Chapter 1

Performance Measurements

Outline

- Performance
- The power wall
- The sea change: the switch from uniprocessor to multiprocessor
- Real stuff: benchmarking the Intel Core i7
- Fallacies and pitfalls
- Concluding remarks

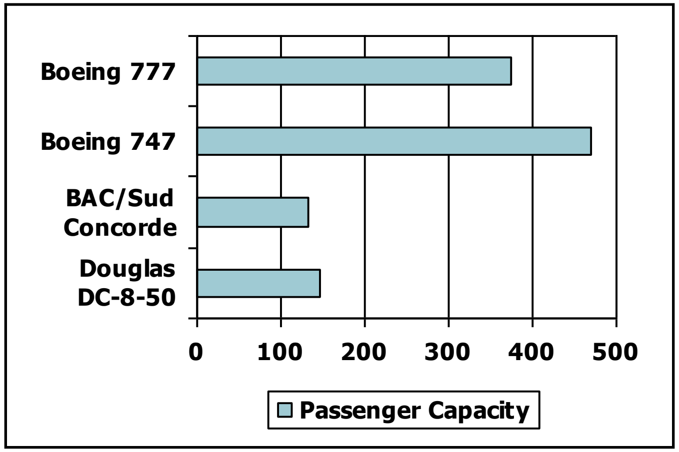
Basic Performance Metric

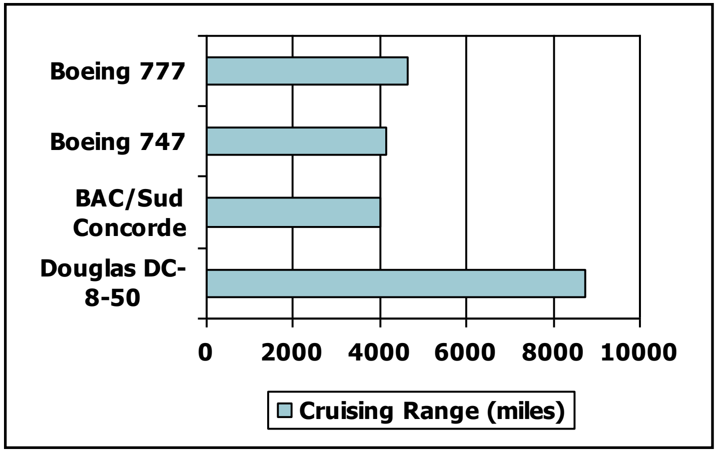
- Latency (Response Time)
  - $\sum T_i$
    — How long does it take for my job to run?
    — How long does it take to execute a job?
    — How long must I wait for the database query?
- Throughput (bandwidth)
  - $\sum W_i / \sum T_i$
    — How many jobs can the machine run at once?
    — # of lines of code per day
    — # of bits per second transmitted over a wire

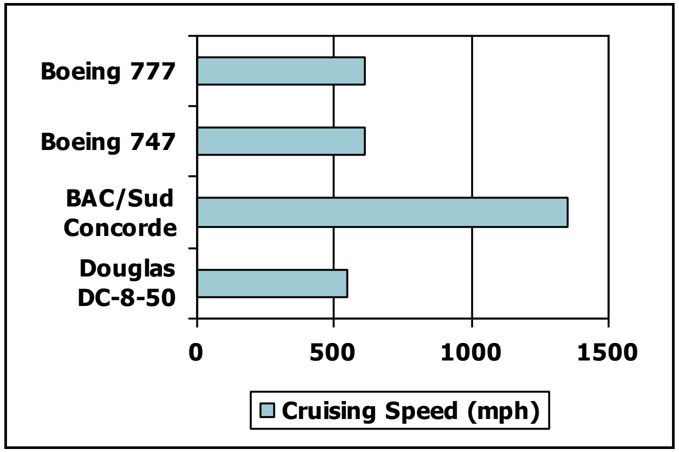
- If we upgrade a machine with a faster processor what do we increase? Latency減少，Throughput會跟著上升
- If we add an additional machine to the lab what do we increase? Latency不變，但平行度的提高可以增加Throughput

