浙江大学
ZheJiang University

Computer Architecture ----A Quantitative Approach

Chapter 1

College of Compute of Zhejiang University
CHEN WEN ZHI
chenwz@zju.edu.cn
Room 511, CaoGuangBiao BLD

Topics in Chapter

1.1 Why take this course ?

1.2 Classes of computers in current computer market

1.3 Defining computer architecture and What's the task of computer design?

1.4 Trends in Technology

1.5 Trends in power in Integrated circuits

1.6 Trends in Cost

1.7 Dependability

1.8 Measuring, Reporting and summerizing Perf.

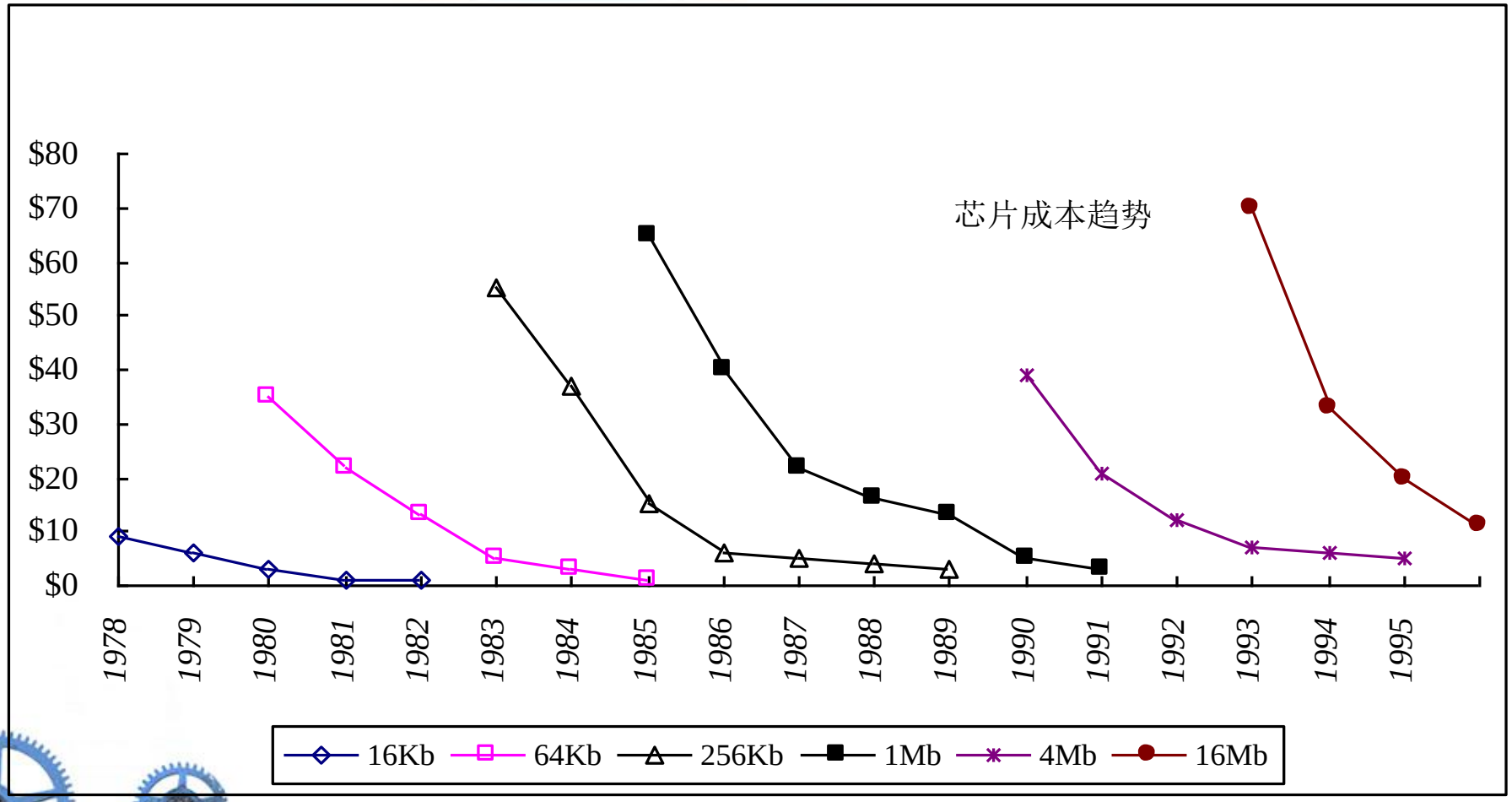
1.9 Quantitative Principles of computer Design

1.10 Putting it altogether

Major Theme: Lower Cost

- Cost Trend
  - Understanding cost trends of component is important for designers, since we design for tomorrow !
- The impact factors for cost:
  - Time----Component prices drop over time without major improvements in manufacturing technology
  - Volume ----Volume decreases cost due to increases in manufacturing efficiency.
  - Commodification----The competition among the suppliers of the components will decrease overall product cost.

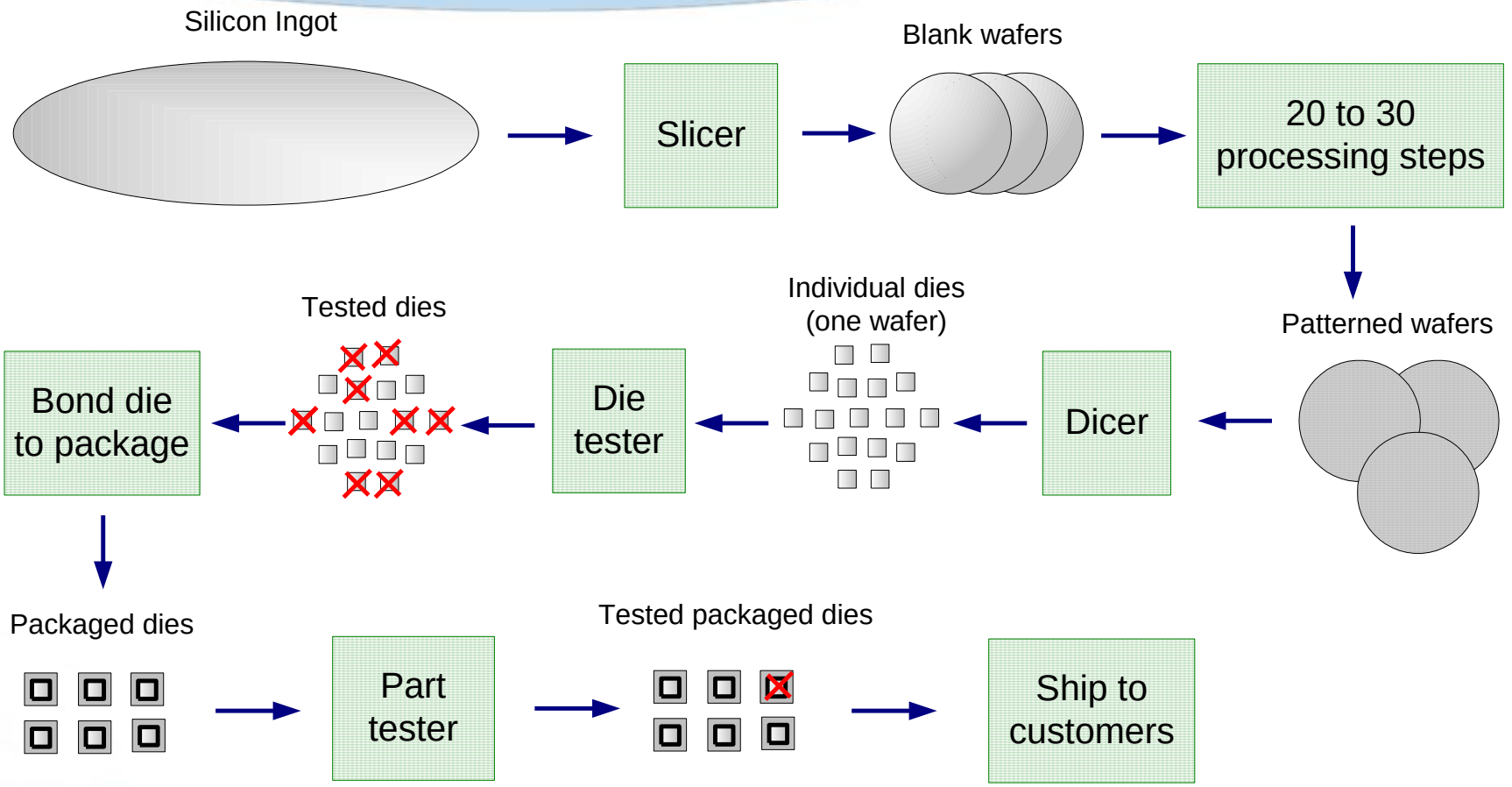
Understanding Cost Trend by Learning Curve



