

Μετρικά Σύγκρισης Επίδοσης και Χρονοπρογράμματα  
(Benchmarking), και Άλλα Μετρικά  
+ Power

Κεφ. 1

Review Paper on power

HW2 due on Wednesday in the lab

# Αξιολόγηση και Σύγκριση Επίδοσης Υπολογιστικών Συστημάτων

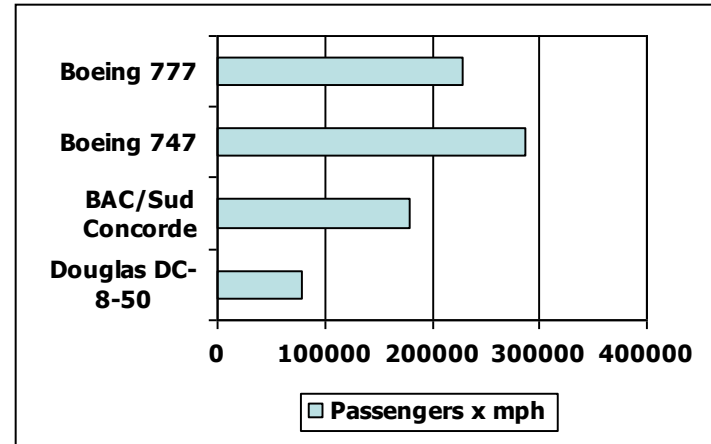
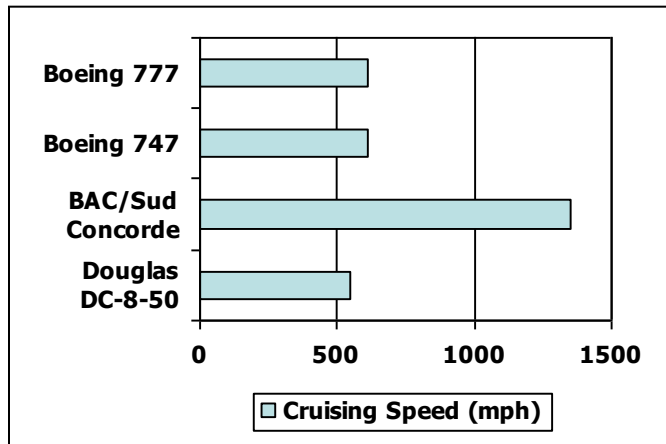
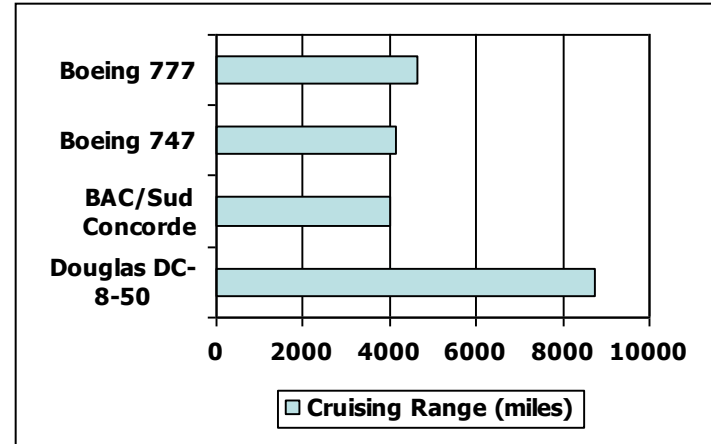
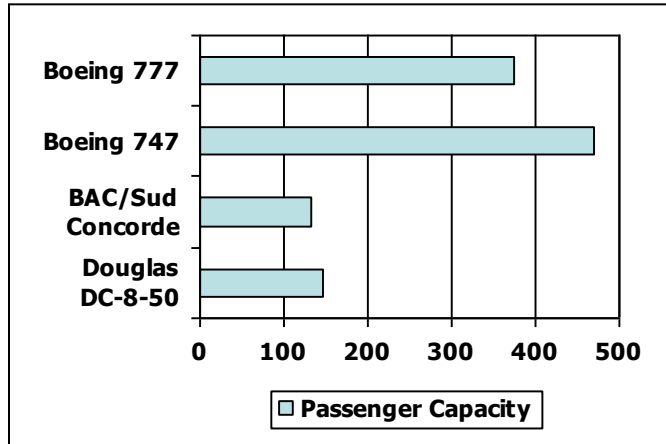
- Χρήσιμη για πολλούς λόγους:
  - Αγοραστή
    - καλή επιλογή
  - Χρήστη
    - βελτίωση συστήματος
  - Προγραμματιστή
    - πιο αποδοτικός κώδικας
  - Κατασκευαστή και Ερευνητή
    - Επίδοση/αξιολόγηση συστήματος
    - Χρησιμότητα καινούργιων ιδεών

# Προκλήσεις

- Πολύ μεγάλη ποικιλία στην αγορά υπολογιστών
- Διαφορετικοί στόχοι και προτεραιότητες
  - Κινητοί (smartphones and tablets)
  - Προσωπικοί (desktops):
  - Εξυπηρετητές (servers)
  - Ενσωματωμένοι (embedded)
  - Data Centers
  - Supercomputers
  - Functional Safety and IOT
- Τι είναι τα κατάλληλα μετρικά και προγράμματα σε κάθε περίπτωση

# Defining Performance

- Which airplane has the best performance?



# Response Time and Throughput

- Response time
  - How long it takes to do a task
- Throughput
  - Total work done per unit time
    - e.g., tasks/transactions/... per hour
- How are response time and throughput affected by
  - A faster processor?
  - More processors?
- We'll focus on response time for now...

# Πως συγκρίνουμε;

- Μέτρηση εκτέλεσης προγράμματος σε δύο υπολογιστές (real/simulated)
- Same configuration except (usually) one difference
  - processor, ram, clock, cores, disks, os, compiler, cooling etc
  - For example: sensitivity to L1 data cache size:16KB vs 64KB

# Μετρο: Χρονος Εκτέλεσης

- Επίδοση μηχανης X στην εκτελεση ενος προγραμματος:

$$\text{Επίδοση}_X = 1 / \text{Χρονος Εκτελεσης}_X$$

- Η μηχανη X εχει καλυτερη επίδοση απο την Y στην εκτελεση ενος προγραμματος

$$\text{Επίδοση}_X > \text{Επίδοση}_Y$$

$$\text{Χρονος Εκτελεσης}_X < \text{Χρονος Εκτελεσης}_Y$$

# Μετρο: Χρονος Εκτελεσης (συν)

- Η μηχανη X είναι  $v$  φορές γρηγοροτερη απο τη Y σημαινει:

$$\text{Επίδοση}_X / \text{Επίδοση}_Y = v$$

$$\text{Χρονος}_Y / \text{Χρονος}_X = v$$



# Τι είναι χρόνος εκτέλεσης;

- Χρόνος Ανταπόκρισης (response time): ο χρόνος που περασε. Αλλα περιλαμβάνει...
  - λειτουργικό συστημα
  - Ε/Ε(I/O)
  - συστήματα πολλαπλων χρηστων
- Χρόνος ΚΜΕ (CPU time): ο χρόνος που ο επεξεργαστης εργαζεται σε ενα προγραμμα
- $\text{CPU time} = \text{User time} + \text{System Time}$