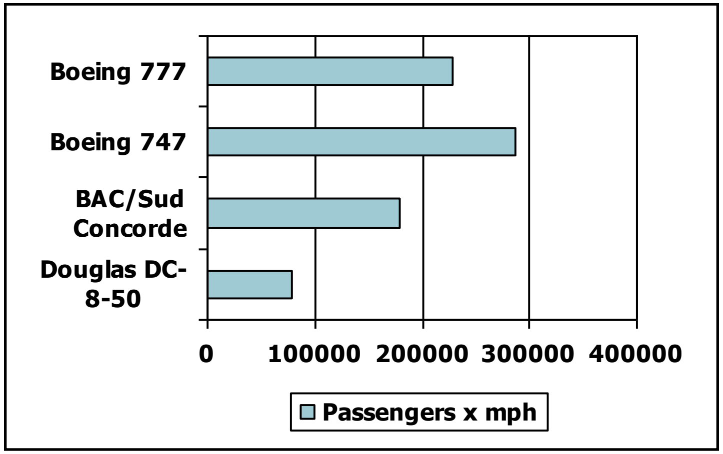
Defining Performance

- Which airplane has the best performance?









Example: Latency vs. Throughput

Plane	DC to Paris	Speed	Passengers	Throughput (pph)
	6.5 hours	610 mph	470	72.3
	3 hours	1350 mph	132	44

- Time to run the task
  - Execution time, response time, latency
- Tasks per day, hour, week, sec, ns …
  - Throughput, bandwidth

Performance

- Speed of Concorde vs. Boeing 747
  - 1350 mph vs 610 mph (2.21:1)
  - Concord is 2.2 times faster in terms of flying time
- Throughput of Boeing 747 vs. Concorde
  - 72.3 pph vs 44 pph (1.63 : 1)
  - Boeing is 1.6 times faster (better) in terms of throughput

Relative Performance

- Define 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
  - So A is 1.5 times faster than B

Measuring Execution Time

- Elapsed time
  - Total response time, including all aspects
    - Processing, I/O, OS overhead, idle time
  - Determines system performance
- CPU time
  - Time spent processing a given job
    - Discounts I/O time, other jobs' shares
  - Comprises user CPU time and system CPU time
  - Different programs are affected differently by CPU and system performance