Rules of Thumb

- Time: learning curve ----yield
  - Twice the yield will have half the cost.
- Volume:
  - Cost decrease about 10% for each doubling of volume.
- Commodities:
  - Vendor competition
  - Supplier competition
  - Volume increase, however limited profits.

Microelectronics Process



Cost of an **Integrated Circuit**

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

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

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

$$\text{Die yield} = \text{Wafer yield} \times \left(1 + \frac{\text{Defects per unit area} \times \text{Die area}}{\alpha}\right)^{-\alpha}$$

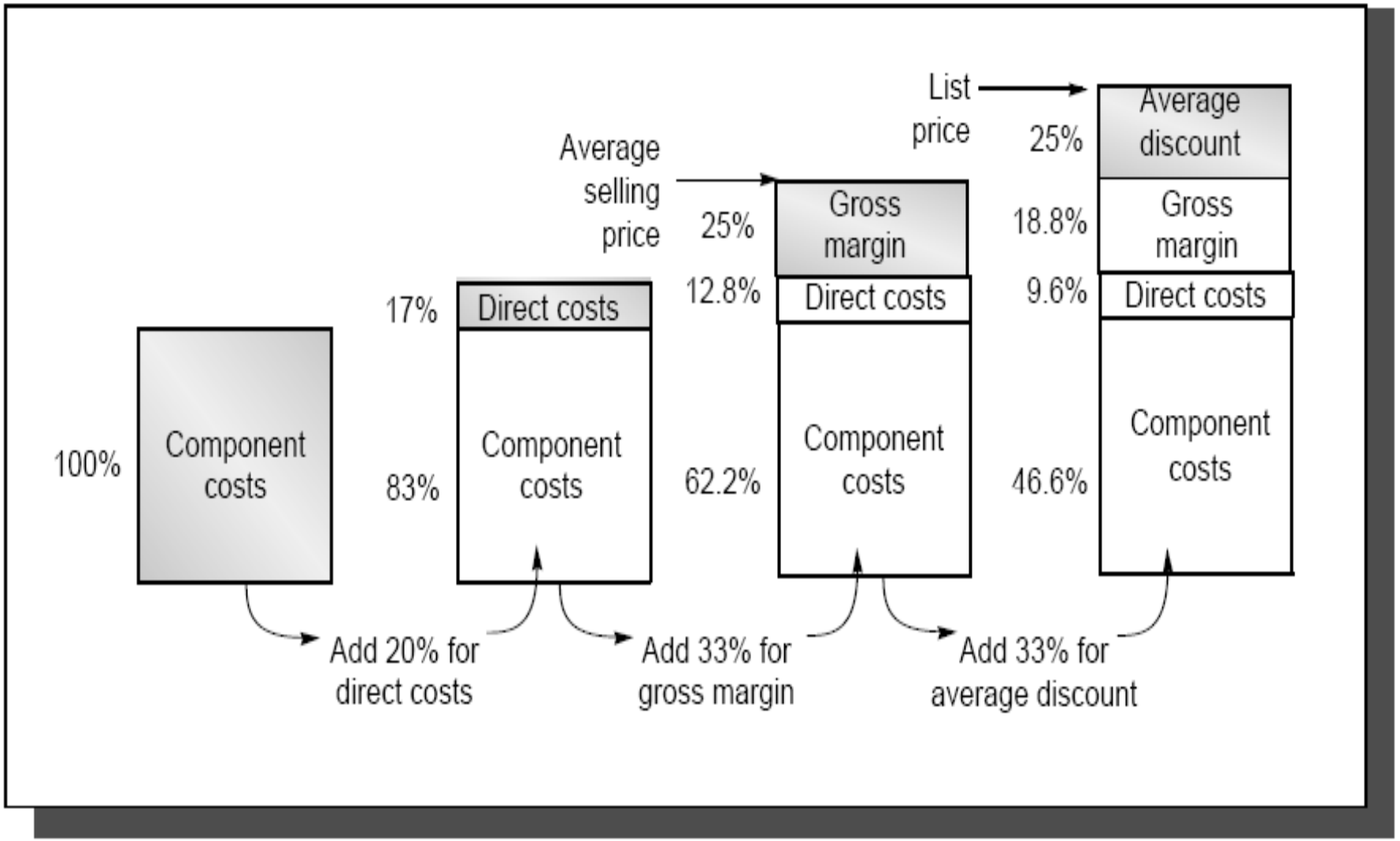
Distribution of Cost in a System

System	Subsystem	Fraction of total
Cabinet	Sheet metal, plastic	2%
	Power supply, fans	2%
	Cables, nuts, bolts	1%
	Shipping box, manuals	1%
	Subtotal	**6%**
Processor board	Processor	22%
	DRAM (128 MB)	5%
	Video card	5%
	Motherboard with basic I/O support, networking	5%
	Subtotal	**37%**
I/O devices	Keyboard and mouse	3%
	Monitor	19%
	Hard disk (20 GB)	9%
	DVD drive	6%
	Subtotal	**37%**
Software	OS + Basic Office Suite	20%

Cost vs. Price

- Component costs
  - **Raw material cost.**
- Direct cost:
  - **Costs incurred to make a single item. Adds 20% to 40% to component cost.**
- Gross margin (Indirect cost):
  - **Overhead not associated with a single item, i.e. R&D, marketing, manufacturing equipment, taxes, etc.**
  - **Only 4%-12% of income are spent on R&D**
- Average Selling Price (ASP):
  - **Component cost + direct cost + indirect cost.**
- List price :
  - **Not ASP. Stores add to the ASP to get their cut. Want 50% to 75% of list price.**

The components of price for a $1000 PC



Cost vs. Price

- This gives you insight on how a design decision will affect selling price,
  - i.e. changing cost by $1,000 increases selling price by $3,000 to $4,000.
- Also, consider volume and price relationship:
  - In general, the fewer computers that are sold, the higher the price.
  - Also, a decrease in volume causes cost to increase, further increasing price.
- Therefore, small changes in cost can have an unexpected large increase in price.

