COMP2611: Computer Organization

Assessing and Understanding Performance

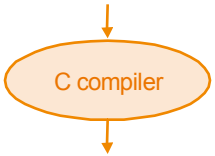
Assessing and Understanding Performance

- Learn the components and factors that influence performance
- Introduce methodologies to evaluate performance
  - Metric for evaluation – performance measure
  - Basis for comparison – benchmark
- Understand cost vs. performance implications of different architectures
- Important for computer architects, programmer and decision maker

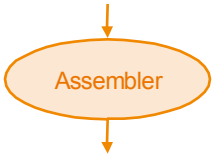
Understanding Performance

- **Algorithm**
  - Determines number of **operations** executed in the program
- **Programming language, compiler, architecture**
  - Determine number of **machine instructions** executed per operation
- **Processor and memory system**
  - Determine how fast each instruction is executed and how fast the program executed
- **I/O system (including OS)**
  - Determines how fast I/O operations are executed

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



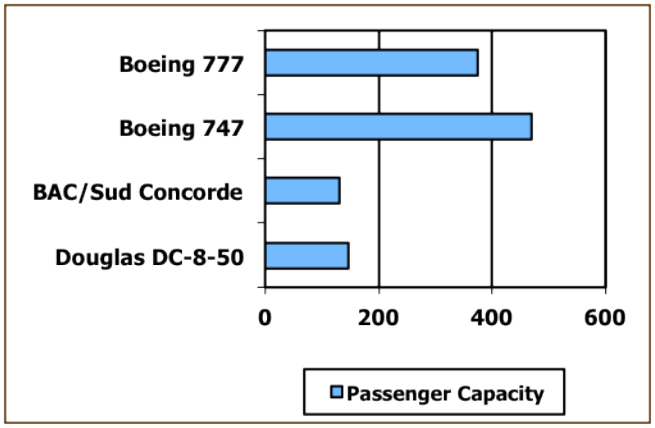

```
swap:
    muli $2, $5,4
    add  $2, $4,$2
    lw   $15, 0($2)
    lw   $16, 4($2)
    sw   $16, 0($2)
    sw   $15, 4($2)
    jr   $31
```

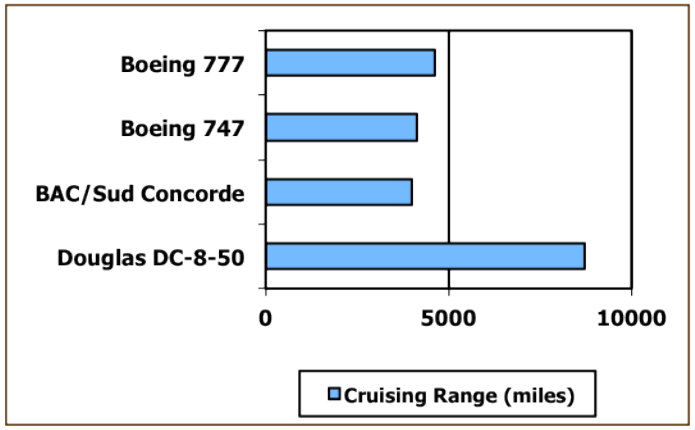


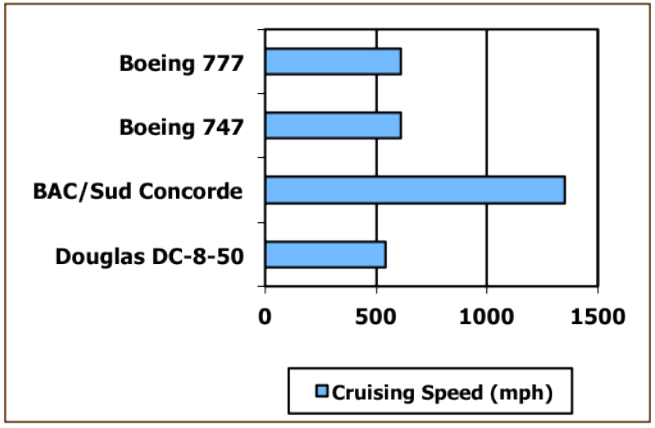

```
00000000101000010000000000011000
00000000100011100001100000100001
10001100011000100000000000000000
10001100111100100000000000000100
10101100111100100000000000000000
10101100011000100000000000000100
00000011111000000000000000001000
```

