POLITECNICO DI MILANO

μ-LAB

High Performance Processors and Systems

Performance and cost

- Hennessy Patterson chapter 1-

Donatella Sciuto: sciuto@elet.polimi.it

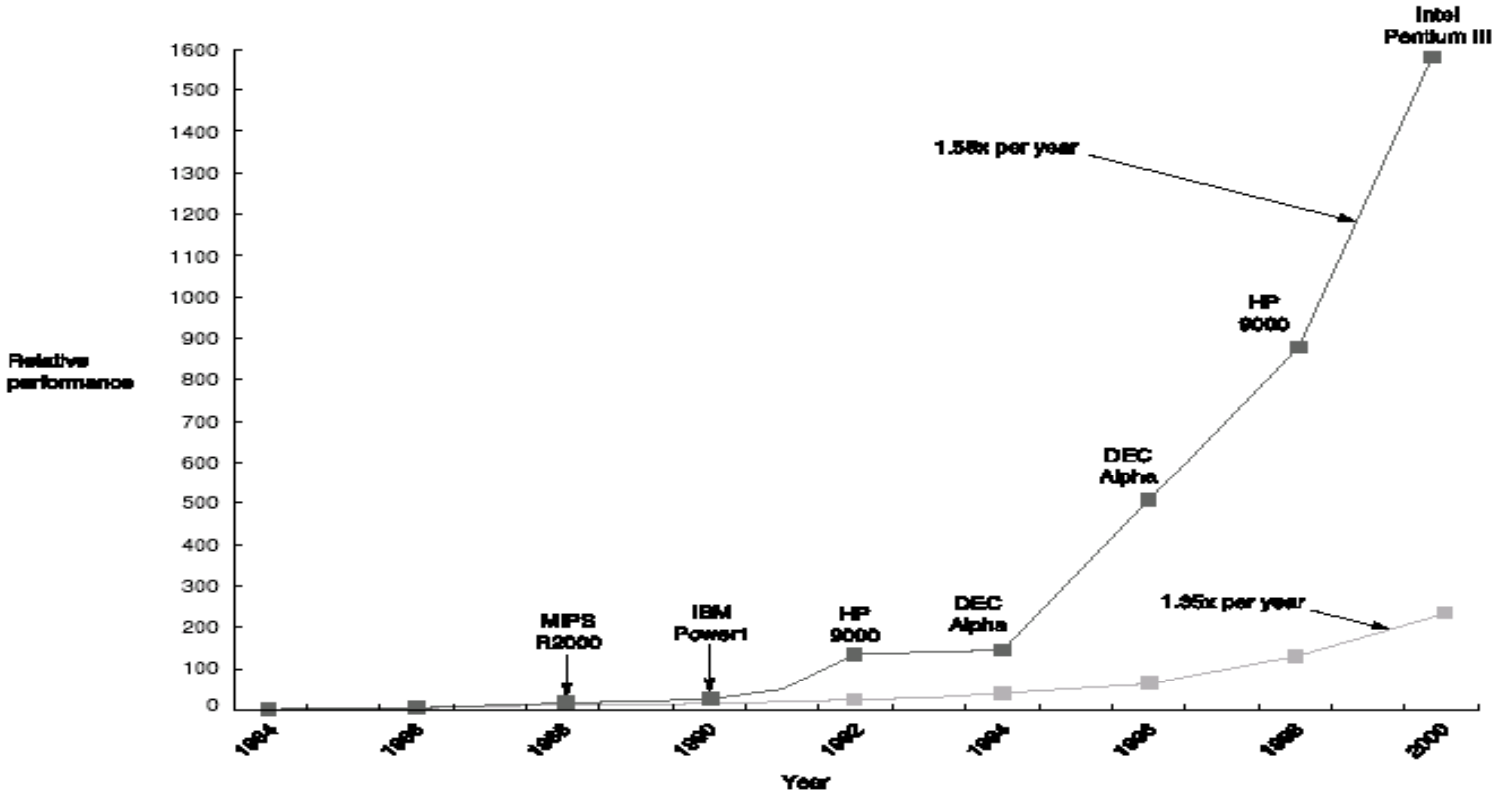
HPPS

Outline

- Measures to evaluate performance
- Quantifying the design process
  - Amdahl's law
  - CPU time and CPI
- Other metrics: MIPS and MFLOPS
- Summarize performance
- Energy/Power
- Cost

Microprocessor performance

15x performance growth can be attributed to architectural innovations



Where are we headed?