Topics in Chapter

1.1 Why take this course ?

1.2 Classes of computers in current computer market

1.3 Defining computer architecture and What's the task of computer design?

1.4 Trends in Technology

1.5 Trends in power in Integrated circuits

1.6 Trends in Cost

1.7 Dependability

1.8 Measuring, Reporting and summerizing Perf.

1.9 Quantitative Principles of computer Design

1.10 Putting it altogether

Dependability

- Dependability is a deliberately broad term to encompass many facets including reliability, security and availability.

Dependability vs. Reliability

- **Dependability**. A measure of the degree to which an item is operable and capable of performing its required function at any (random) time during a specified mission profile, given item availability at the start of the mission.
- its use is restricted to general descriptions in non-quantitative terms.
- Dependability is related to reliability; the intention was that dependability would be a more general concept then reliability.

Measurements of Dependability

- Module reliability: continuous service accomplishment
  - MTTF: Mean Time To Failure
  - MTTR: Mean Time To Repair
  - FIT : Failure In Time = 1/MTTF
  - MTBF: Mean Time Between Failure = MTTF+MTTR
- Module availability
  - $\frac{MTTF}{MTTF + MTTR} = \frac{MTTF}{MTBF}$

Resolution to dependability

- Redundancy:
  - Time redundancy: repeat the operation again to see if it is still in erroneous.
  - Resource redundancy: have other components to take over from the one that failed.

Topics in Chapter

1.1 Why take this course ?

1.2 Classes of computers in current computer market

1.3 Defining computer architecture and What's the task of computer design?

1.4 Trends in Technology

1.5 Trends in power in Integrated circuits

1.6 Trends in Cost

1.7 Dependability

1.8 Measuring, Reporting and summerizing Perf.

1.9 Quantitative Principles of computer Design

1.10 Putting it altogether

performance

Measuring and Reporting Performance

- **Comparing Machines**
  - Execution time (latency)
  - Throughput
  - MIPS - millions of instructions per second
- **Comparing Machines Using Sets of Programs**
  - Choosing which program to evaluate performance
    - Benchmark Suites
  - Different Means: Arithmetic, Harmonic, and Geometric Means

Defining performance

- Performance means different things to different people, therefore its assessment is subtle

- Criteria of performance evaluation differs among users and designers

Perf. Metrics --response time

- **Wall-clock time**
  - Start the program and watch the clock -
  - when the program ends, that's the total wall-clock time
  - Also called response time or elapsed time or
  - Measures user perception of the system speed

- **Problems with wall-clock time**
  - What if more than one program is running on the same machine ?
  - What if the program asks for user input ?

Performance Metrics --CPU time

- Measures the time the CPU is computing, (not waiting for I/O)
  - Measures designer perception of the CPU speed
- CPU time is further divided into:
  - User CPU time - time spent in user mode
  - System CPU time - time spent in the operating system (OS)
- Unix time command reports CPU time as:
  - **90.7u 12.9s 2:39 65%**
  - **90.7 user CPU seconds (in the user's program)**
  - **12.9 system CPU seconds (in the system calls e.g. printf)**
  - **2 minutes, 39 seconds wall-clock time**
  - **65% of the wall clock time was spent running on the CPU**

Performance Metrics ----throughput

- **Amount of work done in a given time**
  - Measure administrator perception of the system perf.
- **We often use throughput to measure**
  - Number of lines of code per day
  - Number bits per second transmitted over a wire
  - Number of web pages served
- **In contrast to latency**
  - amount of time to produce 1 line of code
  - amount of time to send 1 bit over a wire
  - Amount of time spent waiting to receive web page
- **Often, processor performance is only quoted in terms of relative latency**
  - Program A ran 10 times faster than program B
- **But, for many apps, throughput much more important than latency**
  - Financial markets, government statistics (census)

Response time vs. Throughput

- If you improve response time, you usually improve throughput
  - Replacing the processor of a computer with a faster version
- you can also improve throughput without improving response time
  - Adding additional processors to a system that uses multiple processors
  - for separate tasks (e.g. handling of airline reservations system)

Another industry Metric: **MIPS**

- MIPS - Millions of Instructions per Second

$$\text{MIPS} = \frac{\frac{\#\text{ of instructions}}{\text{benchmark}} \times \frac{\text{benchmark}}{\text{total run time}}}{1{,}000{,}000}$$

- When comparing two machines (A, B) with the same instruction set, MIPS is a fair comparison(*sometimes…*)
- But, MIPS can be a "***meaningless*** *indicator of performance…*"

Example: MIPS might be meaningles

- Machine A has a special instruction for performing square root calculations. It takes 100 cycles to execute.
- Machine B doesn't have the special instruction -- must perform square root calculations in software using simple instructions (.e.g, Add, Mult, Shift) that each take 1 cycle to execute
- Machine A: 1/100 MIPS = 0.01 MIPS
- Machine B: 1 MIPS

Another view: Power consumption and Efficiency

- Critical factors for embedded systems:
  - cost
  - physical size
  - memory
  - power consumption
- Fig. 1.27 (old versioin)
  - **AMD ElanSC520**
  - **AMD K6-2E**
  - **IBM PowerPC 750CX**
  - **NEC VR 5432**
  - **NEC VR 4122**

The NEC VR 4122 is the big winner for its best performance/watt, though it is the second lowest performing processor.

Summary of performance metrics

- Response (Execution) time
  - user perception
  - system performance
  - the only unimpeachable measure of performance
- CPU time
  - designer perception
  - CPU performance
- Throughput
  - administrator perception
- MIPS
  - merchant perception

Choose Programs to Evaluate Performance

- Ideal performance evaluation:
  - A random sample of users running their programs and OS commands.
- Many different types of benchmarks
  - **Real applications**--- Scientific and engineering
  - **Modified (or scripted) applications**--- focus on specific features
  - **Kernels** --- critical program fragments
  - **Toy** --- small programs, often measure very little
  - **Synthetic** -- created to represent some aspects of a program (e.g., mix of instruction types)
  - Database -- a world unto itself
  - What really matters is how YOUR application performs

Something about Synthetic

- Synthetic benchmarks :
  - Programs that try to "exercise" the system in the same way to match the average frequency of operations and operands of a large set of programs.
  - Whetstone and Dhrystone.
  - Similar to kernels but are NOT real programs !
  - Compiler and hardware optimizations can artificially inflate performance of these benchmarks but not of real programs.
  - These benchmarks don't reward optimizations!
  - $\mathrm{SQRT(EXP(x))} = \sqrt{e^{x}} = e^{x/2} = \mathrm{EXP(X/2)}$

Notes on performance benchmark

- Benchmarks can focus on specific aspects of a system
  - floating point & integer ALU, memory system, I/O, OS
- Universal benchmarks can be misleading since hardware and compiler vendors might optimize their design for ONLY these programs
- The best types of benchmarks are real applications since they reflect the end-user interest
- Architectures might perform well for some applications and poorly for others
- Compilation can boost performance by taking advantage of architecture-specific features. Application-specific compiler optimization are becoming more popular.