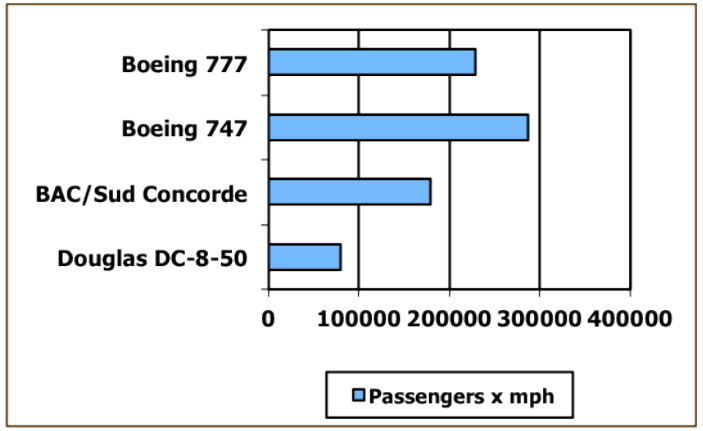
Defining Performance

- Which airplane has the best performance?









Measuring Performance

❶ **CPU time**:
- Time spent running (or executing) the program only
- Does not count I/O and time spent on running other programs
- System CPU time: CPU time spent in operating system (OS)
- Relevant to: Computer architect and programmer

① **Response time** (execution time, elapsed time or wall-clock time):
- Time between the start and completion of a task
  (i.e., CPU time + waiting time)
- Waiting time ⇒ disk access, memory access, I/O, OS overhead,…
- Relevant to: Computer end-users

❷ **Throughput**:
- Total number of jobs completed per unit time
- Relevant to: Computer administrators

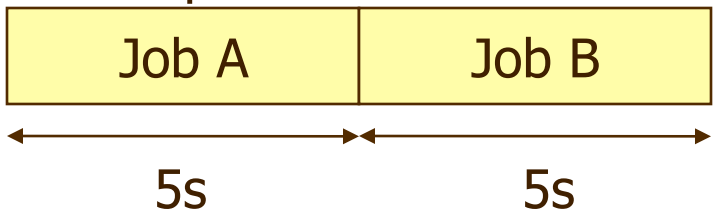
Response Time vs. Throughput

Question: is the following true? Throughput = 1/ Response time

Answer: No! it is true only if components in the system do not overlap in time

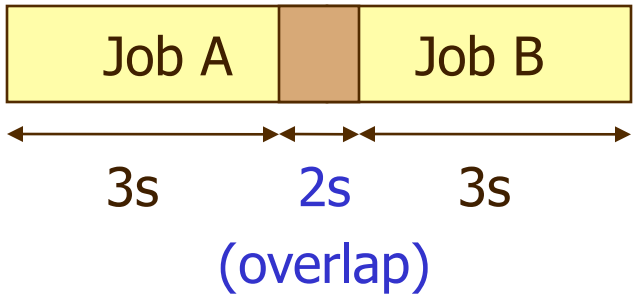
Example

- No overlap



Throughput = 2/10 = 0.2 = 1/5

- Overlap



Throughput = 2/8 = 0.25 ≠ 1/5

Execution Time & Performance

- **For some program running on a machine X:**

$$\text{performance}_X = \frac{1}{\text{execution time}_X}$$

  - Execution time: the smaller the value is better
  - Performance: the bigger the value is better