Modern trends:

- Clock speed improvements are slowing
  - power constraints
  - already doing less work per stage
- Difficult to further optimize a single core for performance
- Multi-cores: each new processor generation will accommodate more cores

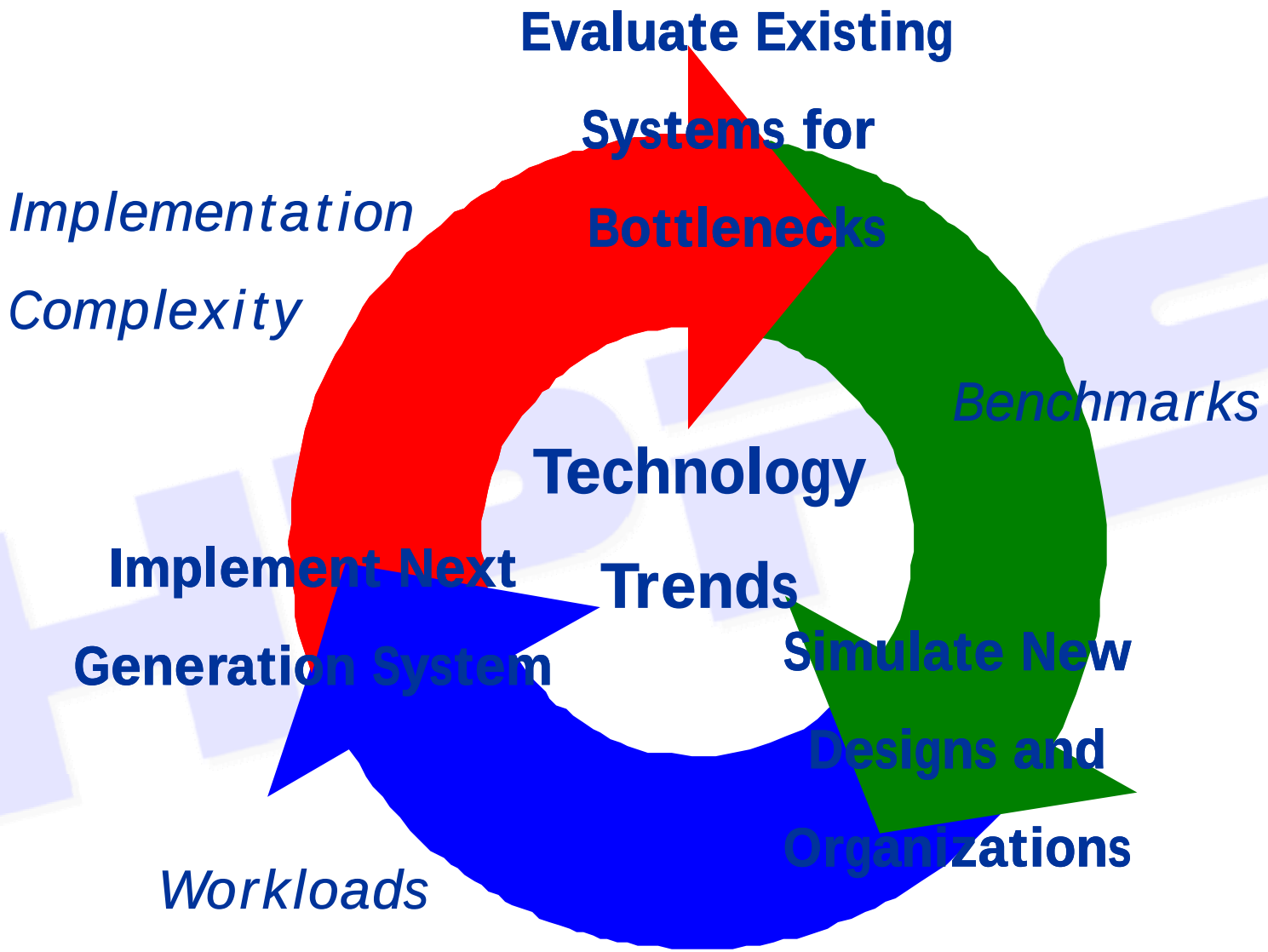
Processor Technology trends

- Shrinking of transistor sizes: 250nm (1997) → 130nm (2002) → 65nm (2007) → 22nm
- Transistor density increases by 35% per year and die size increases by 10-20% per year… more cores!
- Transistor speed improves linearly with size (complex equation involving voltages, resistances, capacitances)… clock speed improvements!
- Wire delays do not scale down at the same rate as logic delays… the Pentium 4 has pipeline stages for wire delays

Technology trends

- DRAM density increases by 40-60% per year, latency has reduced by 33% in 10 years (the memory wall!), bandwidth improves twice as fast as latency decreases
- Disk density improves by 100% every year, latency improvement similar to DRAM
- Networks: primary focus on bandwidth; 10Mb → 100Mb in 10 years; 100Mb → 1Gb in 5 years

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
    - WHY?
    - 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	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