# Παραδειγμα

- Στο UNIX χρονος που περασε δινεται απο την εντολη `time`:  
> *time gcc foo.c -o foo*  
*real 2m39,00*  
*user 1m30,70s*  
*sys 0m12,90s*
- CPU time 103.6s (Χρόνος ΚΜΕ, CPU time)

# Τι επηρεάζει τον χρόνο εκτέλεσης ενός προγράμματος

Χρόνος = Αριθμός Κύκλων x Χρόνος Κύκλου Μηχανής  
= Κυκλοι Μηχανης / Συχνοτητα

Χρόνος = Εντολες x CPI x Χρονος Κυκλου Μηχανης

CPU Time = I x CPI x Clock Cycle Time

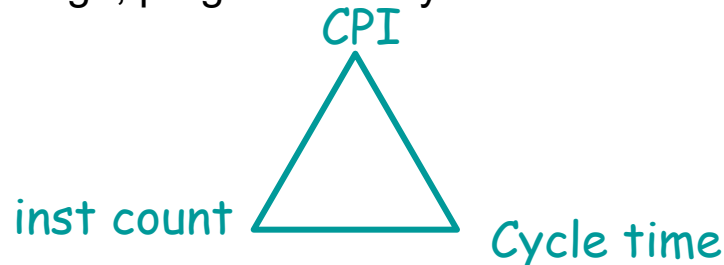
CPU Time = I x CPI / Clock Rate

- Το CPI (ή IPC=1/CPI) επιτρέπει σύγκριση δυο υλοποιήσεων με ίδια αρχιτεκτονική και ρολόι

# Τι επηρεάζει τις παραμετρους επίδοσης

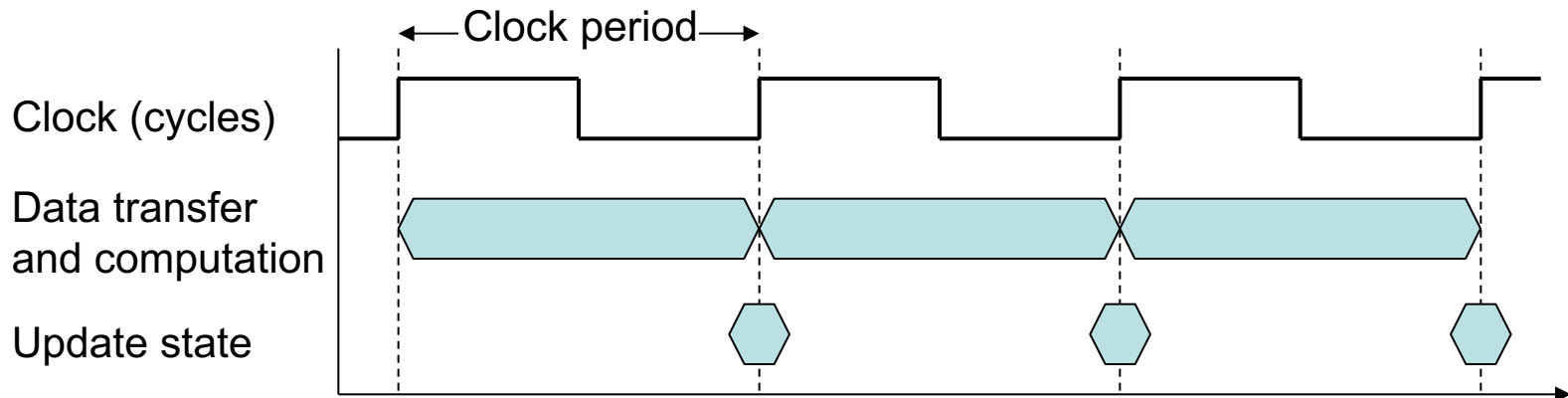
$$\text{Time} = I \times \text{CPI} \times \text{Clock Cycle Time}$$

- I: Instruction Count for a program
  - Determined by algorithm, programming language, compiler, ISA, input data
- CPI: Average cycles per instruction
  - Determined by CPU hardware and program structure
  - Typically different instructions have different CPI
    - Average CPI affected by instruction mix (influenced by parameters affecting I, program locality and control flow predictability)



# CPU Clock Cycle Time

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



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

# Συχνότητα, περίοδος, κύκλος μηχανής

- Επεξεργαστές κατασκευάζονται με ρολοι που τρέχει σε συγκεκριμένη συχνότητα (frequency cycles/s ή Hz (Hertz))
- Η περίοδος =  $1/\text{συχνότητα}$ , ονομάζεται **χρονος κυκλου μηχανης** (clock cycle time)
- Μια μηχανη με εναν επεξεργαστη που τρεχει στα 4GHz. Τι ειναι ο χρονος κυκλου μηχανης:  $1/(4\text{GHz})=0.25 \times 10^{-9}\text{s}=0.25\text{ns}$
- Ποσους κυκλους μηχανης εχει 1s; \_\_\_\_\_
- Τι καθορίζει το κύκλο μηχανής: οργάνωση, τεχνολογία υλοποίησης, power and thermal constraints

# Παράδειγμα

Ένα πρόγραμμα χρειάζεται  $10s$  για να τρέξει στον υπολογιστή A με συχνότητα  $2GHz$ .

Ένας σχεδιαστής προτείνει μια καινούργια μηχανή B, οι οποίοι θα χρειαστεί  $1.2$  περισσότερους κύκλους από την A αλλά με σημαντική αύξηση στην συχνότητα του ρολογιού.

Σε ποια συχνότητα πρέπει να τρέχει η μηχανή B για να έχει χρόνο εκτέλεσης  $6s$ ;

# Παραδειγμα (συν)

- CPU Time A = Clock Cycles A / Clock Rate A  
10s = Clock Cycles A /  $2 \times 10^9$  cycles/s  
Clock Cycles A =  $20 \times 10^9$  cycles

CPU Time B = Clock Cycles B / Clock Rate B  
6s = 1.2 Clock Cycles A / Clock Rate B  
Clock Rate B =  $1.2 \times 20 \times 10^9$  cycles / 6 s = 4 GHz



# Παραδειγμα

Δυο υλοποιησεις της ιδιας αρχιτεκτονικης και compiler εχουν για καποιο προγραμμα

	Κυκλο Ρολογιου	CPI
A	1ns	2
B	2ns	1.2

Ποια μηχανη εχει την καλυτερη επίδοση και ποσο καλυτερη ειναι;

