Introduction and Performance Measures

Prof. Yitzhak (Tsahi) Birk
Electrical Engr. Dept.
Technionm
CPU and memory are only two of many blocks!
Computer Categories

• **Desktop (PC, Workstation) - future?**
  - Optimized for single user
  - “DT/Workstation applications” → memory size, BW, disk space, footprint
  - Design goal: max performance for a given power/price

• **Servers**
  - Multiple users/applications
  - Heavy tasks → computing, throughput, memory footprint
  - Power limitations...
  - Reliability, availability (percent of up-time) critical
  - Throughput (and power) is the main performance measure

• **Notebook computers**
  - “Mobile Desktop” → related applications (“on the go”)
  - Battery life, low power (thin LT), compact,

• **Embedded computers**
  - Application specific, tuned, price and power sensitive, volume
  - System on Chip (SoC) – on die CPU core + custom hardware
  - Examples: mobile phone, PDA, playstation, set-top box, digital appliances

• **Ultra-Mobile Personal Computers (UMPC)**
  - Power envelope (no ventilation 😊)
  - User interface customization
The Role of Computer Architect

- **Requirement specification**
  - Constraints (must be satisfied)
  - Desires (optimization goals)
- **Study of prior solutions**
- **Option evaluation**
- **Conceptual design**
- **Architecture definition and other detailed design**
- **Tuning**

Design the optimal system that meets the constraints.
מטורות הקורס (לkerja והשיג נדרשים)

- הכרת חידיות המחשב ושיטות לテクונוג
- ריכישת כלים לחשיבות בינ yolופת
- כרorgetown ליביון מחששב וחברות
- דגש מיוהו יושם על פיתוחishiיבת ביכורהית
- יהלומי ניתות והשייכות למערכתית.
- מחצל הכרת החידיות הבידיות והבנת פעולות,
- הסתונט נידרש לحادי את הקשים והשעועות
- ההדידיות.
Course Goals

• Know and understand the main components of a computer system and the considerations in their design.
• Acquire tools for comparison among alternatives.
• Know and understand performance measures.
• Understand the interplay among system components, design trade-offs, etc.
• Develop critical “system” thinking and reasoning ability
Computer Architecture - Course Outline

• Fundamental of Computer Systems
  – Computer Architecture scope
  – Technology Trends
  – Performance measures and implications

• Computer System’s components
  – The computing ensemble
  – Central Processing Element:
    » Instruction supply, engine, data supply
  – Memory subsystem
  – Human interface

• Memory hierarchy
  – Caches
  – Virtual memory
  – Main memory
  – Optimizations

• Instruction supply
  – Branch prediction
  – Speculation

• Data supply
  – Data speculation

• The Engine
  – Instruction level parallelism
  – Engine structure: Pipeline, Superscalar

• Multithreading

• Multi-core and implications

• Input-Output (I/O) system
  – Devices
  – Architecture
  – Communication

• Related topics
  – operating systems
  – compilers
COMPUTER SYSTEMS COURSES

• **Computer Architecture**
  – “Limbs”: execution
  – “Reflexes”:
    » “Shallow thinking”
    » Fast

• **Operating Systems**
  – “Brain”
    » “Deep thinking”
    » Slow

• **Tools for analyzing computer systems**
  – Closing the feedback loop

*Anatomy + Physiology!*
Course administration

• Lecturer: Prof. Yitzhak (Tsahi) Birk  birk@ee.technion.ac.il
• Assistant: Zvika Guz zguz@tx.technion.ac.il
• Homework: every week
• Final grade: Final exam: 60%; Midterm: 25%*; Homework: 15%*.
  "Magen", but HW counts only if exam grade $\geq 55$; unused "weight" goes to final exam. Details in separate document.
• Be on your tiptoes (lots of observations and board information)
• Course site:  www.ee.technion.ac.il/courses/046267
Computer Architecture

046267
Today’s topics

• Course Goals
• Course structure
• Course administration

• Fundamentals of Computer architecture
  – The computer
  – Trends
  – Definitions
  – Quantitative approach
Architecture components

• **Instruction Set Architecture (ISA)**
  – The computer basic command
  – Enable performing/efficient implementation

• **Compilers**
  – Generate optimized code sequences

• **Microarchitecture**
  – Implementation of the architecture

• **Logic & Circuit**
  – Implementation of the design

• **Process Technology**
  – The Technology that the logic/circuit is implemented on
Architecture & Microarchitecture

• **Architecture:**
The collection of processor’s (or a system’s) features as they are (SW) seen by the “user”
  – binary executable running on that processor, or
  – assembly level programmer

• **MicroArchitecture:**
The collection of features or ways of implementing a processor (or a system), usually does not affect the user

• **Timing/performance** are considered microarchitecture.

• Also other (non-CPU) issues like I/O
Architecture & Microarchitecture Elements

- **Architecture examples:**
  - Registers data width (8/16/32)
  - Instruction set
  - Addressing modes
  - Addressing methods (Segmentation, Paging, etc...)
  - Protection scheme
  - Bus and bus protocol
  - TLB structure and policy

- **μArchitecture examples:**
  - Timing is μArchitecture (*though it is user visible!*)
  - Execution Pipelines, Number & type execution units
  - Branch prediction
  - Caches & Cache size
  - TLB size
  - Physical memory size
  - Bus width
  - Machine width
The Computer

Computation and “scratch pad”

- CPUs & Caches
- Computation engines (e.g., Graphics engine, Media, Communication engines)
- Memory
- Mass storage

<table>
<thead>
<tr>
<th>The “outside” world</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fast I/O</td>
</tr>
<tr>
<td>e.g., camera, Accelerators</td>
</tr>
<tr>
<td>Slow I/O</td>
</tr>
<tr>
<td>e.g., human interface</td>
</tr>
<tr>
<td>Display</td>
</tr>
<tr>
<td>Communication</td>
</tr>
</tbody>
</table>
Computer system

• How to connect between the elements
  – Bus
  – Point-to-point
  – Packets…

• Integration trend
  – CPU, CPU + $, CPU + $ + MC, CPU + $ + MC + Graphic P, CPU + + +
  – Consequences - MC, NB, G, Media, Communication…

• Platform implications
COMPUTER SYSTEM COMPONENTS

Yesterday

CPU
  cache

North Bridge

South Bridge
  printer scanner
  keyboard mouse...

Disk

MAIN MEMORY
  Network +WLAN

I/O CONTROLLERS
  ● ● ●
COMPUTER SYSTEM COMPONENTS

CPU
MC+cache+G

South Bridge

printer scanner
keyboard mouse...

Disk

MAIN MEMORY

Network +WLAN
Computer Architecture design process

• