POLITECNICO MILANO 1863 | DIPARTIMENTO DI ELETTRONICA INFORMAZIONE E BIOINGEGNERIA

Advanced Computer Architectures

Performance and Cost

A.Y. 2024/2025 | Christian Pilato (christian.pilato@polimi.it)

POLITECNICO MILANO 1863 | DIPARTIMENTO DI ELETTRONICA INFORMAZIONE E BIOINGEGNERIA

Outline

- Measures to evaluate performance
- Quantifying the design process
- Benchmarking
- Energy/Power
- Cost

Nowadays…

- Tegra 2
  - Dual-Core ARM Cortex-A9
  - ULP GeForce, 8 cores
- A6
  - Dual-Core based on ARMv7
  - Triple-core PowerVR SGX 543MP3 GPU
- Tegra 3
  - Quad-Core
  - ULP GeForce, 12 cores

ASUS Eee Pad Slider Tablet

Motorola Photon 4G

iPhone 5

HTC One X

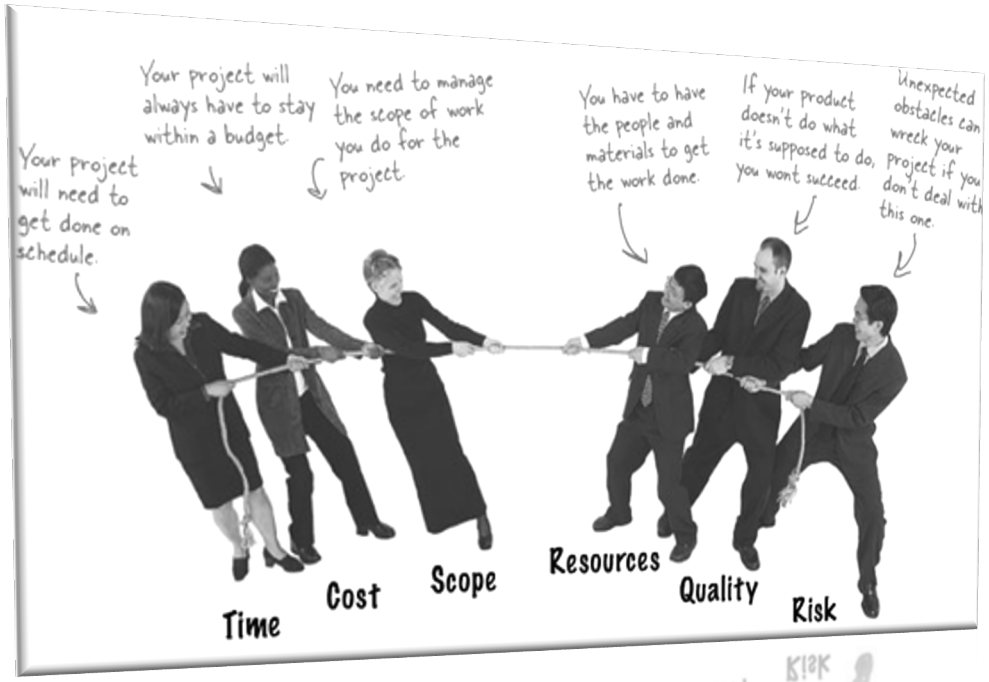
Different classes of computers… Different needs

- Personal Mobile Device (PMD)
  - Emphasis on energy efficiency and real-time
- Desktop Computing
  - Emphasis on price-performance
- Servers
  - Emphasis on availability, scalability, throughput
- Clusters / Warehouse Scale Computers - Used for “SaaS”
  - Emphasis on availability and price-performance
  - Sub-class: Supercomputers, emphasis: floating-point performance and fast internal networks
- Embedded Computers
  - Emphasis: price