- **"Machine X is n times faster than machine Y"** means:

$$n = \frac{\text{performance}_X}{\text{performance}_Y} = \frac{\text{execution time}_Y}{\text{execution time}_X}$$

- **"Machine X is m% faster than Y"** means:

$$n = \frac{\text{performance}_X}{\text{performance}_Y} = \frac{\text{execution time}_Y}{\text{execution time}_X} = 1 + m / 100$$

Performance Comparison: Example

- Example:
- Consider two machines X and Y
  - Machine X runs a program in 10 sec
  - Machine Y runs the same program in 15 sec
  - We know that A is n times faster than B iff:

$$n = \frac{\text{performance}_X}{\text{performance}_Y} = \frac{\text{execution time}_Y}{\text{execution time}_X}$$

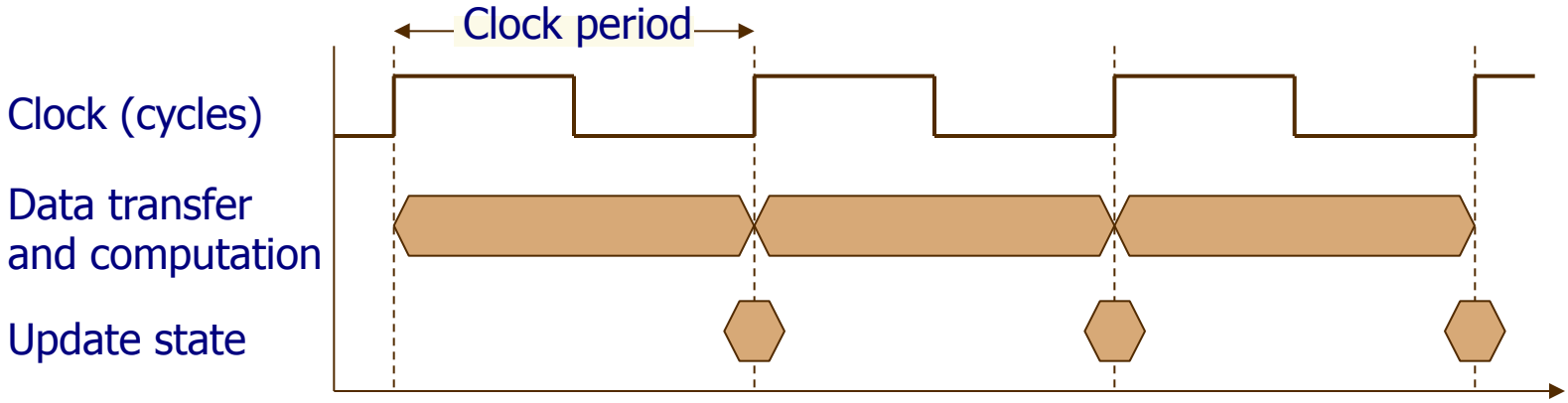
15 / 10 = 1.5 = 1 + 50/100 therefore:

  - X is 1.5 times faster than Y;
  - X is 50% faster than Y.

Clock Cycles

Almost all computers have a digital clock that runs at a constant rate and determines when basic actions take place

- Clock ⇒ to synchronize occurrence of different events in the system
- Clock cycles refer to clock "ticks"



- Clock cycle time (or clock period)
  - time between ticks (in second)
- Clock rate
  - number of clock-ticks per second (in Hz)

Example:

- A 200 MHz clock has a cycle time of $1 / (200 \times 10^6) = 5 \times 10^{-9} = 5$ ns

CPU Time vs. Clock rate

- CPU time

$$\text{CPU Time} = \#\text{CPU Clock Cycles for a program} \times \text{Clock Cycle Time}$$

or

$$\text{CPU Time} = \frac{\text{CPU Clock Cycles for a program}}{\text{Clock Rate}}$$

- Performance can be improved by:
  - Reducing the length of the clock cycle (or increasing the clock rate), or
  - Reducing the number of cycles required for a program

Cycles Per Instruction

Wrong assumption

# of CPU clock cycles in a program = # of instructions in the program

Reasons

- Execution times for different type of instructions may vary
  - Multiplication takes more time than addition
  - Floating-point operations take more time than integer ones
  - Accessing memory takes more time than accessing registers

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

Meaning

Cycles per instructions (**CPI**) = average #of clock cycle each instruction takes to execute in a program.

