree search (Chess)	deepsjeng	2216	0.57	0.556	698	1432	2.05
Artificial Intelligence: Monte Carlo tree search (Go)	leela	2236	0.79	0.556	987	1703	1.73
Artificial Intelligence: recursive solution generator (Sudoku)	exchange2	6683	0.46	0.556	1718	2939	1.71
General data compression	xz	8533	1.32	0.556	6290	6182	0.98
Geometric mean							2.36

FIGURE 1.18 SPECspeed 2017 Integer benchmarks running on a 1.8 GHz Intel Xeon E5-2650L. As the equation on page 35 explains, execution time is the product of the three factors in this table: instruction count in billions, clocks per instruction (CPI), and clock cycle time in nanoseconds.
SPECratio is simply the reference time, which is supplied by SPEC, divided by the measured execution time. The single number quoted as SPECspeed 2017 Integer is the geometric mean of the SPECratios. SPECspeed 2017 has multiple input files for perlbench, gcc, x264, and xz. For this figure, execution time and total clock cycles are the sum running times of these programs for all inputs.

SPEC Power Benchmark

- SEPC Power benchmark suite
  - ➢was created to measure the power and performance characteristics of server-class computer equipment
  - ➢SPECpower_ssj 2008 benchmark suite is its first release
- Performance is measured in throughput
  - ➢the units are business operations per second (ssj_ops)
  - ➢Performance: ssj_ops/sec
- To simplify the marketing of computers
  - ➢a performance number (for power) is used: "Overall ssj_ops per watt"
  - ➢Power: Watts (Joules/sec)

$$\text{Overall ssj\_ops per Watt} = \left(\sum_{i=0}^{10} \text{ssj\_ops}_i\right) \Big/ \left(\sum_{i=0}^{10} \text{power}_i\right)$$



Target Load %	Performance (ssj_ops)	Average Power (watts)
100%	4,864,136	347
90%	4,389,196	312
80%	3,905,724	278
70%	3,418,737	241
60%	2,925,811	212
50%	2,439,017	183
40%	1,951,394	160
30%	1,461,411	141
20%	974,045	128
10%	485,973	115
0%	0	48
Overall Sum	26,815,444	2,165
∑ssj_ops / ∑power =		12,385

FIGURE 1.19 SPECpower_ssj2008 running on a dual socket 2.2GHz Intel Xeon Platinum 8276L with 192GiB of DRAM and one 80GB SSD disk.

Accelerating Matrix Multiply in Python

- If the matrices are 960x960, it takes about 5 minutes to run using Python 2.7
  - ➢ Cost ~6 hours for 4096x4096 matrices (FP ops overheads)
- While it is quick to write the matrix multiply in Python, who wants to wait that long to get the answer?

- Optimizations
  - ➢ Ch. 2 convert the code to a C version
  - ➢ Ch. 3 subword parallelism via C intrinsics
  - ➢ Ch. 4 loop unrolling (multiple inst. Issue and OOO)
  - ➢ Ch. 5 cache blocking
  - ➢ Ch. 6 parallel for loops (OpenMP) exploiting multicore processor
- Combined effect: ~50,000 speedups against the original Python version

```
for i in xrange(n):
    for j in xrange(n):
        for k in xrange(n):
            C[i][j] += A[i][k] * B[k][j]
```

The above Python code runs on the machine. The n1-standard-96 server in Google Cloud Engine, which has two Intel Skylake Xeon chips, and each chip has 24 processors or cores and running Python version 3.1.

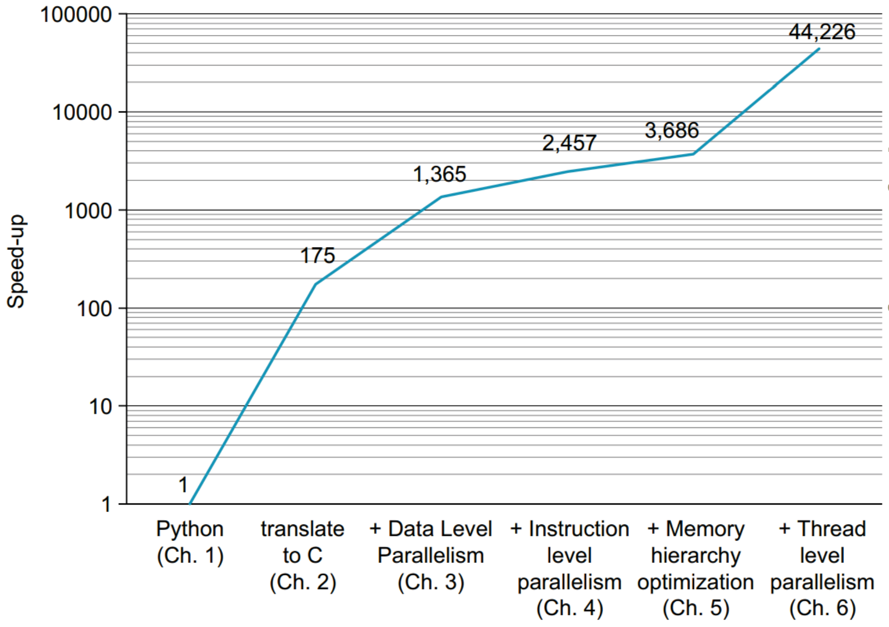


FIGURE 1.20 Optimizations of matrix multiply program in Python in the next five chapters of this book.

Please read the textbook carefully

Section 1.11 Fallacies and Pitfalls

Concluding Remarks

- Cost/performance is improving
  - Due to underlying technology development
- Hierarchical layers of abstraction
  - In both hardware and software
- Instruction set architecture
  - The hardware/software interface
- Execution time: the best performance measure
- Power is a limiting factor
  - Use parallelism to improve performance

Questions?