# Παρατηρήσεις

- Χρονος είναι το μόνο αξιοπιστο μετρο συγκρισεως για επίδοση
- Όταν συγκρίνουμε εξετάζουμε όλες τις μεταβλητές που επηρεάζονται, πχ
  - Πχ όταν ο χρονος κυκλου μηχανης είναι ιδιος και έχουμε περισσότερες εντολες αλλα μικροτερο CPI μπορεί να έχουμε καλύτερη επίδοση

# Static and Dynamic Program Number of Instructions

int array_sum(int *a, int n){	// \$4: a, \$5 " n		
sum = 0;	// <b> \$? is sum, \$? is i</b>		
for(i=0;i<n;i++)	blez	\$5,\$L8	
sum+=a[i];			
return sum;	move	\$2,\$0	
}	move	\$3,\$0	
STATIC:	\$L4:		
How many lines of C	lw	\$6,0(\$4)	
code? How many	addiu	\$3,\$3,1	
assembly instructions?	addu	\$2,\$2,\$6	
	addiu	\$4,\$4,4	
	bne	\$3,\$5,\$L4	
DYNAMIC:			
if n=10. How many	j	\$31	//return
instructions get executed?			
How many assembly	\$L8:		
instructions?	move	\$2,\$0	
	j	\$31	// return

# 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

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

Relative frequency

# Example: Calculating CPI

Base Machine

Op	Freq	CPI	WCPI(i) (% Time)
ALU	50%	1	.5 (33%)
Load	20%	2	.4 (27%)
Store	10%	2	.2 (13%)
Branch	20%	2	.4 (27%)
			<hr/> 1.5

Typical Mix of  
instruction types  
in program

# Πως μετρουμε...

- Clock Cycle Time: κατασκευαστής
- Χρονος ΚΜΕ: σύστημα
- CPI: προσομοίωση, μετρητές υλικού
- I: με instrumentation, προσομοίωση, μετρητες υλικου

# Συνοψη και Συγκριση Επίδοσης

- Ένας αριθμος για την περιγραφη της Επίδοσης σε πολλα προγραμματα
- Προταθηκαν και χρησιμοποιουνται διαφοροι μεσοι οροι:
  - αριθμητικός,
  - Σταθμισμένος αριθμητικός (weighted arithmetic)
  - γεωμετρικός
  - αρμονικός

# Σύνοψη

	Υπολογιστής A	Υπολογιστής B
Προγρ. 1(s)	1	10
Προγρ. 2(s)	1000	100
Συν. Χρονος(s)	1001	110
Αριθ. Μεσος	500.5	55

Για το 1 η μηχανη A είναι 10 φορές πιο γρηγορη  
Για το 2 η μηχανη B είναι 10 φορές πιο γρηγορη  
Συνολικα η μηχανη B είναι 9.1 φορές πιο γρηγορη



# Σύνοψη

- Αριθμητικός Μέσος Όρος (Arithmetic Mean) συνοψίζει τον συνολικό χρόνο εκτέλεσης  $n$  προγραμμάτων

$$AM = \sum X_{\text{ronos}_i} / n$$

- Τι γίνεται εάν το πρόγραμμα 1 είναι πιο “σημαντικό” από το 2; Χρήση βαρών για το κάθε πρόγραμμα. Weighted Arithmetic Mean,

$$WAM = \sum w_i X_{\text{ronos}_i} / n$$

# Πραγματικά Δεδομένα: SPEC CPU

- Οργανισμός αξιολόγησης με μέλη διάφορες εταιρείες
- [www.spec.org](http://www.spec.org)
- Μέτρα:
  - Execution Time Ratio
  - Γεωμετρικός Μέσος (Geometric Mean)
  - Spec Ratio

# SPEC CPU: Γεωμετρικός Μέσος

- Για κάθε πρόγραμμα  $i$  υπολογίσε το execution ratio  $ER_i = (\text{χρονος μηχανής αναφοράς} / \text{χρονος μηχανη μετρησης})$
- Συνοψίσε τα Execution Ratio  $n$  προγραμμάτων με Γεωμετρικό Μέσο

- $$\text{SpecRatio} = \sqrt[n]{\prod_{i=1}^n ER_i}$$



Category	Name	Measures performance of
Cloud	Cloud_IaaS 2016	Cloud using NoSQL database transaction and K-Means clustering using map/reduce
Graphics and workstation performance	CPU2017	Compute-intensive integer and floating-point workloads
	SPECviewperf <sup>®</sup> 12	3D graphics in systems running OpenGL and Direct X
	SPECwpc V2.0	Workstations running professional apps under the Windows OS
	SPECapcSM for 3ds Max 2015™	3D graphics running the proprietary Autodesk 3ds Max 2015 app
	SPECapcSM for Maya <sup>®</sup> 2012	3D graphics running the proprietary Autodesk 3ds Max 2012 app
	SPECapcSM for PTC Creo 3.0	3D graphics running the proprietary PTC Creo 3.0 app
High performance computing	SPECapcSM for Siemens NX 9.0 and 10.0	3D graphics running the proprietary Siemens NX 9.0 or 10.0 app
	SPECapcSM for SolidWorks 2015	3D graphics of systems running the proprietary SolidWorks 2015 CAD/CAM app
	ACCEL	Accelerator and host CPU running parallel applications using OpenCL and OpenACC
	MPI2007	MPI-parallel, floating-point, compute-intensive programs running on clusters and SMPs
Java client/server	OMP2012	Parallel apps running OpenMP
	SPECjbb2015	Java servers
Power	SPECpower_ssj2008	Power of volume server class computers running SPECjbb2015
Solution File Server (SFS)	SFS2014	File server throughput and response time
	SPECsfs2008	File servers utilizing the NFSv3 and CIFS protocols
Virtualization	SPECvirt_sc2013	Datacenter servers used in virtualized server consolidation

**MLPerf**

**for machine learning**

# SPEC CPU 2017

	Benchmark name by SPEC generation					
	SPEC2017	SPEC2006	SPEC2000	SPEC95	SPEC92	SPEC89
GNU C compiler	←					gcc
Perl interpreter	←			perl		espresso
Route planning	←		mcf			li
General data compression	XZ		bzip2		compress	eqntott
Discrete Event simulation - computer network	←	omnetpp	vortex	go	sc	
XML to HTML conversion via XSLT	←	xalancbmk	gzip	ijpeg		
Video compression	X264	h264ref	eon	m88ksim		
Artificial Intelligence: alpha-beta tree search (Chess)	deepsjeng	sjeng	twolf			
Artificial Intelligence: Monte Carlo tree search (Go)	leela	gobmk	vortex			
Artificial Intelligence: recursive solution generator (Sudoku)	exchange2	astar	vpr			
		hammer	crafty			
		libquantum	parser			
Explosion modeling	←	bwaves				fpppp
Physics: relativity	←	cactuBSSN				tomcatv
Molecular dynamics	←	namd				doduc
Ray tracing	←	povray				nasa7
Fluid dynamics	←	lbm				spice
Weather forecasting	←	wrf			swim	matrix300
Biomedical imaging: optical tomography with finite elements	parest	gamses		apsi	hydro2d	
3D rendering and animation	blender			mgrid	su2cor	
Atmosphere modeling	cam4		wupwise	applu	wave5	
Image manipulation	imagick	milc	apply	turb3d		
Molecular dynamics	nab	zeusmp	galgel			
Computational Electromagnetics	fotonik3d	gromacs	mesa			
Regional ocean modeling	roms	leslie3d	art			
		dealll	equake			
		soplex	facerec			
		calculix	ammp			
		GemsFDTD	lucas			
		tonto	fma3d			
		sphinx3	sixtrack			

Some tips for optimization...