"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

- 1650/610 = 2.2

Throughput of Boeing 747 vs. Concorde

- 286700/178200 = 1.6

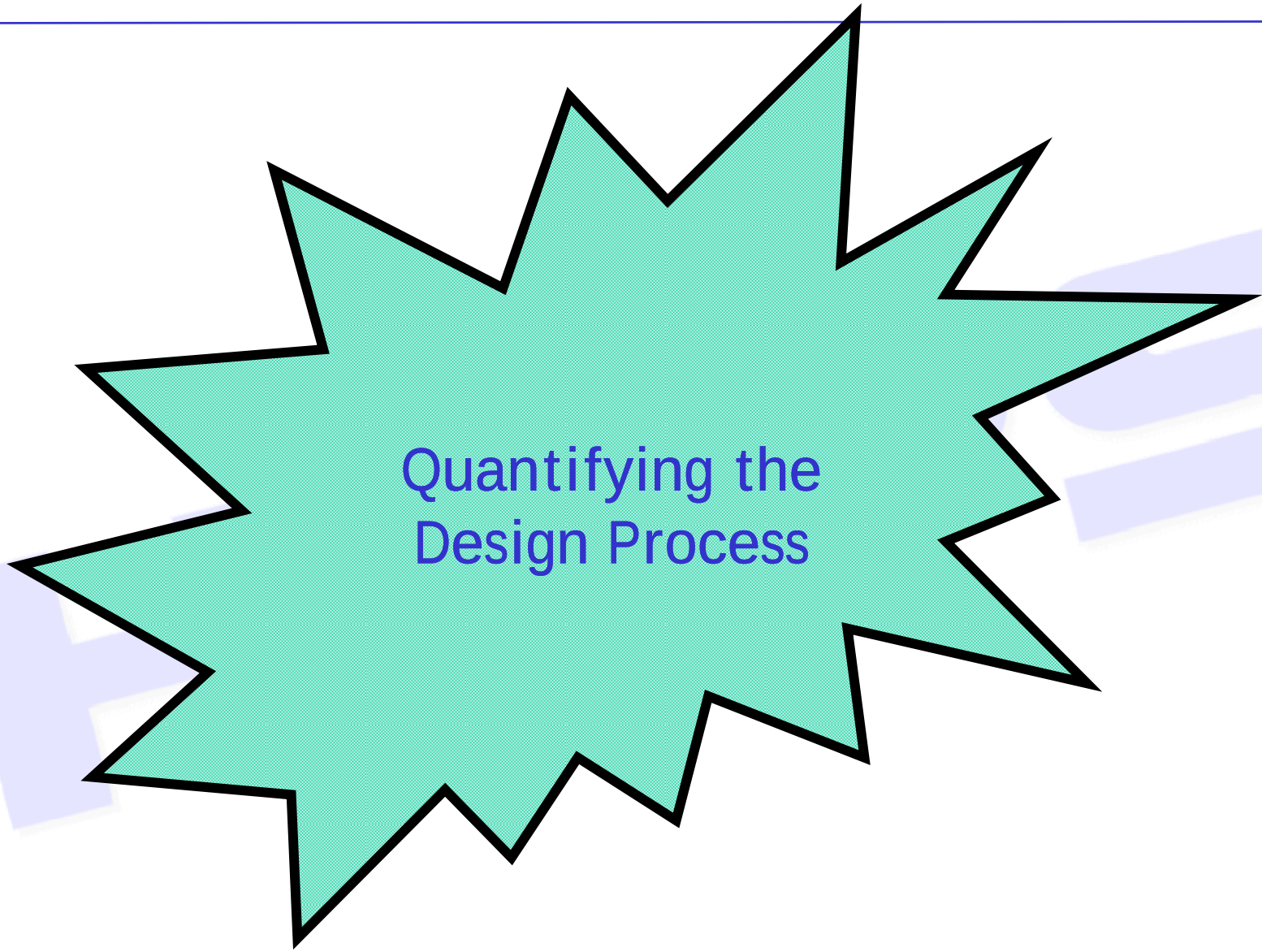
Speedup

X is n% faster than Y if:

$$\text{Speedup}(x,y) = \frac{\text{ExTime}(y)}{\text{ExTime}(x)} = 1+n/100$$

Performance(A)=1/ExTime(A).

Speedup(x,y)= Performance(x)/Performance(y)


Quantifying the
Design Process

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

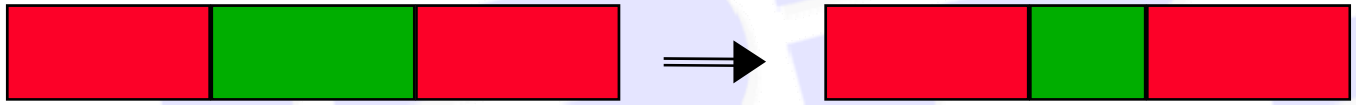
Frequent case

- Frequent case is often simpler and can be done faster than the infrequent case
  - 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