Wall Clock: 包含OS Scheduling、I/O program, 其他程式干擾的時間

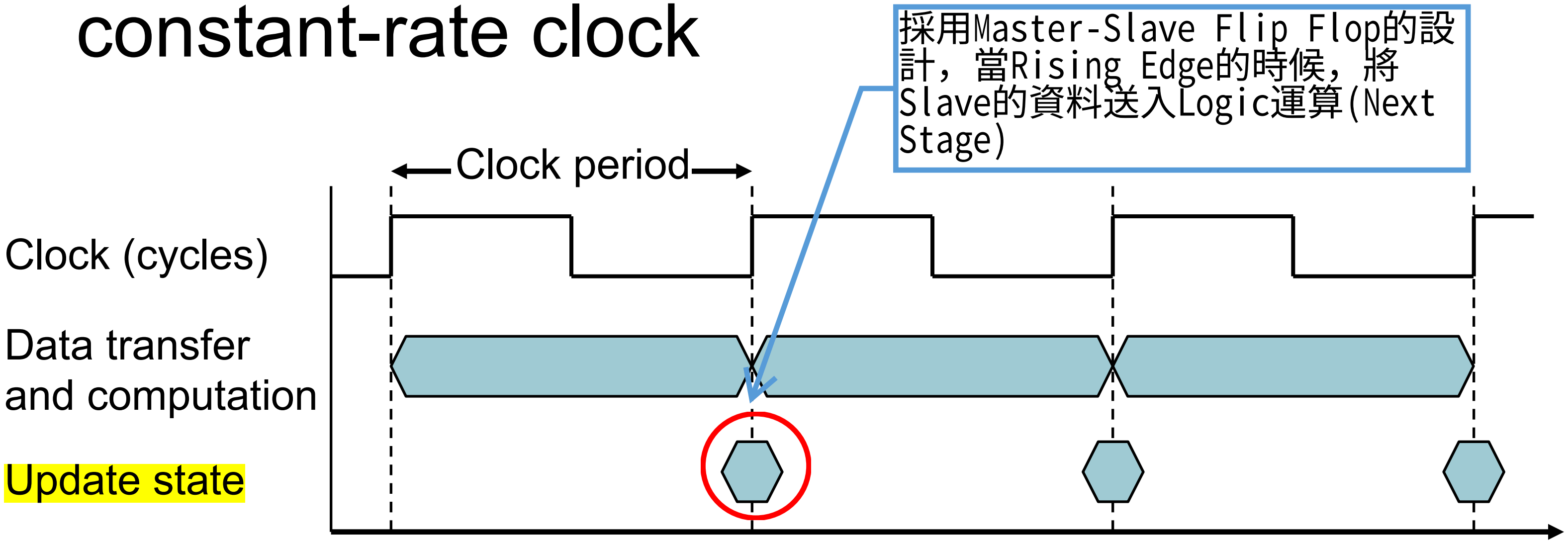
(Non-Deterministic) 通常會排除

通常看CPU Time的影響:
(1) User Time: 自己的Program
(2) System Time: OS Service

End to End time (Total time) = Elapsed Time + CPU Time

CPU Clocking

- Operation of digital hardware governed by a constant-rate clock

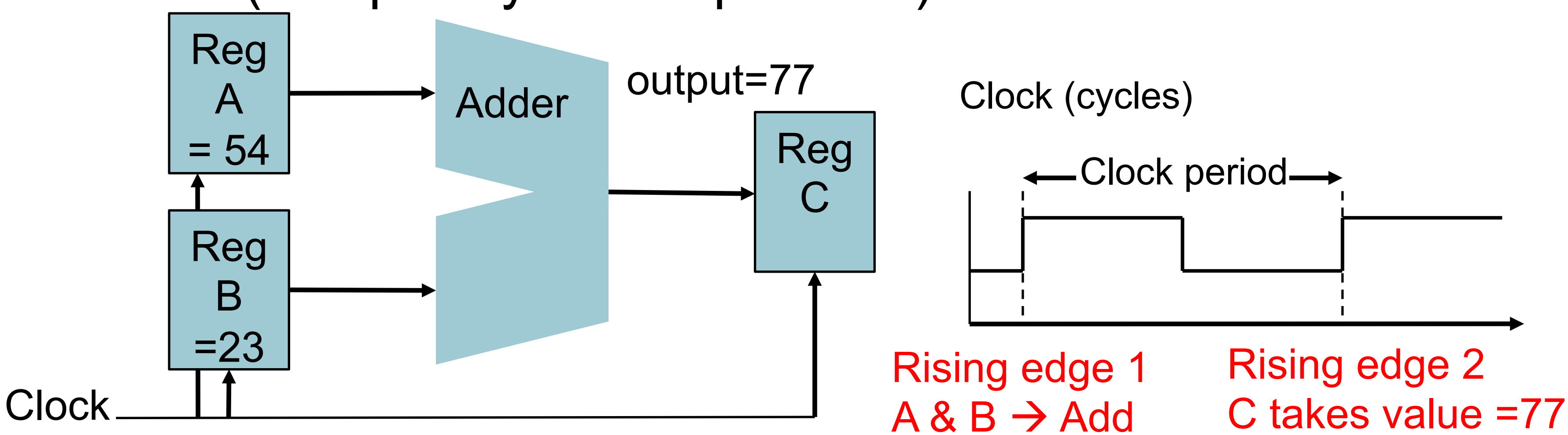


- Clock period: duration of a clock cycle
  - e.g., 250ps = 0.25ns = 250×10^{-12}s
- Clock frequency (rate): cycles per second
  - e.g., 4.0GHz = 4000MHz = 4.0×10^{9}Hz

What is a Clock?

數位邏輯設計採用Clock當作指標，可以忽略電路設計帶來的影響

- Logic signal to determine when "state" should be updated
  - Ex: when a register latches output of the adder
  - It takes time to take values (54, 23) and propagate through adder
  - Clock period = longest paths between registers (complexity of computation)



CPU Time

只考慮程式在CPU被執行所需的時間

Architecture 設計目標

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

電路設計目標

- CPU Time is the time a processor spends executing a piece of software
- Performance improved by
  - Reducing number of clock cycles
  - Increasing clock rate
  - Hardware designer must often trade off clock rate against cycle count

舉例來說:
乘法器需10ps、加法器需1ps
一般來說會取clock為10ps(選較慢的)，但是若將clock訂為1ps時，會增加存在register的時間與次數，使得執行時間大於10ps!