# Focus on the Common Case

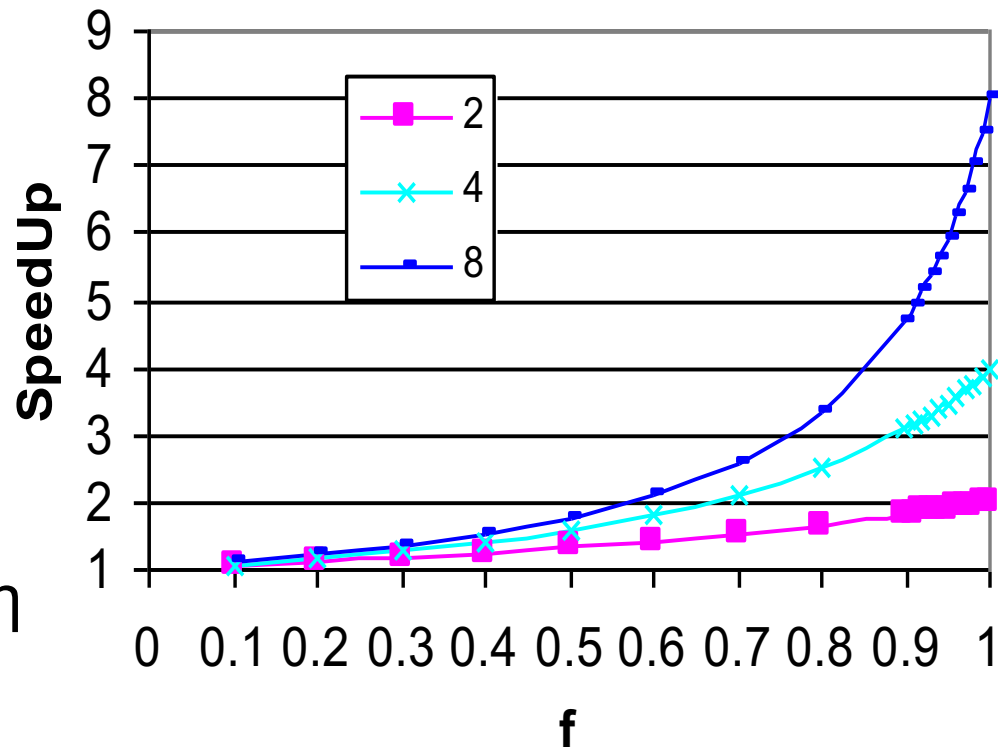
- Common sense guides computer design
  - E.g., Instruction fetch and decode unit used more frequently than multiplier, so optimize it 1st
- Frequent case is often simpler and can be done faster than the infrequent case
  - E.g., overflow is rare when adding 2 numbers, so improve performance by optimizing more common case of no overflow
  - May slow down overflow, but overall performance improved by optimizing for the normal case
- What is frequent case and how much performance improved by making case faster => [Amdahl's Law](#)



# Amdhal's Law

- $SpeedUp = 1/(1-f + f/P)$
- $f$ : fraction of program time that is improved
- $P$ : factor of improvement

- Μην βελτιστοποιήσετε ένα μηχανισμό αν δεν είναι χρήσιμος συχνά
- Κάνετε την συχνή περίπτωση γρήγορη



# Little's Law

- $Q = \lambda \cdot T$
- $Q$ : how many wait on average
- $\lambda$ : rate that tasks arrive (on average)
- $t$ : average time it takes to service a task
- E.g.  $\lambda = 5\text{tasks/s}$ ,  $t = 0.400\text{ms} \Rightarrow Q = 2$

# Measure and Report Deviation

- Does a single mean summarize well the performance of programs in benchmark suite?
- Can decide if mean a good predictor by characterizing variability of distribution using standard deviation
  - Narrow deviation means mean is more representative

# Αξιοπιστία (Dependability)

- Αναξιόπιστα συστήματα στοιχίζουν
  - Έλλειψη εμπιστοσύνης στην αγορά
  - Δυσaréσκεια με προϊόντα
- Types of Failures
  - DUE: detected uncorrected errors (lead to crash, unavailability)
  - SDC: silent data corruption (wrong output without knowing about it)
- Μετρικά
  - MTTF (mean time to failure) in hours
  - FIT (failure in time) =  $10^9\text{hrs}/\text{MTTF}$
  - MTTR (mean time to repair)
  - Availability =  $\text{MTTF}/(\text{MTTF}+\text{MTTR})$ 
    - 99.999 high target (large banks), DC IaaS 99.9-99.99 (9hrs, 1hr)
- E.g. server companies build highly dependable systems
  - On error detection retry
  - Error detection correct
  - If permanent error use spare cores, spare memory
  - ...

# Example calculating reliability

- If modules have *exponentially distributed lifetimes* (age of module does not affect probability of failure) then overall failure rate is the sum of failure rates of the modules
  - Πιθανότητα λάθους ανά πάσα στιγμή ανεξάρτητη του χρόνου
  - Total FIT =  $\sum FIT_i$
- Calculate FIT and MTTF for 10 disks (1M hour MTTF per disk), 1 disk controller (0.5M hour MTTF), and 1 power supply (0.2M hour MTTF):

*FailureRate* =

*MTTF* =

# Example calculating reliability

- If modules have *exponentially distributed lifetimes* (age of module does not affect probability of failure), overall failure rate is the sum of failure rates of the modules
- Calculate FIT and MTTF for 10 disks (1M hour MTTF per disk), 1 disk controller (0.5M hour MTTF), and 1 power supply (0.2M hour MTTF):