- Let there be n different **instruction classes** (with different CPIs)

- For a given program, suppose we know:

  CPI_i = CPI for instruction class **i**

  C_i = number of instructions of class **i**

- The formula

$$\text{CPU clock cycles} = \text{CPI} \times \text{instruction count}$$

  can be generalized to

$$\text{CPU clock cycles} = \sum_{i=1}^{n} (CPI_i \times C_i)$$

- Thus, CPI for the entire program

$$CPI = \sum_{i=1}^{n} (CPI_i \times C_i) \Big/ \sum_{i=1}^{n} C_i$$

Example: Comparing Code Segments

Description

- A particular machine has the following hardware facts:

Instruction class	CPI for this instruction class
A	1
B	2
C	3

- For a given C++ statement, a compiler designer considers two code sequences with the following instruction counts:

Code sequence	Instruction counts for instruction classes		
	A	B	C
1	2	1	2
2	4	1	1

Problem to solve

- Which code sequence executes the most instructions?
- Which is faster?
- What is the CPI for each sequence?

Example: Answer

- Instruction count: Sequence 1 : $2 + 1 + 2 = 5$ instructions
  Sequence 2 : $4 + 1 + 1 = 6$ instructions (more)
- CPU clock cycles: $\text{CPU clock cycles} = \sum_{i=1}^{n} (\text{CPI}_i \times \text{C}_i)$
  $\text{CPU clock cycles}_1 = 1 \times 2 + 2 \times 1 + 3 \times 2 = 10$
  $\text{CPU clock cycles}_2 = 1 \times 4 + 2 \times 1 + 3 \times 1 = 9$ (faster)
- CPI: $\text{CPI}_1 = \frac{10}{5} = 2$ $\text{CPI}_2 = \frac{9}{6} = 1.5$
- Remarks:
  - Sequence 2 has more instructions, but it is actually more efficient
    - Instruction count alone is not a reliable measure, should use CPI
- If these 2 codes are for different machines with different clock rates,
  - Is CPI still a good enough metric to tell which code is better?

The only complete and reliable measure of performance is CPU execution time

Other measures are unreliable because they are not consistent

- e.g., changing the instruction set from RISC to CISC to lower instruction count leads to:
  - a larger CPI, or
  - a complex hardware resulting in slower clock rate
- Either case can offset the improvement in instruction count

Other inconsistent measures

- # of clock cycles to execute a program
- # of instructions in program
- # of cycles per second
- # of cycles per instruction, or # of instructions per second

IRON LAW: Performance Equation

- A program is broken down into instructions
- Each instruction takes multiple clock cycles to execute
- Each clock cycle takes a number of seconds

$$\text{Time} = \frac{\text{seconds}}{\text{program}} = \frac{\text{\# of instructions}}{\text{a program}} * \frac{\text{\# of clock cycles}}{\text{\# of instructions}} * \frac{\text{seconds}}{\text{clock cycle}}$$

$$= \text{instruction count} * \text{CPI} * \text{clock cycle time}$$

$$= \text{instruction count} * \text{CPI} / \text{clock rate}$$

- **Instruction count** (instructions executed, not static code)
  - Mostly determined by compiler and ISA (can be measured by using profilers/simulators)
- **CPI**
  - Mostly determined by ISA and CPU organization
- **Clock rate**
  - Mostly determined by technology and CPU organization (often given)

Example: Using the Performance Equation

Description

- Two implementations of the same ISA, for a program
  - In computer A with clock cycle time 250ps, the CPI is 2.0
  - In computer B with clock cycle time 500ps, the CPI of 1.2

Question

- Which computer is faster for this program, and by how much?

Answer (assume the program has **I** instructions)

$$\text{CPU time}_A = I \times \text{CPI}_A \times \text{clock cycle time}_A$$
$$= I \times 2.0 \times 250\text{ps} = 500 \times I\text{ ps}$$

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

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

$\Rightarrow$ A is faster than B, by 20%

Common Ways to Improve Performance

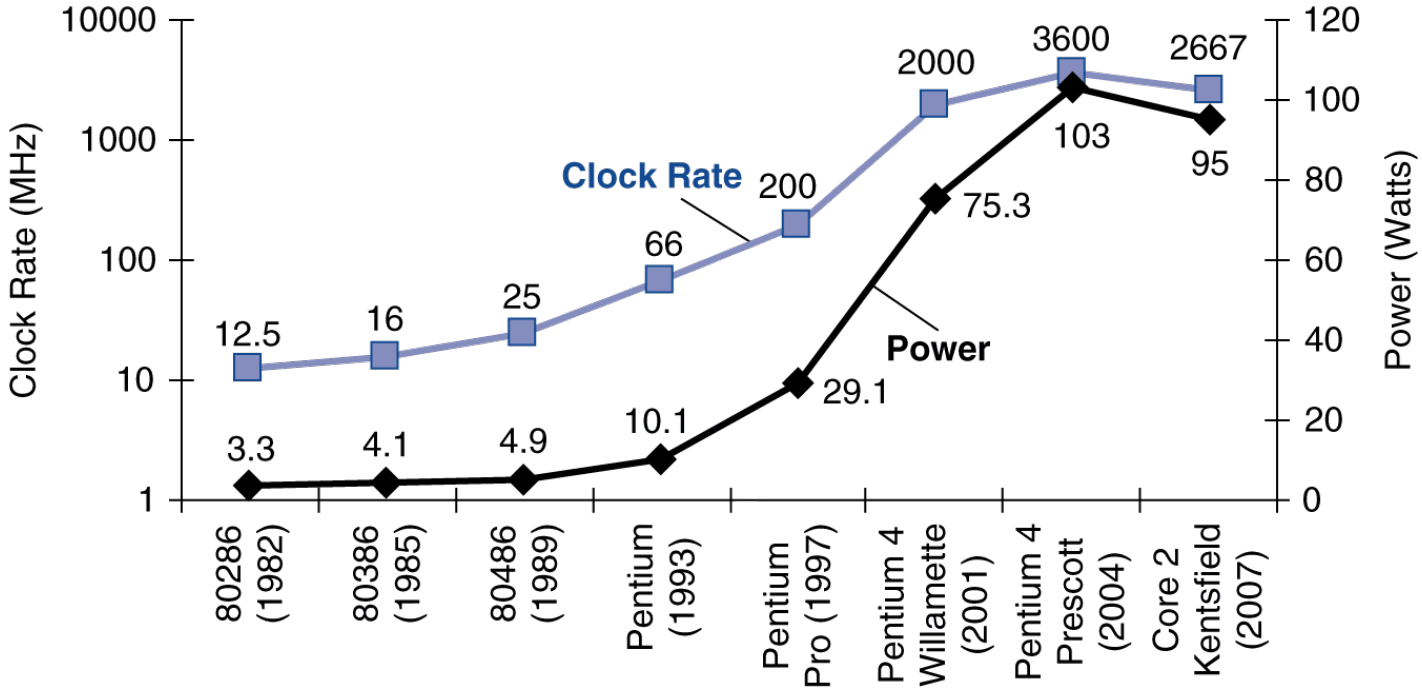
Referring to the three components in the iron law, we can either

- Shorten clock cycle time (i.e. increase clock rate)
  - e.g. run the processor at a higher clock frequency
- Reduce the CPI
  - e.g. pipelining
- Reduce the number of instructions of a program
  - e.g. use a compiler that can optimize the code generation better

Remarks

- Fine to use CPI when two conditions below are true in the comparison
  - # of instructions in the program remain unchanged, and
  - Clock rate remain unchanged
- These two conditions are often not true when comparing two ISAs

Power Trends



- In CMOS IC technology

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

×30 | 5V → 1V | ×1000

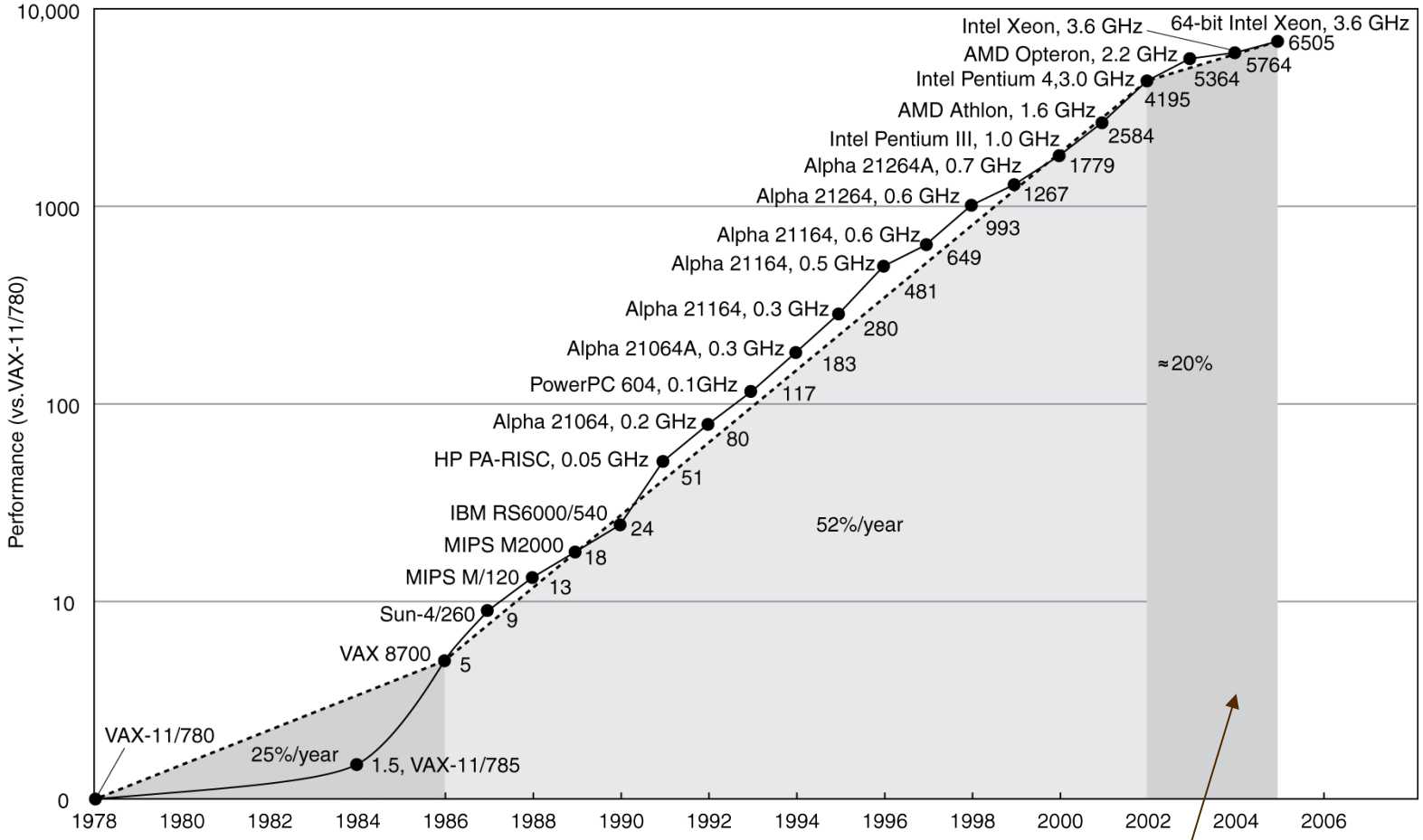
Reducing Power

- Suppose a new CPU has
  - 85% of capacitive load of old CPU
  - 15% voltage and 15% frequency reduction