Amdahl's Law

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

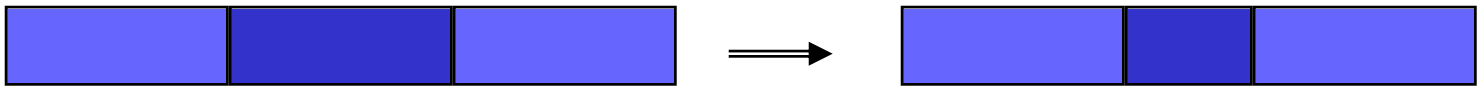
Amdahl's Law

$$ExTime_{new} = ExTime_{old} \times \left[\left(1 - Fraction_{enhanced}\right) + \frac{Fraction_{enhanced}}{Speedup_{enhanced}} \right]$$

$$Speedup_{overall} = \frac{ExTime_{old}}{ExTime_{new}} = \frac{1}{\left(1 - Fraction_{enhanced}\right) + \frac{Fraction_{enhanced}}{Speedup_{enhanced}}}$$

Best you could ever hope to do:

$$Speedup_{maximum} = \frac{1}{\left(1 - Fraction_{enhanced}\right)}$$



Amdahl's Law example

- New CPU 10X faster
- I/O bound server, so 60% time waiting for I/O

$$\text{Speedup}_{\text{overall}} = \frac{1}{\left(1 - \text{Fraction}_{\text{enhanced}}\right) + \frac{\text{Fraction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}}}$$

$$= \frac{1}{\left(1 - 0.4\right) + \frac{0.4}{10}} = \frac{1}{0.64} = 1.56$$

- Apparently, its human nature to be attracted by 10X faster, vs. keeping in perspective its just 1.6X faster

Amdahl's Law

- Floating point instructions improved to run 2X; but only 10% of actual instructions are FP

$$ExTime_{new} = ExTime_{old} \times (0.9 + 0.1/2) = 0.95 \times ExTime_{old}$$

$$Speedup_{overall} = \frac{1}{0.95} = 1.053$$

Law of diminishing return:

Focus on the common case!

Example

- Enhancement that runs 10 times faster than the original machine but is only usable 40% of the time. Speedup?

$$\textbf{Speedup} = \frac{1}{0.6 + 0.4/10} \cong 1.56$$

Exercise on Amdahl's Law

Let's assume that we can improve the CPU speed 5X (with a 5X cost).

Suppose that the CPU is used 50% of the time and that the base CPU cost is 1/3 of the entire system

Is it worth to upgrade the CPU?
Compare speedup and costs!

Solution

- Speedup=1/(0.5+0.5/5)=1.67
- Increased= (2/3)+(1/3)*5=2.33

➔ It is not worth to upgrade the CPU!

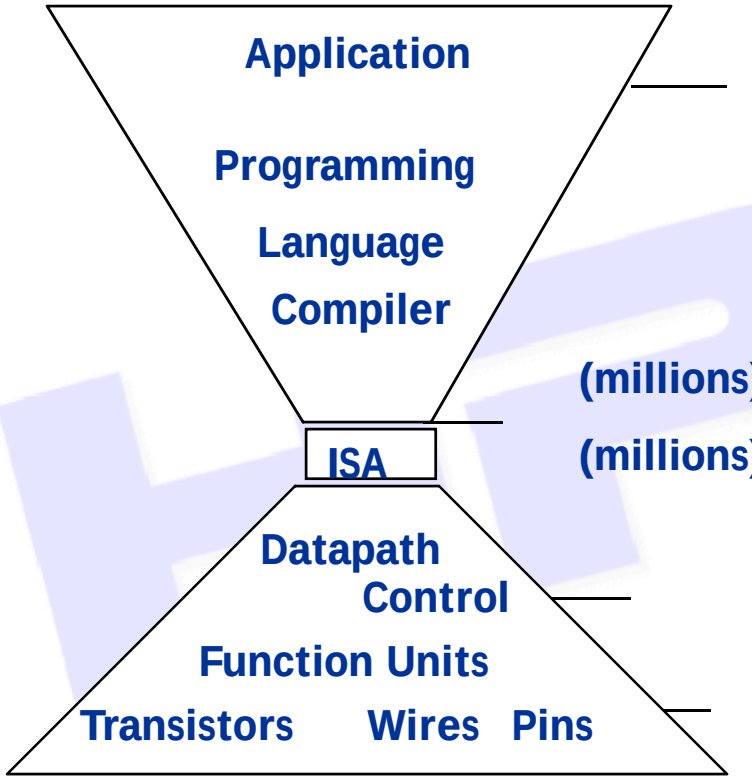
Amdahl's Law

- 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

Metrics of performance



Answers per month

Operations per second

(millions) of Instructions per second: MIPS

(millions) of (FP) operations per second: MFLOP/s

Megabytes per second

Cycles per second (clock rate)

Breaking down performance

- A program is broken into instructions
  - Hardware is aware of instructions not programs
- At 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/cycles, 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 not include I/O wait time and corresponds to CPU

  CPU time = time spent running a program

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

Or

CPU time (P) =

clock cycles needed to execute P x clock cycle 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

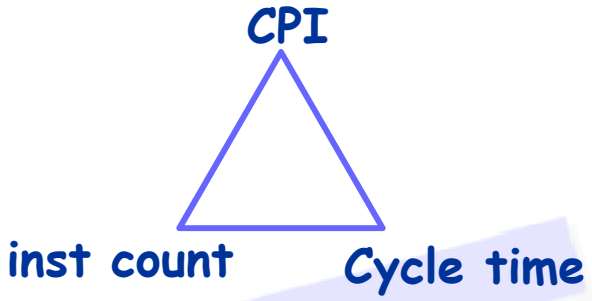
$\frac{\text{Cycles}}{\text{Instruction}}$: CPI

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

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

Instruction Frequency

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

Invest Resources where time is Spent!