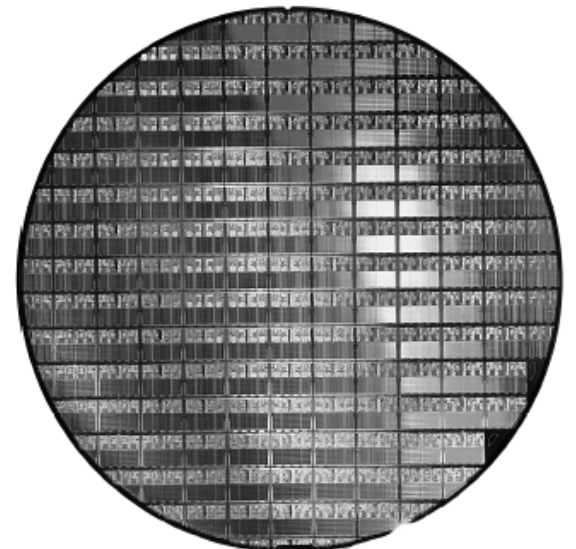
$$\begin{aligned} \text{FailureRate} &= 10 \times (1/1,000,000) + 1/500,000 + 1/200,000 \\ &= (10 + 2 + 5)/1,000,000 \\ &= 17/1,000,000 \implies 17 \text{ failures } \_ \text{in } \_ 1e6 \text{ hours} \\ &= 17,000 \text{ FIT} \end{aligned}$$

$$\begin{aligned} \text{MTTF} &= 1,000,000,000 / 17,000 \\ &\approx 59,000 \text{ hours} \end{aligned}$$

$$59000 / (24 \times 365) \approx 7 \text{ years}$$

# Κόστος Επεξεργαστή

- Το κόστος ενός επεξεργαστή επηρεάζεται από το yield
  - Το ποσοστό των chips που κατασκευάζονται και δεν έχουν κατασκευαστικά λάθη
  - Επηρεάζεται από το μέγεθος του chip και του wafer
  - Δεστε βιβλίο/HW



# Trade-off:

## Throughput vs Response Time

- Some services/applications have tight QoS requirements
  - Response time 99.9% of queries within 300ms
  - 90<sup>th</sup>: the maximum latency of 90% of the queries when sorted in ascending order
  - 99<sup>th</sup>: the maximum latency of 99% of the queries when sorted in ascending order
- Increasing throughput on a machine may use idle resources but contention on shared resources hurt response time
- Challenge: configure system for QoS that maximizes efficiency



# Σύνοψη

- Computer business is all about metrics and measurements
- What is best: faster, more battery life, better graphics, more reliable, available, cheap, easy to fix
- Metrics.... Performance, Throughput, power, energy, temperature, Availability, MTTF, MTTR, IPC, Cycle Time, Area, TCO, Mispredicts, Misses, Stalls
- Means.... Average, Geometric Mean, Harmonic Mean)
- Probability Distributions... exponential, Weibull, lognormal
- You get the idea 😊: Need to understand what they mean and how to measure them
- Validation key : understand what you measure and report is correct

# Σύνοψη

- Πολύ μεγάλη ποικιλία στην αγορά υπολογιστών
- Διαφορετικοί στόχοι και προτεραιότητες
  - Κινητοί (smartphones and tablets)
  - Προσωπικοί (desktops):
  - Εξυπηρετητές (servers)
  - Ενσωματωμένοι (embedded)
  - Data Centers
  - Supercomputers
  - Functional Safety and IOT
- Τι είναι τα κατάλληλα μετρικά και προγράμματα σε κάθε περίπτωση

# Τί συγκρίνουμε;

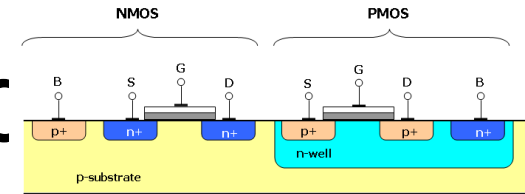
- Μέτρο Σύγκρισης
  - Χρόνος εκτέλεσης για ίδια δουλειά - high performance, desktop
  - Διεκπαιρωτική ικανότητα - πόσες δουλειές ολοκληρώνονται ανα μονάδα χρόνου (throughput, bandwidth) - servers
  - ενέργεια  $\times$  latency, ενέργεια  $\times$  latency<sup>2</sup> – high performance
  - Power – most platforms
  - Θερμοκρασία (peak temperature) – most platforms
  - Battery Life - mobile
  - Αξιοπιστία – πόσο συχνά λάθος, είδος λάθους



# Power and Energy

- Power one of the main design constraint nowadays
- Most compute platforms have fixed power envelopes
  - Thermal and power delivery reasons
- **POWER = ENERGY / TIME**
- Units: Watts (W)  $1W = 1\text{Joule}/1\text{Second}$
- 1 Joule:
  - The energy required to lift a medium-size tomato (100 g) 1 meter vertically from the surface of the Earth,
  - The typical energy released as heat by a person at rest every 1/60 second
- Processors: work to move current across the chip
- Can burn same amount of energy fast or slow (higher or lower power)
- Higher power means higher temperature and ability to deliver required power
- Problem: Get power in, get power out from processors and without burning them (or needing exotic/expensive cooling)