Issues as new opportunities

Programming has become very difficult

Impossible to balance all constraints manually



Issues as new opportunities

Programming has become very difficult

Impossible to balance all constraints manually

- More computational horse-power than ever before
  - Cores are free

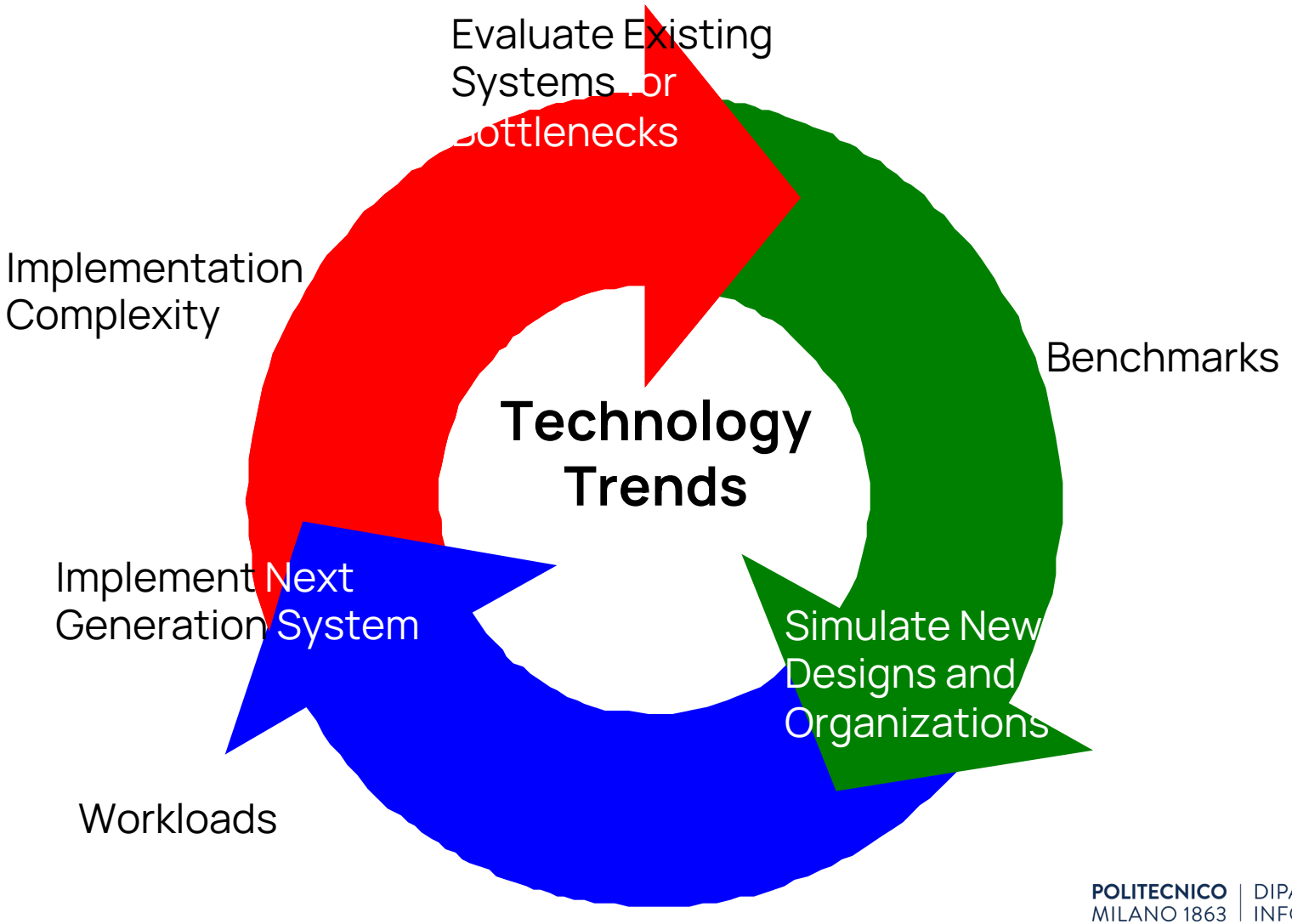
Issues as new opportunities

Programming has become very difficult

Impossible to balance all constraints manually

- More computational horse-power than ever before
  - Cores are free
- Energy is new constraint
  - Software must become energy and space aware