CPU Time Example 1

- CPU Clock freq = 1GHz (clk cycle time = 1 ns = 0.000000001 sec)
- A program takes 5,000,000 cycles to execute
- CPU Time = 5,000,000 * 1 ns = 5,000,000 nsecs = 0.005 seconds

CPU Time Example 2

- Computer A: 2GHz clock, 10s CPU time
- Designing Computer B
  - Aim for 6s CPU time
  - Can do faster clock, but causes 1.2 × clock cycles
- How fast must Computer B clock be?

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

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

$$= 10s \times 2GHz = 20 \times 10^9$$

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

CPI: Cycles Per Instruction

- CPI = Avg. Instruction Execution Time in Cycles= $\frac{CPU\ time\ in\ cycles}{\#\ of\ instructions}$

通常以此來評斷CPU的表現!

- CPI is an average of all instructions in a program or several programs

不同的implementation造成Instruction需要的Cycle數不同

- Useful in comparing two different implementations of the same architecture

Same ISA(same Instruction Count)

Ex: 排除Compiler的影響，因為較好的Compiler會有較少的Instructions，和CPU本身的架構無關!

Compiler、ISA: instruction count
CPU: cycles (CPI)
Processor: clock rate

Instruction Count and CPI

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

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

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

- Instruction Count for a program
  - Determined by program, ISA and compiler
- Average cycles per instruction
  - Determined by CPU hardware
  - Different instructions have different CPI
    - Average CPI affected by instruction mix

CPI Example

- Computer A: Cycle Time = 250ps, CPI = 2.0
- Computer B: Cycle Time = 500ps, CPI = 1.2
- Same ISA Same Instruction Count
- Which is faster, and by how much?

$$\text{CPU Time}_A = \text{Instruction Count} \times \text{CPI}_A \times \text{Cycle Time}_A$$
$$= I \times 2.0 \times 250\text{ps} = I \times 500\text{ps}$$

A is faster…

$$\text{CPU Time}_B = \text{Instruction Count} \times \text{CPI}_B \times \text{Cycle Time}_B$$
$$= I \times 1.2 \times 500\text{ps} = I \times 600\text{ps}$$

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

…by this much

Different CPI in Instruction Sets

- Different instructions take different amount of time to finish
  - Multiply vs. add
  - Cache hit and misses of load/store
  - …

CPI in More Detail

- If different instruction classes take different numbers of cycles

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

- Weighted average CPI

想成第i個Instruction佔所有Instruction的比例乘上其平均CPI

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

Relative frequency

CPI Example 1

- Assume a program has 100 instructions
  - 25 load/store (each takes 2 cycles)
  - 50 adds (each takes 1 cycle)
  - 25 square root (each takes 100 cycles)