SPEC

- SPEC - The System Performance Evaluation Cooperative
  - founded in 1988 by a small number of workstation vendors who realized that the marketplace was in desperate need of realistic, standardize performance tests.
  - Grown to become successful performance standardization bodies with more than 40 member companies.
  - **http://www.spec.org**
- SPEC's Philosophy
  - The goal of SPEC is to ensure that the marketplace has a fair and useful set of metrics to differentiate candidate systems.
  - The basic SPEC methodology is to provide the benchmarker with a standardized suite of source code based upon existing applications

SPEC benchmarks

Desktop Benchmarks

- CPU-intensive benchmarks
  - SPEC89
  - SPEC92
  - SPEC95
  - SPEC2000
  - SPEC CPU2006 (12 CINT2006, 17 CFP2006)
- graphics-intensive benchmarks
  - SPEC2000
    - SPECviewperf
      - is used for benchmarking systems supporting the OpenGL graphics library
    - SPECapc
      - consists of applications that make extensive use of graphics.

SPEC INT 95 Benchmark descriptions

Benchmark	Ref Time (Sec)	Application Area	Specific Task
099.go	4600	Game playing; artificial intelligence	Plays the game Go against itself.
124.m88ksim	1900	Simulation	Simulates the Motorola 88100 processor running Dhrystone and a memory test program.
126.gcc	1700	Programming & compilation	Compiles pre-processed source into optimized SPARC assembly code.
129.compress	1800	Compression	Compresses large text files (about 16MB) using adaptive Limpel-Ziv coding.
130.li	1900	Language interpreter	Lisp interpreter.
132.ijpeg	2400	Imaging	Performs jpeg image compression with various parameters.
134.perl	1900	Shell interpreter	Performs text and numeric manipulations (anagrams/prime number factoring).
147.vortex	2700	Database	Builds and manipulates three interrelated databases.

SPEC FP 95 Benchmark Descriptions

Benchmark	Ref Time (Sec)	Application Area	Specific Task
101.tomcatv	3700	Fluid Dynamics / Geometric Translation	Generation of a two-dimensional boundary-fitted coordinate system around general geometric domains.
102.swim	8600	Weather Prediction	Solves shallow water equations using finite difference approximations. (The only single precision benchmark in CFP95.)
103.su2cor	1400	Quantum Physics	Masses of elementary particles are computed in the Quark-Gluon theory.
104.hydro2d	2400	Astrophysics	Hydrodynamical Navier Stokes equations are used to compute galactic jets.
107.mgrid	2500	Electromagnetism	Calculation of a 3D potential field.
110.applu	2200	Fluid Dynamics/Math	Solves matrix system with pivoting.
125.turb3d	4100	Simulation	Simulates turbulence in a cubic area.
141.apsi	2100	Weather Predication	Calculates statistics on temperature and pollutants in a grid.
145.fpppp	9600	Chemistry	Performs multi-electron derivatives.
146.wave	3000	Electromagnetics	Solve's Maxwell's eqn on cartesian mesh.

New SPEC Int 2000 Benchmarks

164.gzip	C	Compression
175.vpr	C	FPGA Circuit Placement and Routing
176.gcc	C	C Programming Language Compiler
181.mcf	C	Combinatorial Optimization
186.crafty	C	Game Playing: Chess
197.parser	C	Word Processing
252.eon	C++	Computer Visualization
253.perlbmk	C	PERL Programming Language
254.gap	C	Group Theory, Interpreter
255.vortex	C	Object-oriented Database
256.bzip2	C	Compression
300.twolf	C	Place and Route Simulator

New SPEC FP 2000 Benchmarks

168.wupwise	Fortran 77	Physics / Quantum Chromodynamics
171.swim	Fortran 77	Shallow Water Modeling
172.mgrid	Fortran 77	Multi-grid Solver: 3D Potential Field
173.applu	Fortran 77	Parabolic / Elliptic Partial Differential Equations
177.mesa	C	3-D Graphics Library
178.galgel	Fortran 90	Computational Fluid Dynamics
179.art	C	Image Recognition / Neural Networks
183.equake	C	Seismic Wave Propagation Simulation
187.facerec	Fortran 90	Image Processing: Face Recognition
188.ammp	C	Computational Chemistry
189.lucas	Fortran 90	Number Theory / Primality Testing
191.fma3d	Fortran 90	Finite-element Crash Simulation
200.sixtrack	Fortran 77	High Energy Nuclear Physics Accelerator Design
301.apsi	Fortran 77	Meteorology: Pollutant Distribution

SPEC benchmarks

Server Benchmarks

- SPECrate--processing rate of a multiprocessor
  - SPEC CPU2000—throughput-oriented benchmark
  - SPECrate—processing rate of a multiprocessor
  - SPECSFS--file server benchmark
  - SPECWeb--Web server benchmark
  - Transaction-processing (TP) benchmarks
  - TPC benchmark—Transaction Processing Council
    - TPC-A, 1985
    - TPC-C, 1992,
    - TPC-H→ TPC-R→TPC-W

SPEC benchmarks

Embedded Benchmarks

- EDN Embedded Microprocessor Benchmark Consortium (or EEMBC, pronounced "embassy").

Benchmark type	Number of kernels	Example benchmarks
Automotive/industrial	16	6 microbenchmarks (arithmetic operations, pointer chasing, memory performance, matrix arithmetic, table lookup, bit manipulation), 5 automobile control benchmarks, and 5 filter or FFT benchmarks
Consumer	5	5 multimedia benchmarks (JPEG compress/decompress, filtering, and RGB conversions)
Networking	3	Shortest-path calculation, IP routing, and packet flow operations
Office automation	4	Graphics and text benchmarks (Bézier curve calculation, dithering, image rotation, text processing)
Telecommunications	6	Filtering and DSP benchmarks (autocorrelation, FFT, decoder, encoder)

Running Benchmarks

- Key factor: Reproducibility by other experimenters.
- Details, details, and more details !!! List all assumptions and conditions of your experiments.
  - i.e. program input, version of the program, version of the compiler, optimization level, OS version, main memory size, disk types, etc.
- A system's software configuration can significantly affect the performance results for a benchmark.

Comparing Two Machines

Machine	CPI	Clock Period	Avg Instruction Time (secs)
Machine A	1.2	2 ns	
Machine B	2.5	1 ns	

- CPU Time = # of instructions executed * avg instruction time
- Assume 1,000,000, 000 instructions
- Machine A: 1,000,000,000 * 2.4ns = 2.4 seconds
- Machine B: 1,000,000,000 * 2.5ns = 2.5 seconds
- Which machine is faster? Machine A
- How much faster? 2.5 / 2.4 = 1.04 times faster

Comparing Performance

- Often, we want to compare the performance of different machines or different programs. Why?
  - To help engineers understand which is "better"
  - To give marketing a "silver bullet" for the press release
  - To help customers understand why they should buy <my machine>
-
- Performance and Execution time are *reciprocals*

  Maximizing performance means minimizing response (execution) time

$$\text{Performance} = \frac{1}{\text{Execution Time}}$$

Common used phrases

- "Performance of P_1 is better than P_2" is, for a given work load L, P_1 takes less time to execute L than P_2 does

  performance(P1) > Performance(P2)

  $\Rightarrow$ Execution Time(P1, L) < Execution Time(P1, L)

- "Processor X is n times fast than Y" is

$$n = \frac{\text{Execution time}_Y}{\text{Execution time}_X}$$

Comparing Performance Across Multiple Programs

	Computer A	Computer B	Computer C
Program 1 (secs)	1	10	20
Program 2 (secs)	1000	100	20
Program 3 (secs)	1001	110	40

- A is 10 times faster than B for program 1
- B is 10 times faster than A for program 2
- A is 20 times faster than C for program 1
- C is 50 times faster than A for program 2
- B is 2 times faster than C for program 1
- C is 5 times faster than B for program 2

- Each statement above is correct…,

…but we want to know which machine is the best?