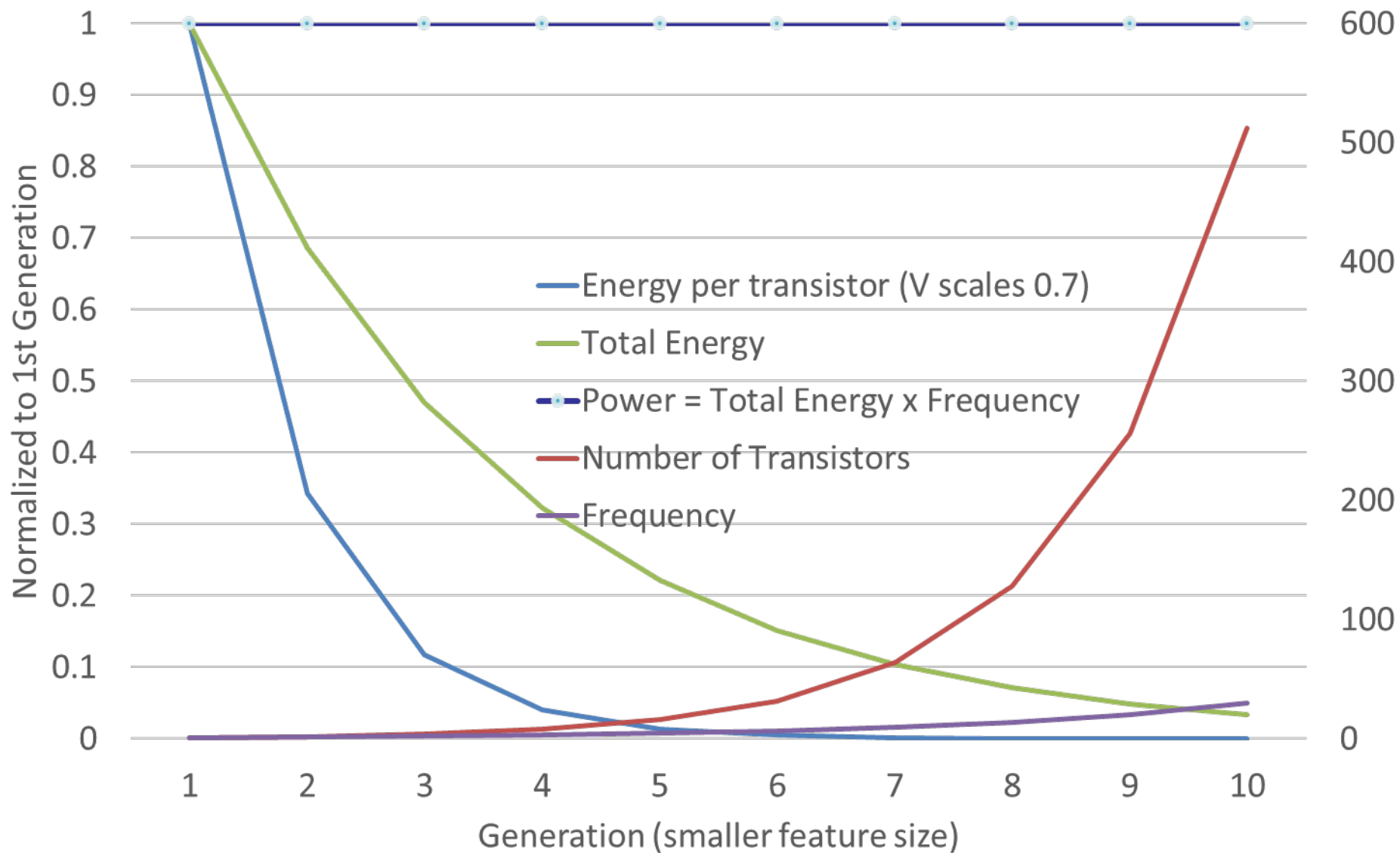
# Dynamic Energy and



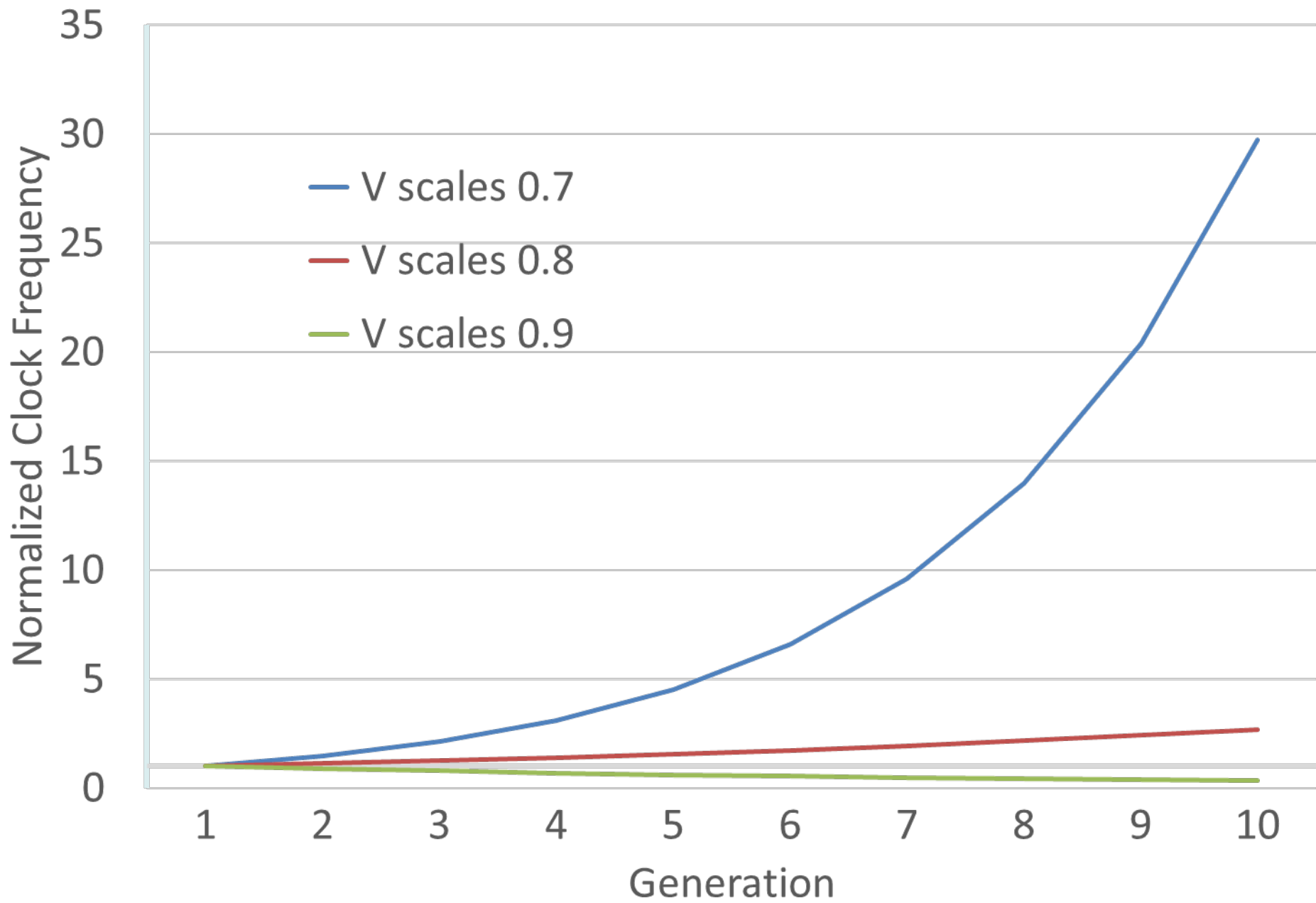
- CMOS Transistors Dynamic energy
  - Transistor switch from 0  $\rightarrow$  1 or 1  $\rightarrow$  0
  - Energy =  $\frac{1}{2} \times \text{Capacitive load} \times \text{Voltage}^2$  (  $\frac{1}{2} C \times V^2$  )
  - Capacitive load: depends on number of transistors switching (not all switch!) =  $\alpha$  Chip Capacitance
    - $\alpha$ : activity factor
  
- Dynamic power
  - Power = Energy/Time (rate of energy consumption)
  - $\frac{1}{2} \times \text{Capacitive load} \times \text{Voltage}^2 / \text{Clock Period}$
  - $\frac{1}{2} \times \text{Capacitive load} \times \text{Voltage}^2 \times \text{Frequency}$
  
- Reducing clock rate reduces power, not energy (the same works gets done but slower)
  - Not for all technologies (some DRAM logic)