Computer Engineering Methodology



Performance Evaluation

- When we say that one computer is faster than another what do we mean?
  - It depends on what is important

- Two Metrics:
  - Computer system user
    - Minimize elapsed time for program execution:
      response time: execution time = time_end – time_start
  - Computer center manager
    - Maximize completion rate = #jobs/sec
      throughput: total amount of work done in a given time

Response time vs throughput

- Is throughput = 1/average response time?
  - YES only if NO overlap
  - Otherwise throughput > 1/average response time
  - Example:
    - A lunch buffet with 5 stations
    - Each person takes 2 minutes at each station
    - Time per person to fill up the tray is 10 minutes
    - BUT throughput is 1 person every 2 minutes
    - Overlap: 5 people simultaneously filling the tray
    - Without overlap throughput = 1/10

Overview of Factors Affecting Performance

- Algorithm complexity and data set
- Compiler
- Instruction set
- Available operations
- Operating system
- Clock rate
- Memory system performance
- I/O system performance and overhead

The Bottom Line: Performance (and Cost)

Plane	DC to Paris
Boeing 747	6.5 hours
BAD/Sud Concorde	3 hours

Time to run the task (**ExTime**)

- Execution time, response time, latency

Tasks per day, hour, week, sec, ns … (**Performance**)

- Throughput, bandwidth

The Bottom Line: Performance (and Cost)

Plane	DC to Paris	Speed
Boeing 747	6.5 hours	610 mph
BAD/Sud Concorde	3 hours	1350 mph

Time to run the task (**ExTime**)

- Execution time, response time, latency

Tasks per day, hour, week, sec, ns … (**Performance**)

- Throughput, bandwidth

The Bottom Line: Performance (and Cost)

Plane	DC to Paris	Speed	Passengers
Boeing 747	6.5 hours	610 mph	470
BAD/Sud Concorde	3 hours	1350 mph	132

Time to run the task (**ExTime**)

- Execution time, response time, latency

Tasks per day, hour, week, sec, ns … (**Performance**)

- Throughput, bandwidth

The Bottom Line: Performance (and Cost)

Plane	DC to Paris	Speed	Passengers	Throughput (pmph)
Boeing 747	6.5 hours	610 mph	470	286,700
BAD/Sud Concorde	3 hours	1350 mph	132	178,200

Time to run the task (**ExTime**)

- Execution time, response time, latency

Tasks per day, hour, week, sec, ns … (**Performance**)

- Throughput, bandwidth

The Bottom Line: Performance (and Cost)

WHAT ABOUT TICKETS PRICES?

286,700
178,200

Time to run the task (**ExTime**)

- Execution time, response time, latency

Tasks per day, hour, week, sec, ns … (**Performance**)

- Throughput, bandwidth

The Bottom Line: Performance (and Cost)

WHAT ABOUT TICKETS PRICES?

BOEING 100$ each

CONCORD 200$ each

286,700
178,200

Time to run the task (**ExTime**)

- Execution time, response time, latency

Tasks per day, hour, week, sec, ns … (**Performance**)

- Throughput, bandwidth

The Bottom Line: Performance

"X is n times faster than Y" means

$$\frac{\text{ExTime(Y)}}{\text{ExTime(X)}} = \frac{\text{Performance(X)}}{\text{Performance(Y)}}$$

Speed of Concorde vs. Boeing 747

1350/610 = 2.2

Throughput of Boeing 747 vs. Concorde

286700/178200 = 1.6

Speedup

"X is n% faster than Y" $\Rightarrow \frac{\text{execution time (y)}}{\text{execution time (x)}} = 1 + \frac{n}{100}$

$$\text{performance}(x) = \frac{1}{\text{execution\_time}(x)}$$

"X is n% faster than Y" $\Rightarrow \frac{\text{performance}(x)}{\text{performance}(y)} = 1 + \frac{n}{100}$