Let's Try a Simpler Example

- **Two machines timed on two benchmarks**
  - How much faster is Machine A than Machine B?

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

- Attempt 1: ratio of run times, normalized to Machine A times
  - **program1: 4/2 program2 : 8/12**
  - **Machine A ran 2 times faster on program 1, 2/3 times faster on program 2**
  - **On *average*, Machine A is (2 + 2/3) /2 = 4/3 times faster than Machine B**
- **It turns this "averaging" stuff can fool us**

Example: Second answer

- **Two machines timed on two benchmarks**
  - How much faster is Machine A than Machine B?

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

- Attempt 2: **ratio of run times, normalized to Machine B times**
  - **program 1: 2/4 program 2 : 12/8**
  - **Machine A ran program 1 in 1/2 the time and program 2 in 3/2 the time**
  - **On average, (1/2 + 3/2) / 2 = 1**
  - **Put another way, Machine A is 1.0 times faster than Machine B**

Example: Third answer

- **Two machines timed on two benchmarks**
  - How much faster is Machine A than Machine B?

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

- Attempt 3: **ratio of run times, aggregate (total sum) times,**
  - **Machine A took 14 seconds for both programs**
  - **Machine B took 12 seconds for both programs**
  - **Therefore, Machine A takes 14/12 of the time of Machine B**
  - **Put another way, Machine A is 6/7 faster than Machine B**

Which is Right?

- **Question:**
  - How can we get three different answers?
- **Solution**
  - Because, while they are all reasonable calculations…
  - …each answers a *different* question
- We need to be more precise in understanding and posing these performance & metric questions

Arithmetic and Harmonic Mean

- Total Execution Time: A Consistent Summary Measure
  - **Arithmetic mean** is the average of the execution time that tracks total execution time.

$$\frac{1}{n}\sum_{i=1}^{n} Time_i$$

  - If performance is expressed as a **rate**, then the average that tracks total execution time is the **harmonic mean**

$$\frac{n}{\sum_{i=1}^{n}\frac{1}{Rate_i}}$$

Problems with Arithmetic Mean

- Applications do not have the same probability of being run
- Longer programs weigh more heavily in the average
- For example, two machines timed on two benchmarks

	Machine A	Machine B
Program 1	2 seconds (20%)	4 seconds (20%)
Program 2	12 seconds (80%)	8 seconds (80%)

- If we do arithmetic mean, Program 2 "counts more" than Program 1
  - an improvement in Program 2 changes the average more than a proportional improvement in Program 1
- But perhaps Program 2 is 4 times more likely to run than Program 1

Weighted Execution Time

- Often, one runs some programs more often than others. Therefore, we should weight the more frequently used programs' execution time

$$\sum_{i=1}^{n} Weight_i \times Time_i$$

- Weighted Harmonic Mean

$$\frac{1}{\sum_{i=1}^{n} \frac{Weight_i}{Rate_i}}$$

Using a Weighted Sum (or weighted average)

	Machine A	Machine B
Program 1	2 seconds (20%)	4 seconds (20%)
Program 2	12 seconds (80%)	8 seconds (80%)
Total	10 seconds	7.2 seconds

Allows us to determine relative performance 10/7.2 = 1.38

--> Machine B is 1.38 times faster than Machine A

Another Solution

- Normalize run time of each program to a reference

	Machine A (ref)	Machine B
Program 1	2 seconds	4 seconds
Program 2	12 seconds	8 seconds
Total	10 seconds	7.2 seconds

	Machine A (norm to B)	Machine B (norm to A)
Program 1	0.5	2.0
Program 2	1.5	0.666
Average?	1.0	1.333

- So when we normalize A to B, and average, it looks like A & B are the same.
- But when we normalize B to A, it looks like B is 33% better!

Example on P37(old version)

	Programs			Weightings		
	A	B	C	W(1)	W(2)	W(3)
Program P1 (secs)	1.00	10.00	20.00	0.50	0.909	0.999
Program P2 (secs)	1000.00	100.00	20.00	0.50	0.091	0.001
Arithmetic mean: W(1)	500.50	55.00	20.00			
Arithmetic mean: W(2)	91.91	18.19	20.00			
Arithmetic mean: W(3)	2.00	10.09	20.00			

$$W(B)_1 = \frac{1}{10 \times (1/10 + 1/100)} = 0.909$$

$$W(B)2 = \frac{1}{100 \times (1/10 + 1/100)} = 0.091$$

$$w_i = \frac{1}{\text{Time}_i \times \sum_{i=1}^{n} \left(\frac{1}{\text{Time}_i} \right)}$$

Geometric Mean

- Used for relative rate or performance numbers

$$Relative\_Rate = \frac{Rate}{Rate_{ref}} = \frac{Time_{ref}}{Time}$$

- Geometric mean

$$\sqrt[n]{\prod_{i=1}^{n} Relative\_Rate_i} = \frac{\sqrt[n]{\prod_{i=1}^{n} Rate_i}}{Rate_{ref}}$$

Using Geometric Mean

	Machine A (norm to B)	Machine B (norm to A)
Program 1	0.5	2.0
Program 2	1.5	0.666
Geometric Mean	0.866	1.155

1.155 = 1/0.8666!

- Drawback:
  - Geometric mean does NOT predict run time because it automatically
- normalizes.
  - Each application now counts equally.
  - Irrelevance of the reference computer in relative performance

Summary of comparing performance

- Total execution time or arithmetic mean
  - consistent result
  - programs in the workload are NOT always run an equal number of times
- Weighted arithmetic mean
  - take into account the frequency of use in the workload
  - solution depends on which machine is the reference.
- Normalized Geometric Mean
  - consistent result, no matter which machine is the reference.
  - Geometric mean does NOT predict run time
- Ideal solution : Measure a real workload and weight the programs according to their frequency of execution.
- **What really matters is how YOUR application performs**

New SPEC Performance Numbers

- Geometric Mean of 12 (SpecInt) and 14 (SpecFP) Benchmarks
  - Performance measured against SPARC 10/40
- 2000 Performance Numbers (Microprocessor Report, Dec. 2000)

	Alpha 21264B 833MHz	Intel PentiumIII 1GHz	MIPS R12000 400MHz	HP PA-8600 552MHz	IBM Power 3-II 450MHz	Sun Ultra III 900MHz
Int	518	454	320	417	286	438
FP	590	329	319	400	356	369

New SPEC Performance Numbers

- Geometric Mean of 12 (SpecInt) and 14 (SpecFP) Benchmarks
  - Performance measured against SPARC 10/40
- 2001 Performance Numbers (Microprocessor Report, Aug. 2001)

	Alpha 21264C 1001MHz	Intel P4 1.8GHz	MIPS R14000 500MHz	HP PA-8600 552MHz	IBM Power 3-II 450MHz	Sun Ultra III 900MHz
Int	561	599	397	417	286	439
FP	585	615	362	400	356	369

Topics in Chapter

1.1 Why take this course ?

1.2 Classes of computers in current computer market

1.3 Defining computer architecture and What's the task of computer design?

1.4 Trends in Technology

1.5 Trends in power in Integrated circuits

1.6 Trends in Cost

1.7 Dependability

1.8 Measuring, Reporting and summerizing Perf.

1.9 Quantitative Principles of computer Design

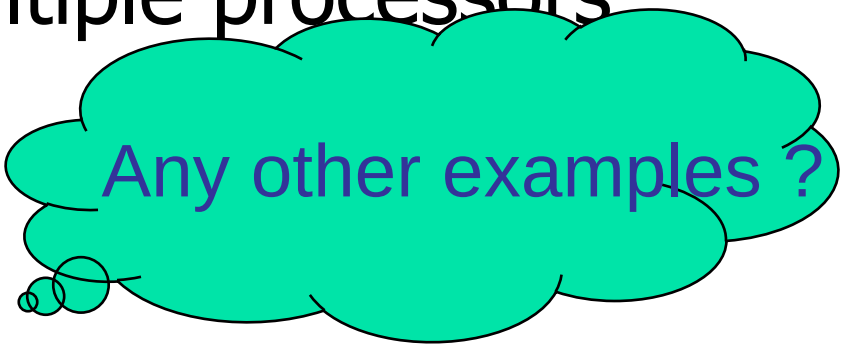
1.10 Putting it altogether

1.9 Quantitative Principles

- Take advantage of parallelism
- Principle of Locality
- Focus on the common case
- Amdahl's Law
- CPU Performance Equation

Take advantage of parallelism

- Most important methods of improving performance
- Parallelism levels
  - System level: use multiple processors
  - Instruction level:
    - Pipelining
  - Operation level:
    - set-associate cache
    - Pipelined function unit



Principle of Locality

- Program Property: Programs tend to reuse data and instructions they have used recently.
- Rule of thumb:
  - a program spends 90% of its execution time in only 10% of the code
- Tempor
  - Recently accessed items are likely to be accessed in the near future.
- Spatial locality
  - Items whose addresses are near one another tend to be referenced close together in time.

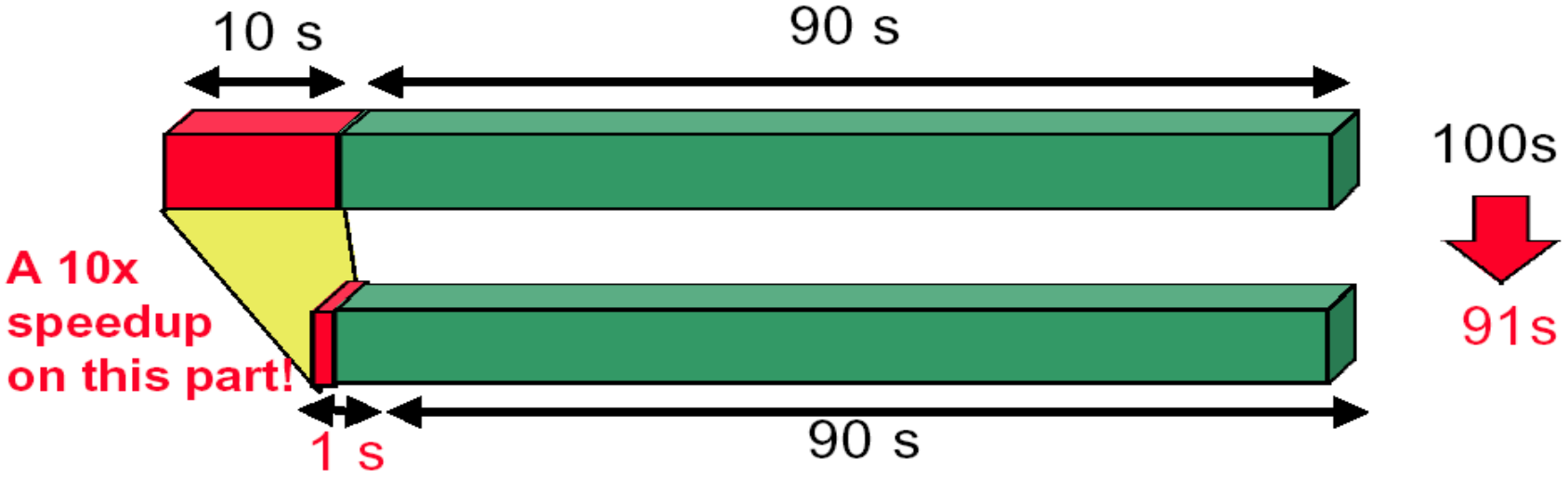
Any example ?

Focus on the common case

- The most important and pervasive principle of computer design.
  - Power, resource allocation, performance, dependability.
  - Rule of thumb: simple is fast.
  - Frequent case is often simpler and can be done faster.
- A fundamental law, called *Amdahl's Law*, can be used to quantify this principle.

Amdahl's Law

- The performance improvement to be gained from using some faster mode of execution is limited by the fraction of the time the faster mode can be used.
- Example



Amdahl's law

$$\text{Execution time after imrpovement} = \frac{\text{Execution time affected by the improvement}}{\text{Amount of improvement}} + \text{Execution time unaffected}$$

- Increasing the clock rate would not affect memory access time
- Using a floating point processing unit does not speed integer ALU operations

Amdahl's law

- Amdahl's law defines the speedup

$$\text{Speedup} = \frac{\text{Performance with enhancement}}{\text{Performance without enhancement}} = \frac{\text{Execution time w/o enhancement}}{\text{Execution time with enhancement}}$$

- If we know two factors:
  - **Fraction enhanced : Fraction of computation time in original machine that can be converted to take advantage of the enhancement.**
  - **Speedup enhanced in enhanced mode : Improvement gained by enhanced execution mode:**

$$\text{Exec time}_{new} = \text{Exec time}_{old} \times \left(\left(1 - Fraction_{enhanced}\right) + \frac{Fraction_{enhanced}}{Speedup_{enhanced}} \right)$$

Speedup Equation

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

- Example:
  - A server system with an enhanced CPU(10 times faster than the original one) used for Web serving. Assuming the original CPU is busy with computation 40% of the time and is waiting for I/O 60% of the time.
- Answer:
  - $Fraction_{enhanced} = 0.4$, $Speedup_{enhanced} = 10$
  - $Speedup = \frac{1}{0.6 + \frac{0.4}{10}} = \frac{1}{0.64} = 1.56$

Another Example

- Implementations of floating-point (FP) square root vary significantly in performance
- Two enhancement proposal
  - One proposal is to **enhance the FPSQR hardware** and speed up this operation by a factor of 10.
  - The alternative is just to try to make **all FP instructions** in the graphics processor run faster by a factor of 1.6;
- Assuming
  - FP square root (FPSQR) is responsible for **20%** of the execution time of a critical graphics benchmark.
  - FP instructions are responsible for a total of **50%** of the execution time for the application.
  - The design team believes that they do both enhancement with the same effort.
- Compare these two design alternatives.

Solution of the example

$$\text{Speedup}_{\text{PPSQR}} = \frac{1}{(1-0.2)+\frac{0.2}{10}} = \frac{1}{0.82} = 1.22$$

$$\text{Speedup}_{\text{PP}} = \frac{1}{(1-0.5)+\frac{0.5}{1.6}} = \frac{1}{0.8125} = 1.23$$

example

- Assume:
  - An enhancement to a compu that improves some of ex ctor of 10.
  - execution
    mode is in use.
- Question:
  - What is the speedup we have obtained from fast mode ?
  - What percentage of the original execution time has been converted of fast mode ?

Important Notes !

F is the fraction in original machine !

What the Amdahl's Law imply ?