# Technology Scaling (simplified model)

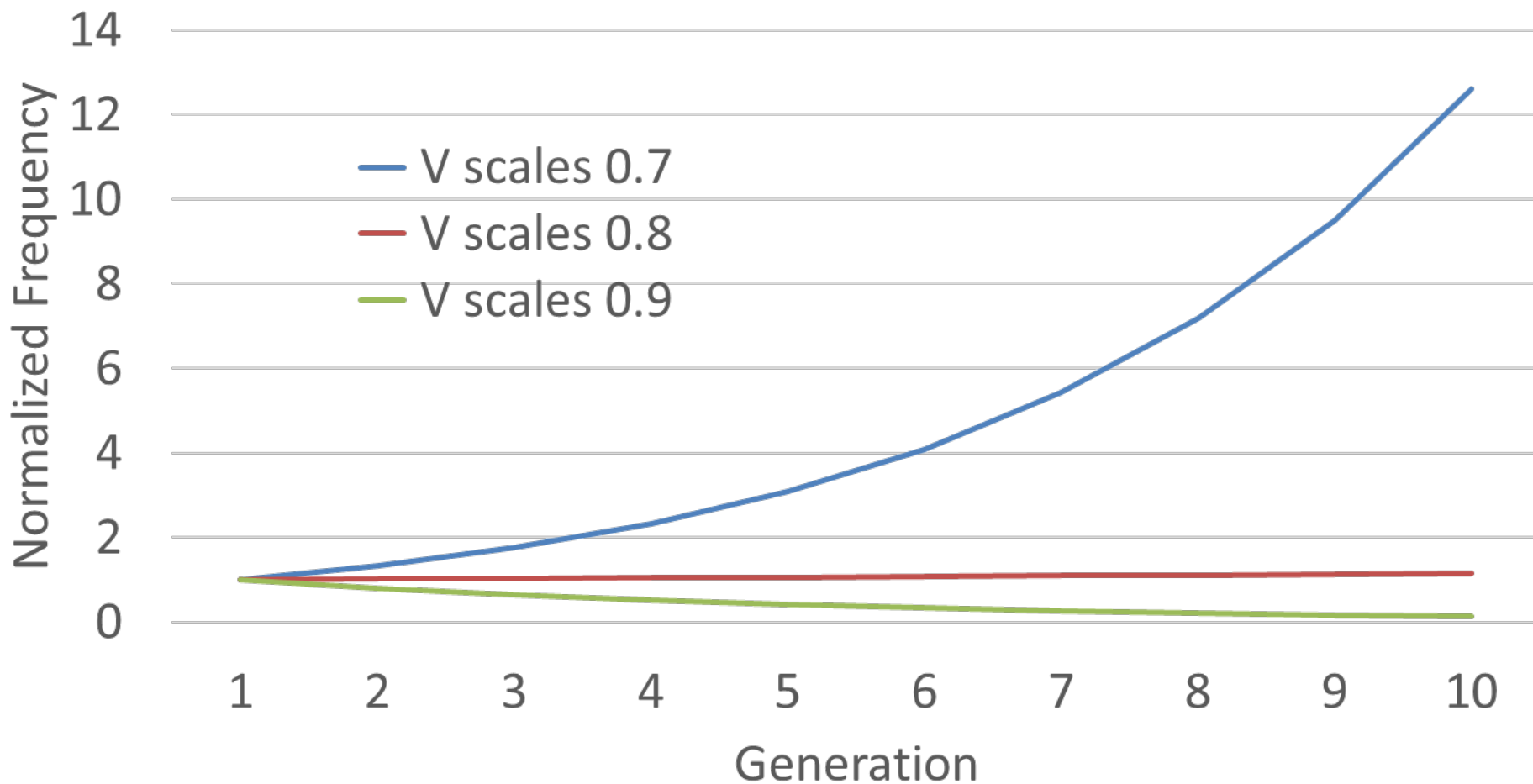
- Number of Active Devices x2
- Capacitance x0.7
- Voltage x0.7
  
- Device Energy x0.35
  - $0.7 \times (0.7)^2 = 0.35$
- Chip Energy x0.7
  - $2 \times 0.7 \times (0.7)^2 = 0.7$
  - Energy Decreases
  - For same power room options:
    - increase Frequency by  $1/0.7=1.4$
    - Include more transistors (functionality)
  
- Unfortunately V stop scaling by 0.7x. Implications?





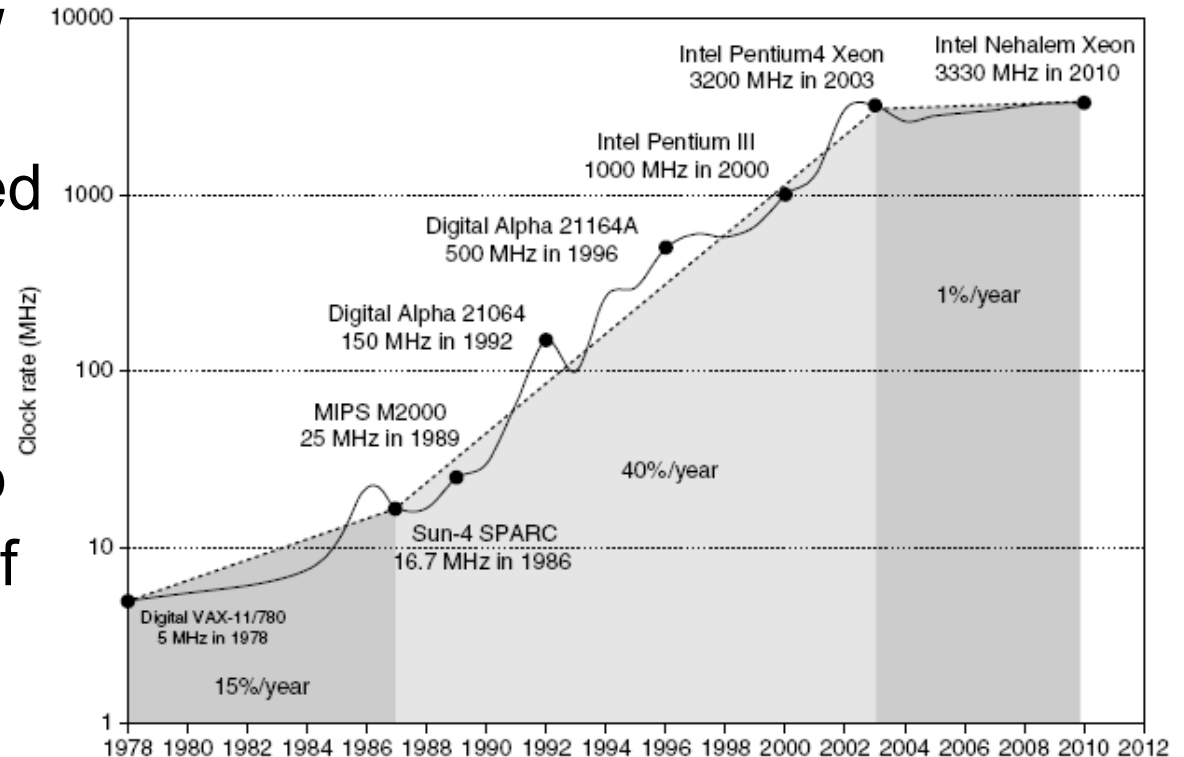


# Die size? Increase by 20 per generation



# Power Wall

- Intel 80386 consumed ~ 2 W
- 3.3 GHz Intel Core i7 consumed 130 W
- Heat must be dissipated from 1.5 x 1.5 cm chip
- This is the limit of what can be cooled by air



# Static Power

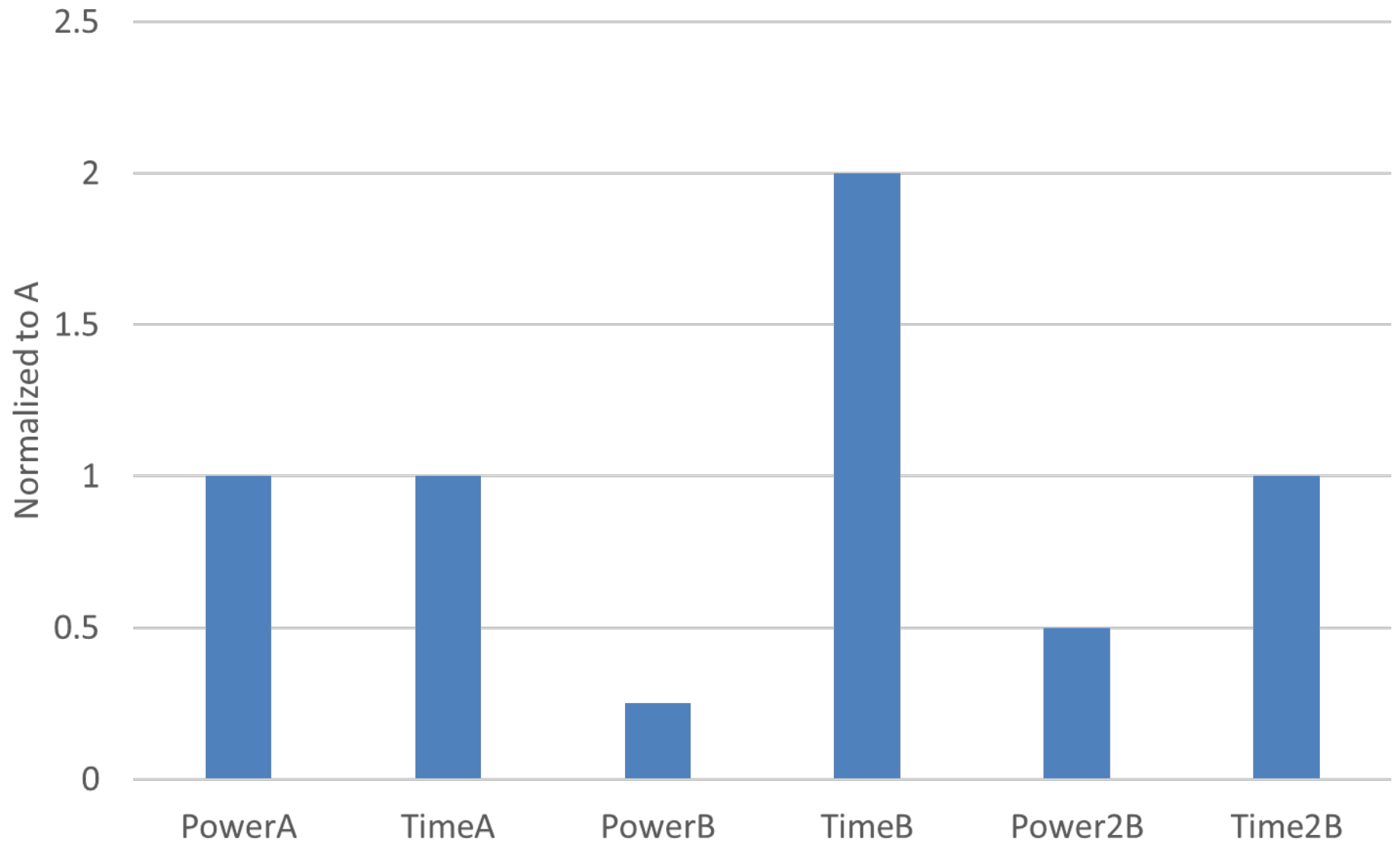
- Power = Dynamic + Static
  - 20-30% Static
- Static is losses due to imperfections
  - also called leakage
- Static power consumption
  - $\text{Current}_{\text{static}} \times \text{Voltage}$
  - Scales with number of transistors
- How to reduce Static Power
  - Higher supply Voltage ☹️ - breaks energy scaling
  - Power gating (no current for inactive parts)

# Reducing Power

- Techniques for reducing power:
  - Dynamic Voltage-Frequency Scaling (DVFS)
  - Low power state for DRAM, disks
  - Turning off cores (power gating)
  - Clock Gating
  - Do nothing well
  - Better transistors
    - Choose between power hungry with less area transistors over power efficient but larger area transistors

# Reducing Power

- **Parallelism:** get same performance with less power.
- Provided program is parallelizable
- Assume  $F$  proportional to  $V$  (simplified example)
  1.  $P = C V^2 F$
  2.  $P = 2 (C (V/2)^2 F/2) = C V^2 F / 4$ 
    - use twice real estate
- What is best choice 1 or 2?
- Answer of processor industry 1.x



# Πως μετρούμε Power, Energy, Temperature

- Πραγματικές Μηχανές
  - Μετρητές υλικού παρέχουν δυνατότητα ρύθμισης/παρατήρησης energy, voltage, frequency και της θερμοκρασίας ενός επεξεργαστή (DRAM)
- Υπό μελέτη-κατασκευή
  - με εργαλεία προσομοίωσης (wattch, cacti, hotspot, atmi, mcspat, cad tools)



# Μετρικά Απόδοσης Ισχύος (Power Efficiency Metrics)

- Energy – more emphasis on energy ignores performance
  - If you voltage scale then low energy BUT very slow!
  - Energy useful metric if a system does not use voltage scaling
- Energy.Delay – considers performance
  - Ίδιο βάρος σε ενέργεια και χρόνο
- Energy.Delay<sup>2</sup> - more emphasis on performance
  - Περισσότερη έμφαση στην επίδοση
  - If you voltage scale you will pay square on performance
- Energy/Instruction (energy efficiency)

- Παράδειγμα για το ίδιο έργο A: 1W, 1s, B: 2W, 0.75s

	A	B
• E(J)	1	1.5
• ED(Js)	1	1.125
• ED <sup>2</sup> (Js <sup>2</sup> )	1	0.85

# Turbo Mode

- TDP for a CPU
- Nominal frequency is below maximum possible frequency
- if there is headroom increase frequency  
else decrease frequency
- Modern systems multiple domains
  - Distributed power budget depend on different domains needs

# Performance(Clock Frequency)

- What happens to the execution time of a program when we lower/increase clock frequency of a processor (DVFS)
  - Execution Time = I . CPI . Cycle Time
- ***Memory operates at a different clock rate***
  - ***Has its own clock (memory controller)***

# Performance(Clock Frequency)

- What happens to the execution time of a program when we lower/increase clock frequency of a processor
  - Execution Time = I . CPI . Cycle Time
- Memory operates at a different clock rate
  - Has its own clock (memory controller)
- $T_{ref} = T_{CPU\_REF} + T_{MEM\_REF}$
- $T_{CPU} = T_{CPU\_REF} * F_{new}/F_{ref}$
- $T_{MEM} = T_{MEM\_REF}$