Speedup

$$\text{“X is n\% faster than Y”} \Rightarrow \frac{\text{execution time (y)}}{\text{execution time (x)}} = 1 + \frac{n}{100}$$

$$\text{performance}(x) = \frac{1}{\text{execution\_time}(x)}$$

$$\text{“X is n\% faster than Y”} \Rightarrow \frac{\text{performance}(x)}{\text{performance}(y)} = 1 + \frac{n}{100}$$

$$\text{Speedup}(x,y) = \text{Performance}(x)/\text{Performance}(y)$$

Focus on the Common Case

- Common sense guides computer design
  - Since its engineering, common sense is valuable
- In making a design trade-off, favor the frequent case over the infrequent case
  - E.g., Instruction fetch and decode unit used more frequently than multiplier, so optimize it 1st
  - E.g., If database server has 50 disks / processor, storage dependability dominates system dependability, so optimize it 1st

Frequent case

- Frequent case is often simpler and can be done faster than the infrequent case
- **What is frequent case and how much performance improved by making case faster**

Frequent case

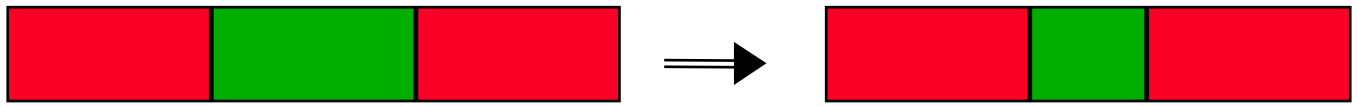
- Frequent case is often simpler and can be done faster than the infrequent case
- **What is frequent case and how much performance improved by making case faster**
  => Amdahl's Law

Amdahl's Law

--> you cannot accelerate everything, and your latency is dominated by what you cannot accelerate.

Speedup due to enhancement E:

$$\text{Speedup(E)} = \frac{\text{ExTime w/o E}}{\text{ExTime w/ E}} = \frac{\text{Performance w/ E}}{\text{Performance w/o E}}$$



Suppose that enhancement E accelerates a fraction F of the task by a factor S, and the remainder of the task is unaffected

Amdahl's Law

- Expresses the law of diminishing return

- Corollary

  If an enhancement is only usable for a fraction of a task we can't speed up the task by more than the reciprocal of 1 minus the fraction

Serves as a guide to how much an enhancement will improve performance and how to distribute resources to improve cost/performance

Breaking down performance

A **program** is broken into **instructions**

- Hardware is aware of instructions, not programs

At a lower level, hardware breaks **instructions** into **clock cycles**

Lower-level state machines change state every cycle

For example:

500 MHz P-III runs 500M cycles/sec, 1 cycle = 2 ns

2 GHz P-IV runs 2G cycles/sec, 1 cycle = 0,5 ns

What is performance for us?

For computer architects

- Response time = latency due to the completion of a task including disk accesses, I/O activity, OS, …
  Elapsed time = CPU time + I/O wait
- CPU time = does !include I/O wait time and corresponds to CPU
  CPU time = time spent running a program

$$\text{CPU time (P)} = \frac{\text{clock cycles (cc) needed to execute P}}{\text{clock frequency}}$$

Or

CPU time (P) = cc needed to execute P x cc time

CPU time

Processor Performance = $\frac{\text{Time}}{\text{Program}}$

$$\text{CPU time} = \frac{\text{Seconds}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Cycle}}$$

$\frac{\text{Instructions}}{\text{Program}}$: Instruction Count, code size

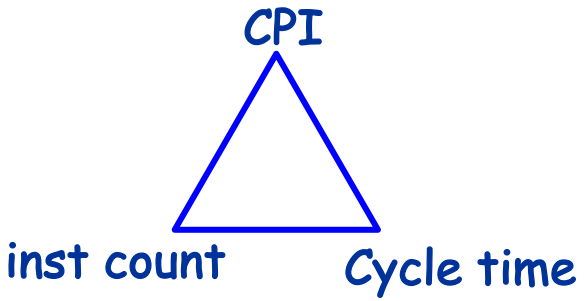
$\frac{\text{Cycles}}{\text{Instruction}}$: CPI