Aspects of CPU performance

- The CPI can vary among instructions

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

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		

Exercise

- Procedure calls in architecture A are very expensive.
- Suppose to introduce a new architecture similar B such that:
  - A has a clock 5% faster than B.
  - The fraction of loads/stores of A is 30%.
  - B executes 30% loads/stores less than A
  - Loads/stores require 1 clock cycle.
- Compare CPU times of A and B.

Solution

Number of instr. of B

NB = [1-(0.3x0.3))]*NA=0.91*NA

Clock Period of B: TB=TA*1.05

CPUtimeA = 1*NA*TA

CPUtimeB = 0.9*NA*TA*1.05*1 = 0.945*CPUtimeA

Exercise

- A RISC processor shows the following statistics:

OPERATION	FREQ	CYCLES
ALU	50%	1
Load	20%	5
Store	10%	3
Branch	20%	2

Compute the average CPI and the speedup w.r.t.:

1. The same machine with an improved memory (load cycles = 2)
2. The same machine with a branch CPI=1
3. The same machine with 2 ALUs working in parallel

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

MIPS problems

- Depends heavily on the ISA
  - Difficult to compare MIPS of computers with different instruction sets
  - Depends on the benchmark
  - It can be the inverse of performance!
  - Example:
    - Without FP hardware an FP op may require 50 cycle instr., with FP hardware, only one 2-cycle instruction
    - Adding FP hardware:
      - Decreases IC (why?) 50 → 1
      - Increases CPI (why?) 1 → 2
      - CPU time decreases 50 → 2
    - BUT MIPS gets worse 50 MIPS → 2 MIPS !!!

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 = 0.5

Ex Time = 2 sec

MFLOPS

- 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)

Benchmarks: SPEC2000

- System Performance Evaluation Cooperative
  - Formed in 89 to combat benchmarketing
  - SPEC89, SPEC92, SPEC95, now SPEC2000
- 12 integer and 14 floating point programs
- Compute intensive performance of:
  - The CPU
  - The memory architecture
  - The compilers

SPEC INT 2000 benchmarks

Benchmark	Description
164.gzip	Compression
175.vpr	FPGA place and route
176.gcc	C compiler
181.mcf	Combinatorial optimization
186.crafty	Chess
197.parser	Word processing, grammatical analysis
252.eon	Visualization (ray tracing)
253.perlbmk	PERL script execution
254.gap	Group theory interpreter
255.vortex	Object-oriented database
256.bzip2	Compression
300.twolf	Place and route simulator

SPEC FP 2000 benchmarks

Benchmark	Description
164.gzip	Compression
175.vpr	FPGA place and route
176.gcc	C compiler
181.mcf	Combinatorial optimization
186.crafty	Chess
197.parser	Word processing, grammatical analysis
252.eon	Visualization (ray tracing)
253.perlbmk	PERL script execution
254.gap	Group theory interpreter
255.vortex	Object-oriented database
256.bzip2	Compression
300.twolf	Place and route simulator

Benchmarks pitfalls

- Benchmark not representative
  - If workload is I/O bound SPECint is useless
- Benchmark is too old
  - Benchmarks age poorly; benchmarketing pressure causes vendors to optimize compiler/hardware/software to benchmarks
  - Need to periodically refreshed

How to average performance

	Computer A	Computer B	Computer C
P1	1	10	20
P2	1000	100	20
Total	1001	110	40

- **Which computer is faster?**

Summarize performance

- The simplest approach to summarizing relative performance is to use the total execution time of the two programs
  - B is 9.1 times faster than A for P1 and P2
  - C is 25 times faster than A for P1 and P2
  - C is 2.75 times faster than B for P1 and P2
- Another possibility: **arithmetic mean** $\sum_{i=1}^{n} times$
- **Arithmetic mean of times**
  - **gives the same result:**

    AM(A) = 1001/2 = 500.5

    AM(B) = 110/2 =55

    500.5/55 = 9.1x

$$\frac{1}{n} x \sum_{i=1}^{n} time(i)$$

Valid only if programs run equally often

How to average

- If different frequency of execution use the **weighted arithmetic mean**

$$\left\{ \sum_{i=1}^{n} weight(i) \times time(i) \right\} \div \frac{1}{n}$$

Other averages

- Consider for example that a drive takes 30 mph for the first 10 miles and then 90 mph for the next 10 miles, what is the average speed?
- Average speed = (30+90)/2 **WRONG**
- Average speed = total distance/total time
  = (20 / (10/30 + 10/90)) = 45 mph
- When dealing with **rates (mph) do not use arithmetic mean!!**
- **Another example**
  - **B1: 10 Minst, 1 MIPS -> 10 sec**
  - **B2: 10 Mints, 5 MIPS -> 2 sec**
  - **Average would give 3 MIPS BUT….**
    20 Minst/12 sec = 1.7 MIPS