看似improve平方根的cycle數能夠大大增進Avg CPI的表現，但在實務上卻不實際(因為平方根運算不常用)。

Average CPI= total cycles/# of instructions

=[(25*2) + (50*1) + (25*100)]/100

=(25/100)*2 + (50/100)*1 + (25/100)*100

=26.0

frequency cycles

CPI Example 2

- Alternative compiled code sequences using instructions in classes A, B, C

Instruction Count

Class	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

Performance Summary

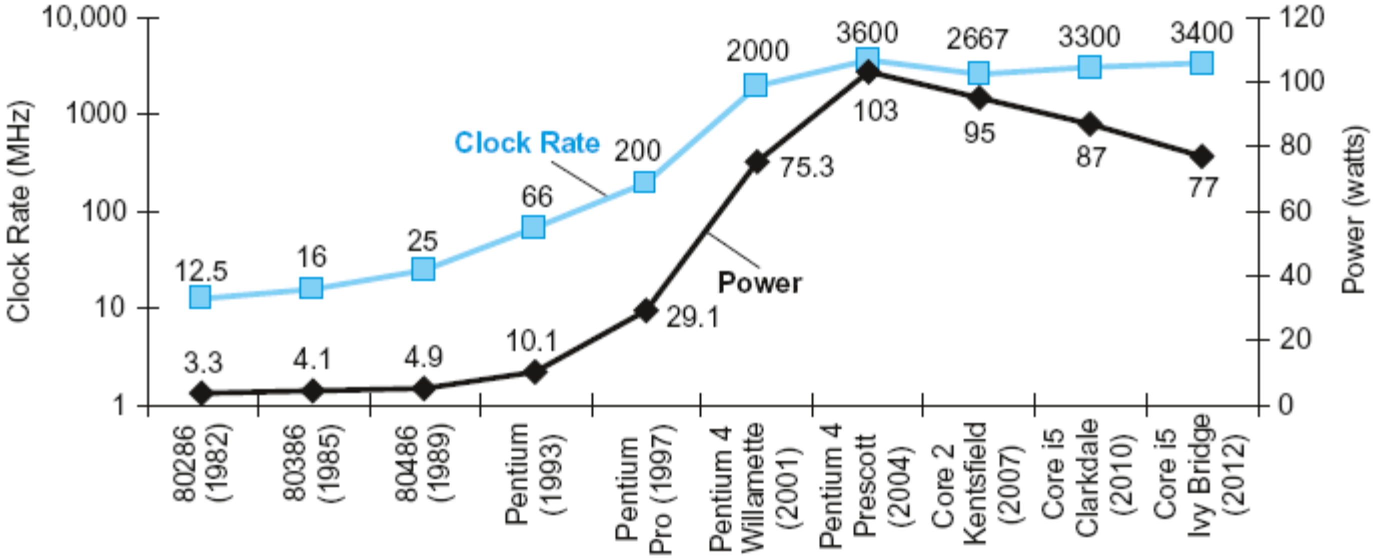
The BIG Picture

Program內的Instruction個數 CPI(CPU架構問題) Clock Period

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

- Three principle components of runtime:
  - Instruction count
  - CPI
  - Clock rate
- Performance depends on
  - Algorithm: affects IC, possibly CPI
  - Programming language: affects IC, CPI
  - Compiler: affects IC, CPI
  - Instruction set architecture: affects IC, CPI, T_c

Power Trends



- In CMOS IC technology

Voltage很難再往下降，因為Voltage要超越Threshold值

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

若拉高Frequency，則必須提高CPU散熱(Power提高)

×30

可以持續提升

5V → 1V

×1000

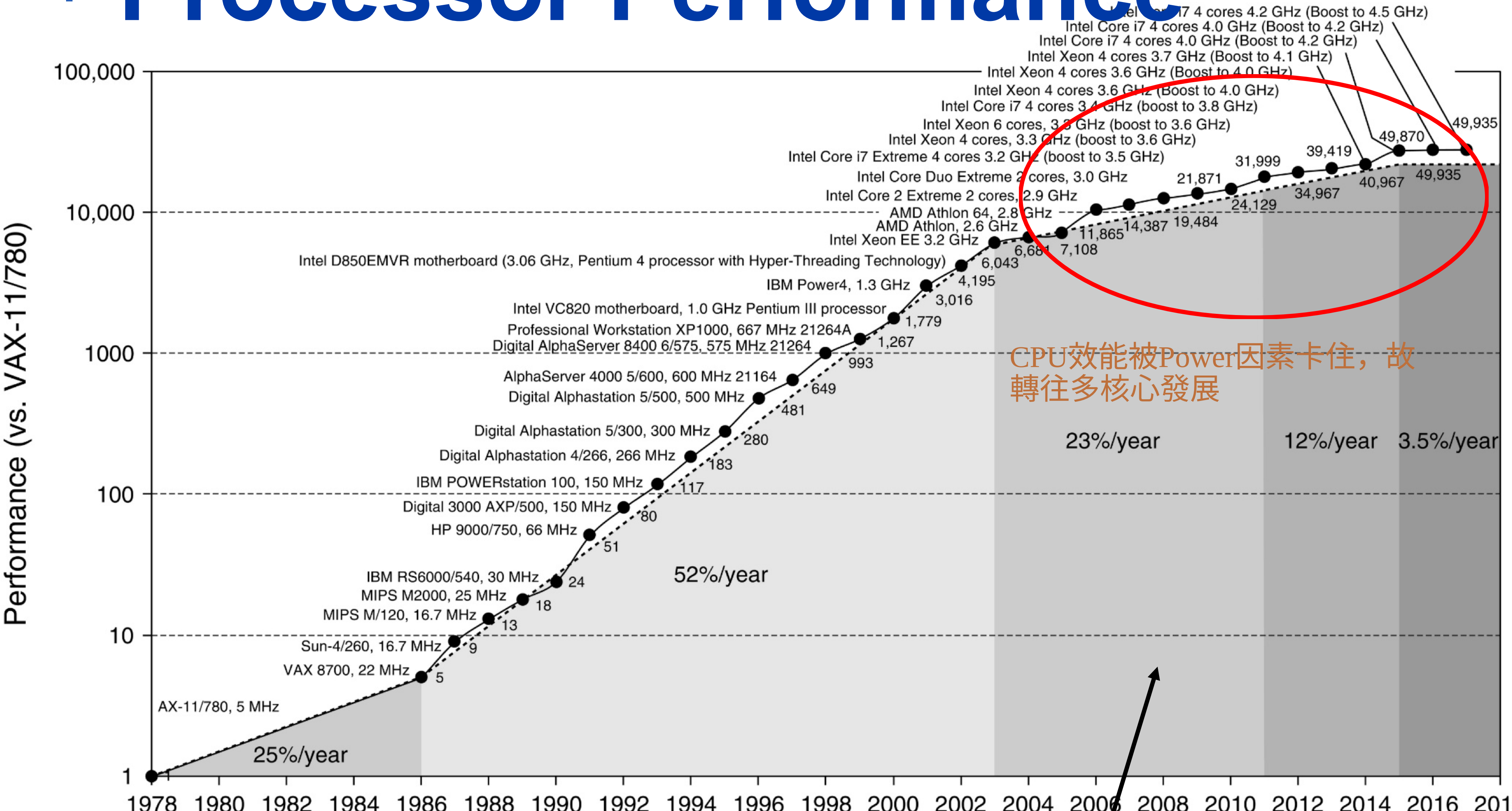
Reducing Power

- 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?

Processor 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

Comparing Performance

- Recap
- "X is n time faster than Y"

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

- It's easy to compare for **one** program
- What about multiple programs?

Comparing Multiple Programs

- Two machines with two programs

	Machine A	Machine B
Program 1	2 s	4 s
Program 2	12 s	8 s

算數平均不能用倒數互比，如果有Ratio，計算出的平均必須互為倒數

- Try to average over machine A

平均快幾倍

(program 1 + program 2)/2 = (4/2 + 8/12)/2 = 4/3

- Try to average over machine B

(program 1 + program 2)/2 = (2/4 + 12/8)/2= 1

Solution

- Use Geometric Mean

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

	Machine A (Normalized to B)	Machine B (Normalized to A)
Program 1	0.5 *A時間為B的0.5倍*	2.0
Program 2	1.5	0.666
Geometric Mean	0.866	1.155

Note: 1.155=1/0.866

由Geometric Mean可知，A比B快!

SPEC CPU Benchmark

- Programs used to measure performance
  - Supposedly typical of actual workload
- Standard Performance Evaluation Corp (SPEC)
  - Develops benchmarks for CPU, I/O, Web, …
- SPEC CPU2006
  - Elapsed time to execute a selection of programs
    - Negligible I/O, so focuses on CPU performance
  - Normalize relative to reference machine
  - Summarize as geometric mean of performance ratios
    - CINT2006 (integer) and CFP2006 (floating-point)

CINT2006 for Intel Core i7 920

SPEC Ratio = Reference Time / Execution Time

Description	Name	Instruction Count x 10^9	CPI	Clock cycle time (seconds x 10^{-9})	Execution Time (seconds)	Reference Time (seconds)	SPECratio
Interpreted string processing	perl	2252	0.60	0.376	508	9770	19.2
Block-sorting compression	bzip2	2390	0.70	0.376	629	9650	15.4
GNU C compiler	gcc	794	1.20	0.376	358	8050	22.5
Combinatorial optimization	mcf	221	2.66	0.376	221	9120	41.2
Go game (AI)	go	1274	1.10	0.376	527	10490	19.9
Search gene sequence	hmmer	2616	0.60	0.376	590	9330	15.8
Chess game (AI)	sjeng	1948	0.80	0.376	586	12100	20.7
Quantum computer simulation	libquantum	659	0.44	0.376	109	20720	190.0
Video compression	h264avc	3793	0.50	0.376	713	22130	31.0
Discrete event simulation library	omnetpp	367	2.10	0.376	290	6250	21.5
Games/path finding	astar	1250	1.00	0.376	470	7020	14.9
XML parsing	xalancbmk	1045	0.70	0.376	275	6900	25.1
Geometric mean	–	–	–	–	–	–	25.7

SPEC Power Benchmark

- Power consumption of server at different workload levels
  - Performance: ssj_ops/sec
  - Power: Watts (Joules/sec)

Performance和Power Consumption之間做Tradeoff

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

SPECpower_ssj2008 for Xeon X5650

Target Load %	Performance (ssj_ops)	Average Power (Watts)
100%	865,618	258
90%	786,688	242
80%	698,051	224
70%	607,826	204
60%	521,391	185
50%	436,757	170
40%	345,919	157
30%	262,071	146
20%	176,061	135
10%	86,784	121
0%	0	80
Overall Sum	4,787,166	1,922
Σssj_ops/Σpower =		2,490

對於Cloud端來說WorkLoad很少大於60%，故Perf/Power Ratio很難達到很高

在WorkLoad偏低時，Perf/Power Ratio較差

Things to Note

- Performance is specific to a particular program/s
  - Total **execution time** is a consistent summary of performance
- For a given architecture performance increases come from:
  - increases in clock rate (without adverse CPI affects and power limits)
  - improvements in processor organization that lower CPI
  - compiler enhancements that lower CPI and/or instruction count

Pitfall: Amdahl's Law

- Improving an aspect of a computer and expecting a proportional improvement in overall performance

用於計算能夠Improve的limit(Ex:平行化)

$$T_{improved} = \frac{T_{affected}}{\text{improvement factor}} + T_{unaffected}$$

無法Improve的部分

- Example: multiply accounts for 80s/100s
  - How much improvement in multiply performance to get 5× overall?

$$20 = \frac{80}{n} + 20$$

  - Can't be done!
- Corollary: make the common case fast

謬誤

Fallacy: Low Power at Idle

- Look back at i7 power benchmark
  - At 100% load: 258W
  - At 50% load: 170W (66%)
  - At 10% load: 121W (47%)
- Google data center
  - Mostly operates at 10% – 50% load
  - At 100% load less than 1% of the time
- Consider designing processors to make power proportional to load

Pitfall: MIPS as a Performance Metric

- MIPS: Millions of Instructions Per Second
  - Doesn't account for
    - Differences in ISAs between computers
    - Differences in complexity between instructions

$$\text{MIPS} = \frac{\text{Instruction count}}{\text{Execution time} \times 10^6}$$

$$= \frac{\text{Instruction count}}{\dfrac{\text{Instruction count} \times \text{CPI}}{\text{Clock rate}} \times 10^6} = \frac{\text{Clock rate}}{\text{CPI} \times 10^6}$$

  - CPI varies between programs on a given CPU

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

Tradeoff between Clock Period and Total Cycle Count

- One possible case

Adder=1ps, Multiplier=1.5ps,
# add instruction= 100
# mul instruction = 20

- Originally, if clock= 1.5ps
  Total cycle= # add+#mul=100+20=120
  Total time=120 cycle*1.5ps=180ps

若Clock為1ps，則multiplier需要2個Cycle才能完成

- Now, if clock=1ps
  Total cycle= # add+#mul*2=100+40=140
  Total time=140 cycle*1ps=140ps
  Total time= 100 cycle*1ps+20 cycle*1ps+20 cycle*1.1=142ps
  (+0.1ps overhead each pipeline stage)

clock+0.1ps

- Now, if clock=0.5ps
  Total cycle= # add*2+#mul*3=200+60=260
  Total time=260 cycle*0.5ps=130ps
  Total time= 100*0.5+100*0.6+20*0.5+40*0.6=144ps
  (+0.1ps overhead each pipeline stage)

真實時間和理想時間差異頗大
overhead可能造成表現比大clock cycle差