$\frac{\text{Time}}{\text{Seconds}}$: cycle time

CPU time

- Instruction Count, IC
  - Instructions executed, not static code size
  - Determined by algorithm, compiler, Instruction Set Architecture
- Cycles per instructions, CPI
  - Determined by ISA and CPU organization
  - Overlap among instructions (pipelining) reduces this term
- Time/cycle
  - Determined by technology, organization and circuit design

Performance equation



	Inst. Count	CPI	Clock Rate
Program	X		
Compiler	X	(X)	
Instr. Set	X	X	
Organization		X	X
Technology			X

Goal of CPU performance

- Minimize time which is the product, not isolated terms
- Common error to miss terms while devising optimizations
  - E.g. ISA change to decrease instruction count
  - BUT leads to CPU organization which may make clock slower
- **BOTTOM LINE: terms are inter-related**

Average CPI

The *average* Clock Cycles per Instruction (CPI) can be defined as:

$$CPI(P) = \frac{\text{clock cycles needed to exec. P}}{\text{number of instructions}}$$

$$CPUtime = Tclock * CPI * Ninst = (CPI * Ninst)/f$$

Cycles per instruction

Average Cycles per Instruction

CPI = (CPU Time * Clock Rate) / Instr Count = Cycles/Instr Count

$$\text{CPU time} = \text{CycleTime} * \sum_{i=1}^{n} \text{CPI}_i * \text{I}_i$$

Instruction Frequency

$$\text{CPI} = \sum_{i=1}^{n} \text{CPI}_i * \text{F}_i \quad \text{where} \quad \text{F}_i = \frac{\text{I}_i}{\text{Instruction Count}}$$

Invest Resources where time is Spent!

Aspects of CPU performance

The CPI can vary among instructions

$$\text{CPU time} = \text{CycleTime} * \sum_{i=1}^{n} \text{CPI}_i * \text{I}_i$$

Aspects of CPU performance

The CPI can vary among instructions

$$\text{CPU time} = \text{CycleTime} * \sum_{i=1}^{n} \text{CPI}_i * \text{I}_i$$

Example

OPER
ALU
Load
Store
Branch

Aspects of CPU performance

The CPI can vary among instructions

$$\text{CPU time} = \text{CycleTime} * \sum_{i=1}^{n} \text{CPI}_i * \text{I}_i$$

Example

OPER	FREQ
ALU	50%
Load	20%
Store	10%
Branch	20%

Aspects of CPU performance

The CPI can vary among instructions

$$\text{CPU time} = \text{CycleTime} * \sum_{i=1}^{n} \text{CPI}_i * \text{I}_i$$

Example

OPER	FREQ	CYCLES
ALU	50%	1
Load	20%	2
Store	10%	2
Branch	20%	2

Aspects of CPU performance

The CPI can vary among instructions

$$\text{CPU time} = \text{CycleTime} * \sum_{i=1}^{n} \text{CPI}_i * \text{I}_i$$

Example

OPER	FREQ	CYCLES	CPI
ALU	50%	1	.5
Load	20%	2	.4
Store	10%	2	.2
Branch	20%	2	.4
			1.5

Aspects of CPU performance

The CPI can vary among instructions

$$\text{CPU time} = \text{CycleTime} * \sum_{i=1}^{n} \text{CPI}_i * \text{I}_i$$

Example

OPER	FREQ	CYCLES	CPI	COMP
ALU	50%	1	.5	0.5/1.5
Load	20%	2	.4	0.4/1.5
Store	10%	2	.2	0.2/1.5
Branch	20%	2	.4	0.4/1.5
			1.5	

Aspects of CPU performance

The CPI can vary among instructions

$$\text{CPU time} = \text{CycleTime} * \sum_{i=1}^{n} \text{CPI}_i * \text{I}_i$$

Example

OPER	FREQ	CYCLES	CPI	COMP	% TIME
ALU	50%	1	.5	0.5/1.5	33%
Load	20%	2	.4	0.4/1.5	27%
Store	10%	2	.2	0.2/1.5	13%
Branch	20%	2	.4	0.4/1.5	27%
			1.5		