Harmonic mean

- Harmonic mean of rates

$$\frac{n}{\sum_{i=1}^{n} \frac{1}{rate\ (n)}}$$

- Use Harmonic Mean if forced to work with rates
  - e.g. reporting CPI or miss rates or branch misprediction rates

Dealing with ratios

	Machine A	Machine B
Program 1	1	10
Program 2	1000	100
Total	1001	110

If we take ratios with respect to machine A

	Machine A	Machine B
Program 1	1	10
Program 2	1	0.1

Average for machine A is 1 and for machine B is 5.05

Dealing with ratios

If we take ratios with respect to machine B

	Machine A	Machine B
Program 1	0.1	1
Program 2	10	1
Avergae	5.05	1

- Can't both be true!
- Don't use arithmetic mean on ratios!!
- Use geometric mean

Geometric mean

- Use geometric mean for ratios

$$\sqrt[n]{\prod_{i=1}^{n} ratio(i)}$$

- Independent of reference machine
- In the example the geometric mean w.r.t. machine A and w.r.t machine B is also 1
  - Normalized with respect to either machine

Geometric mean drawbacks

- GM of ratios is not proportional to total time
- Arithmetic mean in the comparison between machine A and B says machine B is 9.1 times faster
- GM says they are equal
- Based on total execution time A and B are equal only if program 1 is run 100 times more than program 2
- Generally GM will mispredict for three or more machines; OK for comparing two
- GM does not predict execution time therefore doubling the performance of the slower benchmak is equal to doubling the performance of the faster….encourages spending time to optimize the most simple benchmarks

What's best?

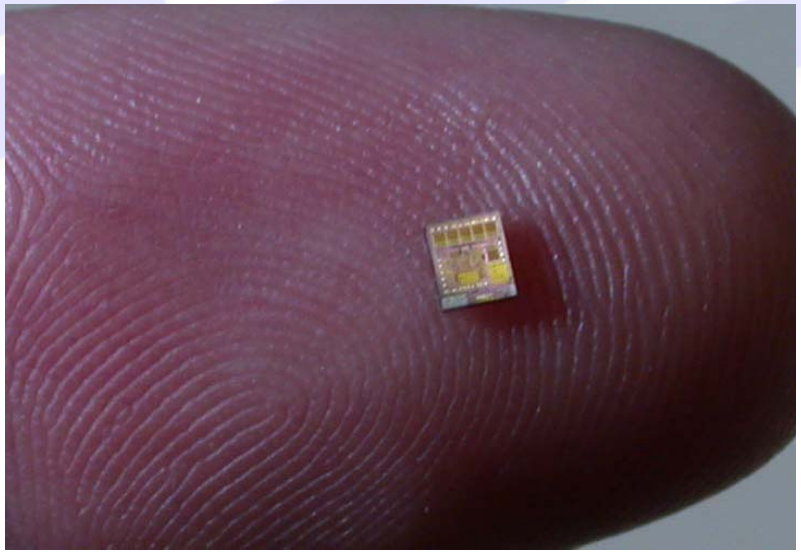
- Use arithmetic mean for times
- Use harmonic mean if forced to use rates
  - Speeding up slower benchmarks gives more reward
  - Reflects total execution time
- Use geometric mean if forced to use ratios
- Best: use unnormalized numbers
  - E.g. CPU time

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!**

It is not just about faster….

- Complete computing systems can be tiny and cheap
- System on a chip
- Resource efficiency
  - Real-estate, power, pins, …