- If an enhancement is only usable for a fraction of task, then the total speedup will be no more than 1/ (1-F).
- Serve the guide
  - **to how much an enhancement will improve performance**
  - **to how to distribute resource to improve cost-performance**
- Useful for comparing
  - **the overall system performance of two alternatives,**
  - **two CPU design alternatives**
- We can improve the performance by
  - **increasing the $Fraction_{enhanced}$**
  - **or, increasing the $Speedup_{enhanced}$**

The CPU Performance Equation

- The "Iron Law" of processor performance:
  - Often it is difficult to measure the improvement in time using a new enhancement directly.
- CPU Performance Equation

$$\text{CPU time} = \text{CPU clock cycles for a program} \times \text{Clock cycle time}$$

$$\text{CPU time} = \frac{\text{CPU clock cycles for a program}}{\text{Clock rate}}$$

Calculation of CPU Time

$$\text{CPU time} = \text{Instruction count} \times \text{CPI} \times \text{Clock cycle time}$$

Or

$$\text{CPU time} = \frac{\text{Instructio n count} \times \text{CPI}}{\text{Clock rate}}$$

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

Architecture --> Implementation --> Realization

Compiler Designer Processor Designer Chip Designer

Component of performance	Units of measure
CPU execution time for a program	Seconds for the program
Instruction count	Instructions executed for the program
Clock cycles per instructions (CPI)	Average number of clock cycles/instruction
Clock cycle time	Seconds per clock cycle

Related technologies

- CPU performance is dependent upon 3 characteristics:
  - clock cycle (or rate) (CCT)
  - clock cycles per instruction (CPI)
  - instruction count. (IC)

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

- One difficulty: It is difficult to change one in isolation of the others.

Other format of CPU Performance Equation

$$\text{CPU clock cycles} = \sum_{i=1}^{n} \text{IC}_i \times \text{CPI}_i$$

$$\text{CPU time} = \left(\sum_{i=1}^{n} \text{IC}_i \times \text{CPI}_i \right) \times \text{Clock cycle time}$$

$$\text{CPI} = \frac{\sum_{i=1}^{n} \text{IC}_i \times \text{CPI}_i}{\text{Instruction count}} = \sum_{i=1}^{n} \frac{\text{IC}_i}{\text{Instruction count}} \times \text{CPI}_i$$

Example of CPUtime calculation

- Suppose we have made the following measurements:
  - **Frequency of FP operations (other than FPSQR) = 25%**
  - **Average CPI of FP operations = 4.0**
  - **Average CPI of other instructions = 1.33**
  - **Frequency of FPSQR = 2%**
  - **CPI of FPSQR = 20**
- Two design alternatives
  - **decrease the CPI of FPSQR to 2**
  - **decrease the average CPI of all FP operations to 2.5.**
- Compare these two design alternatives using the CPU performance equation.

Answer to the question

$$\text{CPI}_{\text{original}} = \sum_{i=1}^{n} \text{CPI}_i \times \left(\frac{\text{IC}_i}{\text{Instruction count}} \right)$$

$$= (4 \times 25\%) + (1.33 \times 75\%) = 2.0$$

$$\text{CPI}_{\text{with new FPSQR}} = \text{CPI}_{\text{original}} - 2\% \times (\text{CPI}_{\text{old FPSQR}} - \text{CPI}_{\text{of new FPSQR only}})$$

$$= 2.0 - 2\% \times (20 - 2) = 1.64$$

$$\text{CPI}_{\text{new FP}} = (75\% \times 1.33) + (25\% \times 2.5) = 1.625$$

- Since the CPI of the overall FP enhancement is slightly lower, its performance will be marginally better.

Compare the result with that from Amdahl's law

- This is the same speedup we obtained using Amdahl's Law:

$$\text{Speedup}_{\text{new FP}} = \frac{\text{CPU time}_{\text{original}}}{\text{CPU time}_{\text{new FP}}} = \frac{\text{IC} \times \text{Clock cycle} \times \text{CPI}_{\text{original}}}{\text{IC} \times \text{Clock cycle} \times \text{CPI}_{\text{new FP}}}$$

$$= \frac{\text{CPI}_{\text{original}}}{\text{CPI}_{\text{new FP}}} = \frac{2.00}{1.625} = 1.23$$

Performance & price-performance

- Performance & price-performance for desktop systems Fig1.18
- Factors that responsible for the wide variation in price
  - Different levels of expandability
  - Use of cheaper disks and cheaper memory
  - Cost of CPU varies
  - Software differences
  - Lower-end system use PC commodity parts in fans, power supply, support chip sets
  - Commoditization effect

Five desktop and rack-mountable systems

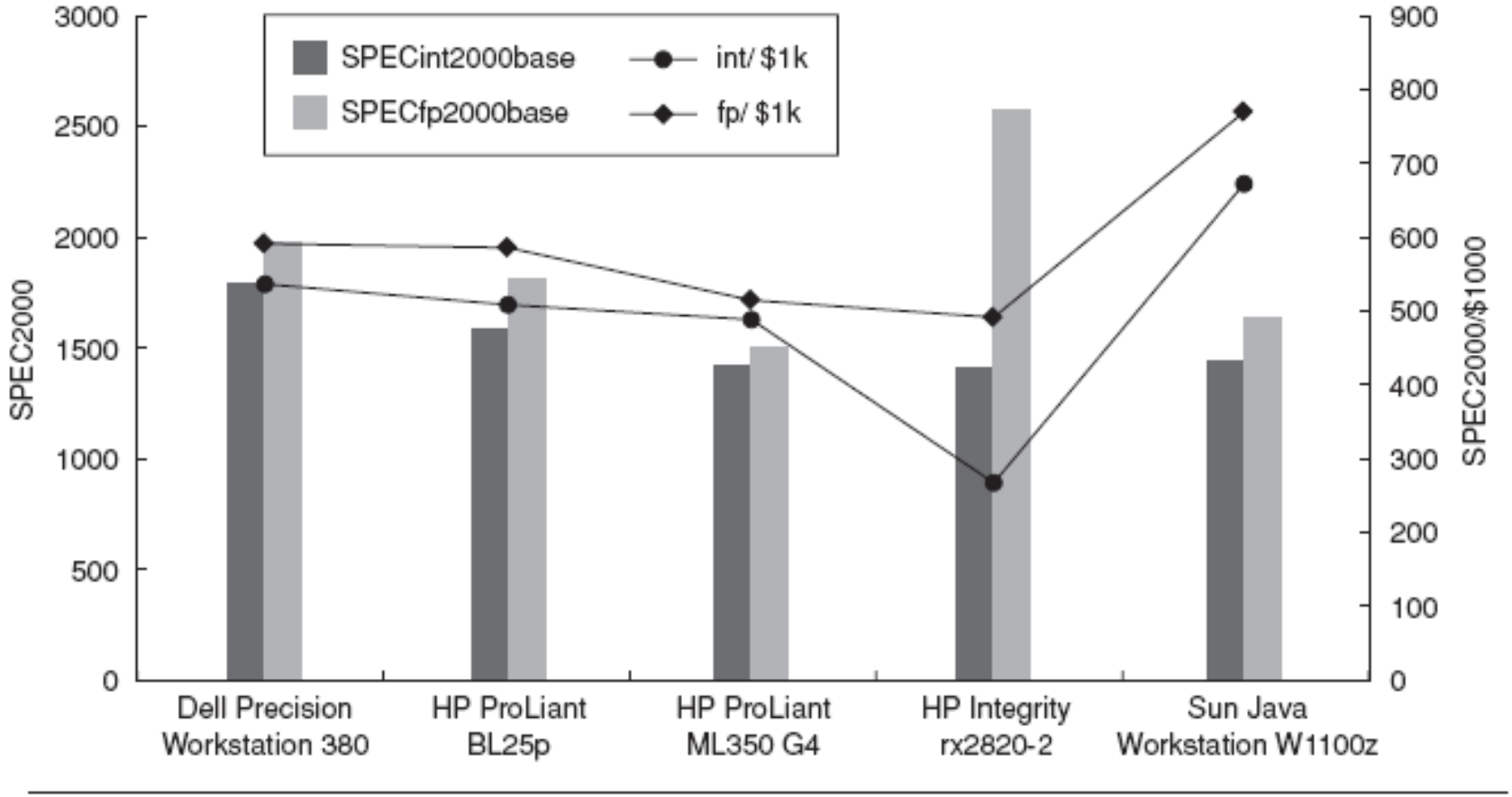
Vendor/model	Processor	Clock rate	L2 cache	Type	Price
Dell Precision Workstation 380	Intel Pentium 4 Xeon	3.8 GHz	2 MB	Desk	$3346
HP ProLiant BL25p	AMD Opteron 252	2.6 GHz	1 MB	Rack	$3099
HP ProLiant ML350 G4	Intel Pentium 4 Xeon	3.4 GHz	1 MB	Desk	$2907
HP Integrity rx2620-2	Itanium 2	1.6 GHz	3 MB	Rack	$5201
Sun Java Workstation W1100z	AMD Opteron 150	2.4 GHz	1 MB	Desk	$2145