Other metrics

MIPS and MFLOPS

- MIPS = millions of instructions per second
  = number of instructions / (execution time x 10^6)
  = clock frequency/(CPI x 10^6)
- Execution time = Instruction count / (MIPS x 10^6)
- Since MIPS is a rate of operations per unit time, performance can be specified as the inverse of execution time, with faster machines having higher MIPS rating
- BUT MIPS has serious shortcomings

Example

Execution of a loop (a = a + a x b) 1 million times

ARCH 1 - 1 MHz

ADD 1 cycle

MUL 2 cycles

ARCH2 - 1 MHz

MAC 2 cycles

Example

Execution of a loop (a = a + a x b) 1 million times

ARCH 1 - 1 MHz

ADD 1 cycle

MUL 2 cycles

ARCH2 - 1 MHz

MAC 2 cycles

$$CPI(P) = \frac{\text{clock cycles needed to exec. P}}{\text{number of instructions}}$$

Example

Execution of a loop (a = a + a x b) 1 million times

ARCH 1 - 1 MHz

ADD 1 cycle

MUL 2 cycles

CPI=1.5

ARCH2 - 1 MHz

MAC 2 cycles

CPI=2

Example

Execution of a loop (a = a + a x b) 1 million times

ARCH 1 - 1 MHz
ADD 1 cycle
MUL 2 cycles
CPI=1.5
MIPS=1/1.5 = 0.66
EX time = 3 sec

ARCH2 - 1 MHz
MAC 2 cycles

CPI=2
MIPS = 1/2 = 0.5
Ex Time = 2 sec

MFLOPS

$$\text{MFLOPS} = \frac{\text{Floating Point operations in program}}{\text{CPU time} \times 10^6}$$

Assuming FP operations independent of compiler and ISA

- Often safe for numeric codes: matrix size determines # of FP ops/program
- However, not always safe:
  - Missing instructions (e.g. FP divide, square rot, sin, cos)
  - Optimizing compilers

Benchmarks

- Execution time of what program?
- Standard performance test programs: benchmarks
  - Programs chosen to measure performance defined by some group
  - Available to the community
  - Run on machines and performance is reported
  - Can compare to reports on other machines
  - Representative?

Types of benchmarks

- Real programs
  - Representative of real workload
  - Only accurate way to characterize performance
  - E.g. C compilers, text processing, other applications
  - Sometimes modified: CPU oriented benchmarks may remove I/O
- Kernels or microbenchmarks
  - “Representative” program fragments
  - Good for focusing on individual features
- Synthetic benchmarks
  - Same philosophy of kernels
  - Try to match the average frequency of operations and operands of a large set of programs
- Instruction mixes (for CPI)

Conclusions

- For better or worse, benchmarks “shape a field”
- Good products created when have:
  - Good benchmarks
  - Good ways to summarize performance

- Given sales is a function in part of performance relative to competition, investment in improving product as reported by performance summary
- If benchmarks/summary inadequate, then choose between improving product for real programs vs. improving product to get more sales
  - Sales almost always wins!

- Execution time is the measure of computer performance!