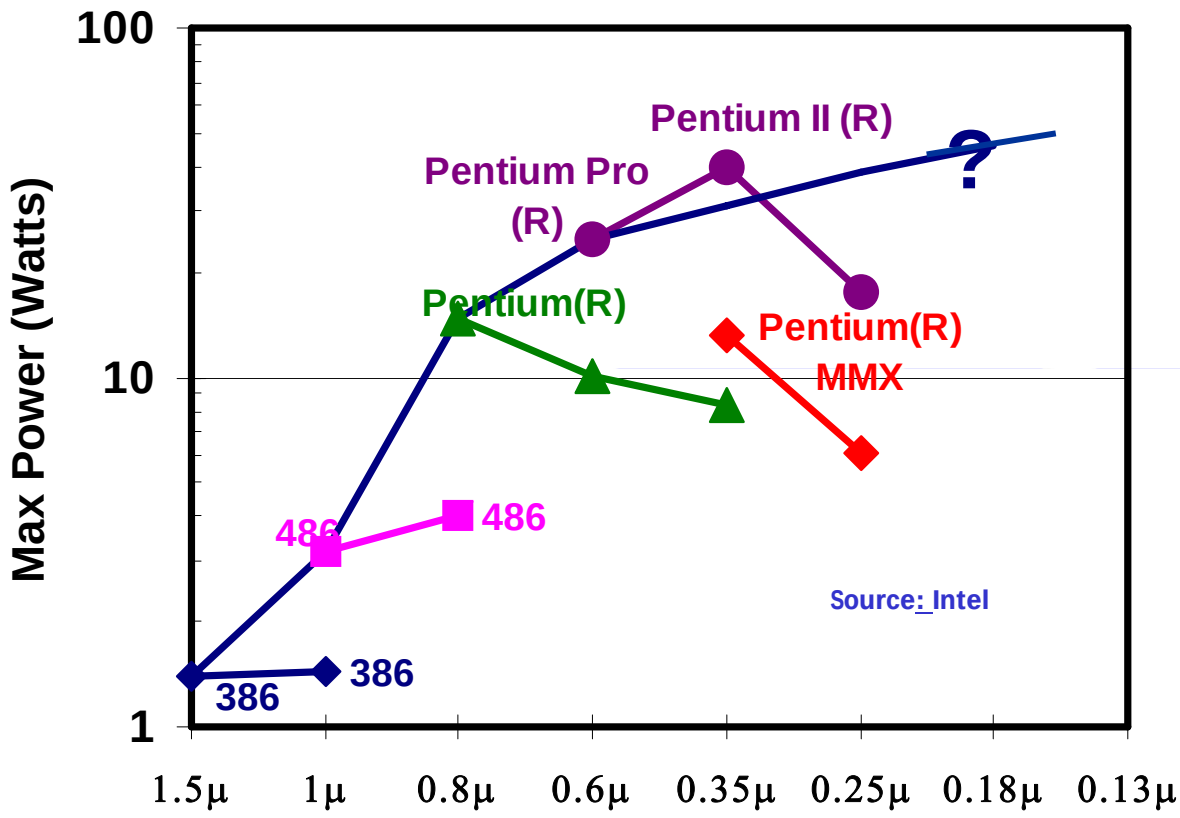


Crossroads: Conventional Wisdom in Comp. Arch

- Old Conventional Wisdom: Power is free, Transistors expensive
- New Conventional Wisdom: "Power wall" Power expensive, Xtors free (Can put more on chip than can afford to turn on)
- Old CW: Sufficiently increasing Instruction Level Parallelism via compilers, innovation (Out-of-order, speculation, VLIW, …)
- New CW: "ILP wall" law of diminishing returns on more HW for ILP
- Old CW: Multiplies are slow, Memory access is fast
- New CW: "Memory wall" Memory slow, multiplies fast (200 clock cycles to DRAM memory, 4 clocks for multiply)
- Old CW: Uniprocessor performance 2X / 1.5 yrs
- New CW: Power Wall + ILP Wall + Memory Wall = Brick Wall
  - Uniprocessor performance now 2X / 5(?) yrs

  ⇒ Change in chip design: multiple "cores" (2X processors per chip / ~ 2 years)
    - More simpler processors are more power efficient

Power / Energy



- Lead processor power increases every generation
- Compactions provide higher performance at lower power