Expandability: Sun Java worktation < Dell ….< HP BL25p

Cost of processor: die size and L2 cache , processor

Softerware difference

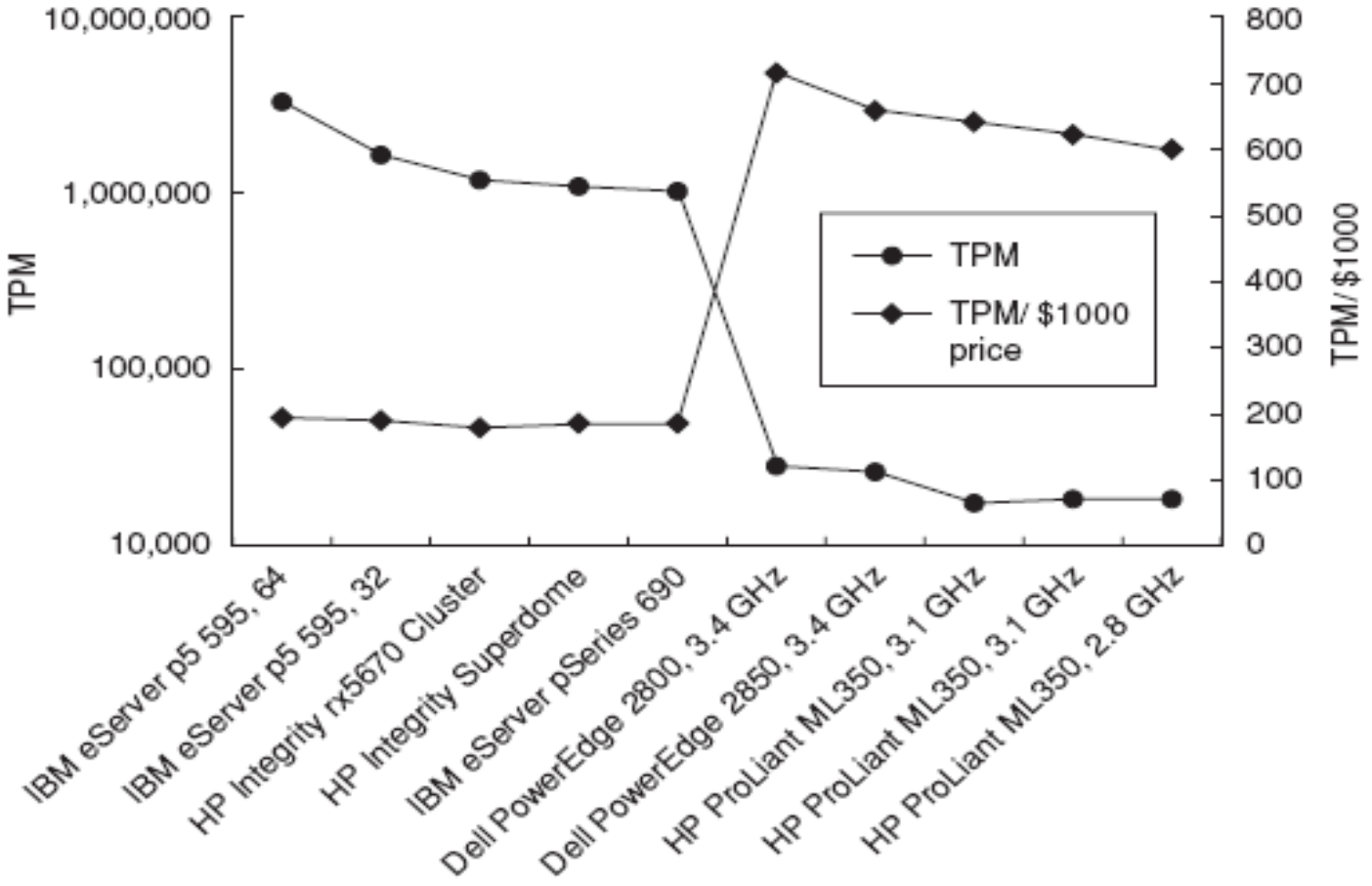
Price-performance



Messurements-1

- For Servers Fig 1.17, 1.18
  - TPC-C : standard industry benchmark for OLTP
    - Reasonable approximation
    - Measure total system performance
    - Rules of measurement are very complete
    - Vendors devote significant effort
    - Report both performance & price-performance

Price-performance



Fallacies & pitfalls

- Pitfall:
  - Falling prey to Amdahl's Law.
  - A single point of failure
  - Fault detection can lower availability

	Processor + cabinetry	Memory	Storage	Software
IBM eServer p5 595	28%	16%	51%	6%
IBM eServer p5 595	13%	31%	52%	4%
HP Integrity rx5670 Cluster	11%	22%	35%	33%
HP Integrity Superdome	33%	32%	15%	20%
IBM eServer pSeries 690	21%	24%	48%	7%
Median of high-performance computers	21%	24%	48%	7%
Dell PowerEdge 2800	6%	3%	80%	11%
Dell PowerEdge 2850	7%	3%	76%	14%
HP ProLiant ML350	5%	4%	70%	21%
HP ProLiant ML350	9%	8%	65%	19%
HP ProLiant ML350	8%	6%	65%	21%
Median of price-performance computers	7%	4%	70%	19%

- Fallacy
  - Benchmarks remain valid indefinitely
  - The rated mean time to failure of the disks is 1200000hours or almost 140 years, so disks practically never fail.
  - Peak performance tracks observed performance.

Homework for Chapter 1

- Read the section of 1.0
- Question:
  - You can select any 4 questions from textbook.
- Due time: Before the lecture begin
- write your answer in English.
- submit it to website, NOT via email.

再见，谢谢！