$$\frac{P_{new}}{P_{old}} = \frac{C_{old} \times 0.85 \times (V_{old} \times 0.85)^2 \times F_{old} \times 0.85}{C_{old} \times V_{old}^2 \times F_{old}} = 0.85^4 = 0.52$$

- **The power wall**
  - We can't reduce voltage further
  - We can't remove more heat
- **How else can we improve performance?**

Uniprocessor Performance



Constrained by power, instruction-level parallelism, memory latency

Multiprocessors

- Multicore microprocessors
  - More than one processor per chip
- Requires explicitly parallel programming
  - Compare with instruction level parallelism
    - Hardware executes multiple instructions at once
    - Hidden from the programmer
  - Hard to do
    - Programming for performance
    - Load balancing
    - Optimizing communication and synchronization

- Today, we typically **use multi-core to improve the throughput** rather than the speed of a particular program

What should we use? Ideally, we should use real applications that we use everyday

- Programs typical of expected class of applications
  e.g., software development tools, scientific applications, graphics, etc.

In reality, benchmarks

- SPEC (Standard Performance Evaluation Cooperation) suite
  - Develops benchmarks for CPU, I/O, Power, Mail, Web, …
  - CPU performance: SPEC2006
    - Measures elapsed time of a selection of programs with negligible I/O
    - Normalize the result relative to a reference machine
    - Summarize the result as a geometric mean of performance ratios contains 12 integer benchmarks (CINT2006) and 17 floating point benchmarks (CFP2006)

$$\sqrt[n]{\prod_{i=1}^{n} \text{Execution time ratio}_i}$$

+ Save money and effort
+ Smaller than real programs, easier to standardize
– Not representative of real workload

CINT2006 for Opteron X4 2356

Name	Description	IC×10^9	CPI	Exec time	Ref time	SPECratio
perl	Interpreted string processing	2,118	0.75	637	9,777	15.3
bzip2	Block-sorting compression	2,389	0.85	817	9,650	11.8
gcc	GNU C Compiler	1,050	1.72	724	8,050	11.1
mcf	Combinatorial optimization	336	10.00	1,345	9,120	6.8
go	Go game (AI)	1,658	1.09	721	10,490	14.6
hmmer	Search gene sequence	2,783	0.80	890	9,330	10.5
sjeng	Chess game (AI)	2,176	0.96	837	12,100	14.5
libquantum	Quantum computer simulation	1,623	1.61	1,047	20,720	19.8
h264avc	Video compression	3,102	0.80	993	22,130	22.3
omnetpp	Discrete event simulation	587	2.94	690	6,250	9.1
astar	Games/path finding	1,082	1.79	773	7,020	9.1
xalancbmk	XML parsing	1,058	2.70	1,143	6,900	6.0
Geometric mean						**11.7**

Key Concepts to Remember

- Five basic components of a computer
  - **input**, **output**, **memory**, **processor** (**datapath** + **control**)
- **Hierarchical layers of abstraction in hardware and software**
  - Help cope with design complexity by hiding low level details
- **Instruction set architecture**
  - Important abstraction interfaces hardware and low-level software
- **Iron Law:**
  - **execution time** is the single-most consistent measure of performance
- **Performance** is always relative to a **specific program**
- To **compare performance** it is easier to use **benchmarks**
  - The best programs to use for **benchmarks** are **real applications**
- **Power consumption** (heat dissipation) is a limiting factor
  - Use **parallelism** and multi-core to improve performance