Energy/Power

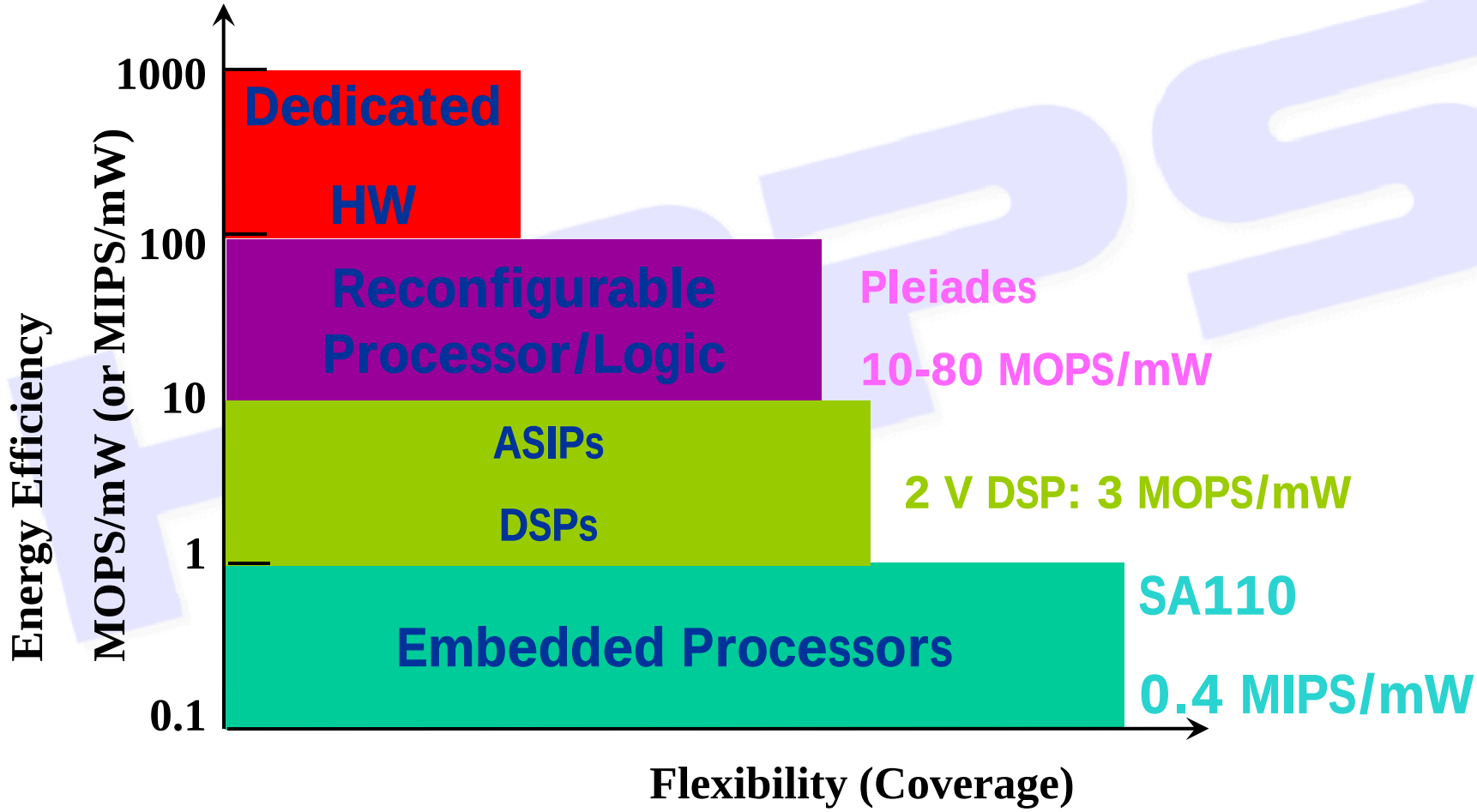
- Power dissipation: rate at which energy is taken from the supply (power source) and transformed into heat

$$P = E / t$$

- Energy dissipation for a given instruction depends upon type of instruction (and state of the processor)

$$\mathbf{P = (1/CPU\ Time) * \sum_{i=1}^{n} E_i * I_i}$$

The Energy-Flexibility Gap

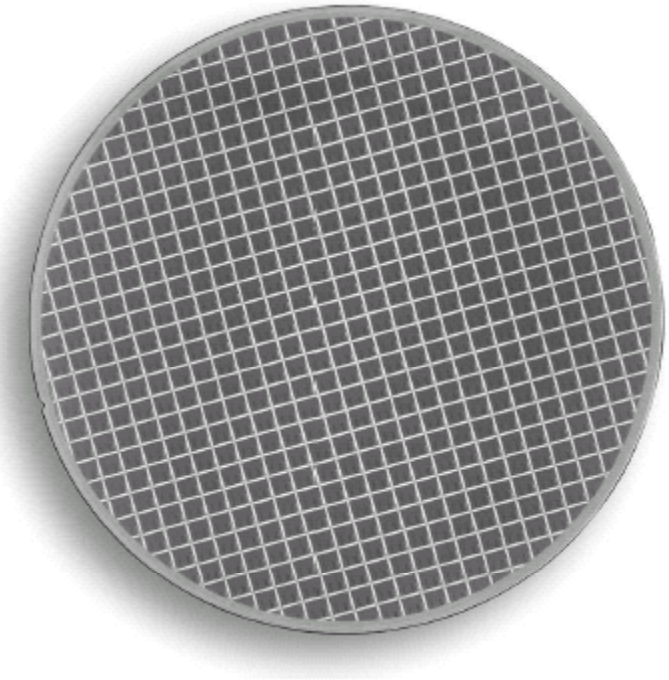


Factors determining cost

- ✓ 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

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 -
π wafer diam / die diag

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

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

Example of IC costs

A 30 cm diameter wafer cost $5-6K in 2001

Such a wafer yields about 366 good 1 cm^2 dies and 1014 good 0.49 cm^2 dies (note the effect of area and yield)

Die sizes: Alpha 21264 1.15 cm^2 , Itanium 3.0 cm^2 , embedded processors are between 0.1 – 0.25 cm^2

Contribution to total system cost

Subsystem	Fraction of total cost
Cabinet: sheet metal, plastic, power supply, fans, cables, nuts, bolts, manuals, shipping box	6%
Processor	22%
DRAM (128 MB)	5%
Video card	5%
Motherboard	5%
Processor board subtotal	37%
Keyboard and mouse	3%
Monitor	19%
Hard disk (20 GB)	9%
DVD drive	6%
I/O devices subtotal	37%
Software (OS + Office)	20%

Questions