Introducing Energy and Power

Problem definition

Energy and power consumption - **Are they a constraint for…**

Problem definition

Energy and power consumption - **Are they a constraint for…**

- Embedded & IoT
- Mobile devices

Problem definition

Energy and power consumption - **Are they a constraint for…**

- Embedded & IoT
- Mobile devices

Batteries capacities

Problem definition

Energy and power consumption - **Are they a constraint for…**

- Embedded & IoT
- Mobile devices
- Desktops
- Servers
- HPC Clusters

Batteries capacities

Problem definition

Energy and power consumption - **Are they a constraint for…**

- Embedded & IoT
- Mobile devices
- Desktops
- Servers
- HPC Clusters

Batteries capacities

Energy costs

Problem definition

Energy and power consumption - **Are they a constraint for…**

- Embedded & IoT
- Mobile devices
- Desktops
- Servers
- HPC Clusters

Energy and Power budgets

Problem definition

Energy and power consumption - **Are they a constraint for…**

- Embedded & IoT
- Mobile devices
- Desktops
- Servers
- HPC Clusters

Energy and Power budgets

Any optimization needs a deep understanding of the phenomenon

Power and Energy

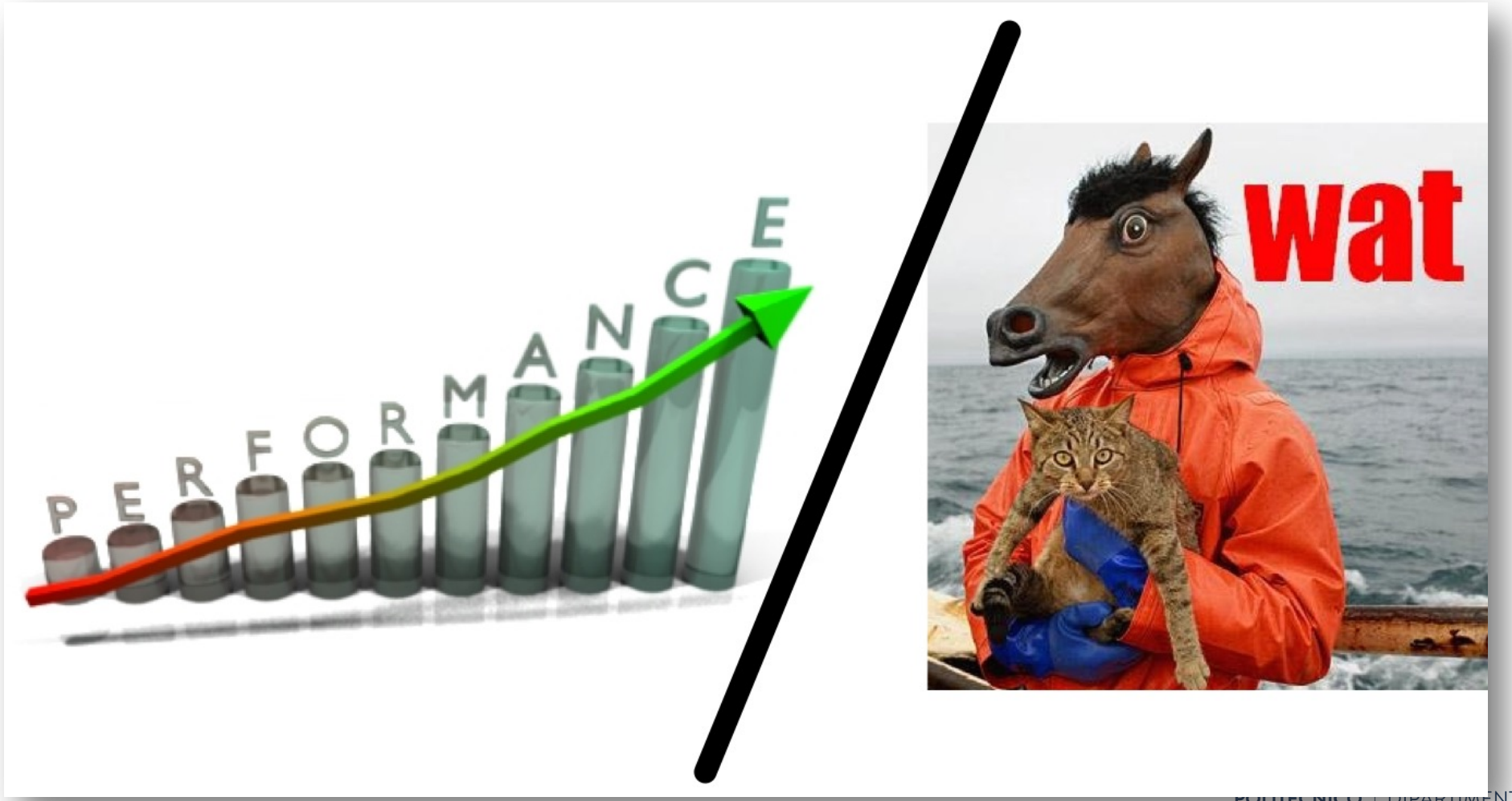
- Problem: Get power in, get power out
- Thermal Design Power (TDP)
  - Characterizes sustained power consumption
  - Used as target for power supply and cooling system
  - Lower than peak power, higher than average power consumption
- Clock rate can be reduced dynamically to limit power consumption
- Energy per task is often a better measurement

Reducing Power

- Techniques for reducing power:
  - Do nothing well
  - Dynamic Voltage-Frequency Scaling
  - Low power state for DRAM, disks
  - Overclocking, turning off cores

- Static power consumption
  - $\text{Current}_{\text{static}}$ x Voltage
  - Scales with number of transistors
  - To reduce: power gating

Energy and Power, but not only…



Other factors?

Factors Determining Cost

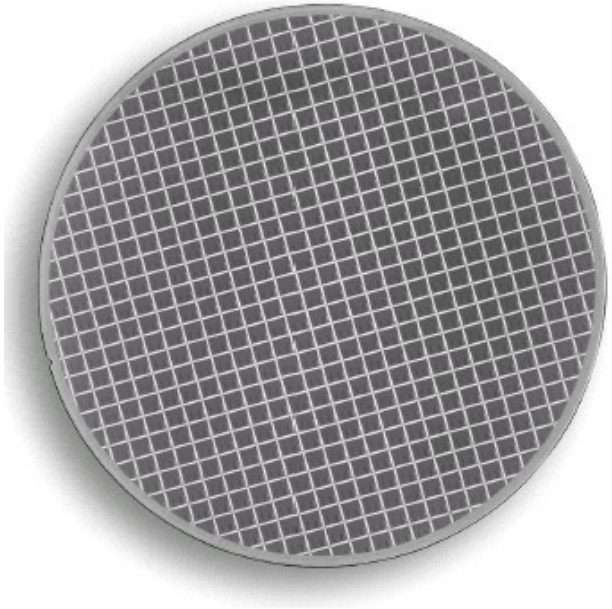
- Cost: amount spent by manufacturer to produce a finished good
- High volume → faster learning curve, increased manufacturing efficiency (10% lower cost if volume doubles), lower R&D cost per produced item
- Commodities: identical products sold by many vendors in large volumes (keyboards, DRAMs) – low cost because of high volume and competition among suppliers

Trends in Cost

- Cost driven down by learning curve
  - Yield
- DRAM: price closely tracks cost
- Microprocessors: price depends on volume
  - 10% less for each doubling of volume

Wafers and Dies

An entire wafer is produced and chopped into dies that undergo testing and packaging



Integrated Circuit Cost

Cost of an integrated circuit =
(cost of die + cost of packaging and testing) / final test yield

Cost of die = cost of wafer / (dies per wafer x die yield)

Dies/wafer = wafer area / die area -
p wafer diam / die diag

Die yield = wafer yield x
(1 + (defect rate x die area) / a)$^{-a}$

Thus, die yield depends on die area and complexity arising from multiple manufacturing steps (a ~ 4.0)

Integrated Circuit Cost

Integrated circuit

$$\text{Cost of integrated circuit} = \frac{\text{Cost of die} + \text{Cost of testing die} + \text{Cost of packaging and final test}}{\text{Final test yield}}$$

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

- Bose-Einstein formula:

$$\text{Dies per wafer} = \frac{\pi \times (\text{Wafer diameter}/2)^2}{\text{Die area}} - \frac{\pi \times \text{Wafer diameter}}{\sqrt{2 \times \text{Die area}}}$$

- Defects per unit area = 0.016-0.057 defects per square cm (2010)

$$\text{Die yield} = \text{Wafer yield} \times 1/(1 + \text{Defects per unit area} \times \text{Die area})^N$$

- N = process-complexity factor = 11.5-15.5 (40 nm, 2010)

Dependability

- Module reliability
  - Mean time to failure (MTTF)
  - Mean time to repair (MTTR)
  - Mean time between failures (MTBF) = MTTF + MTTR
  - Availability = MTTF / MTBF

Questions?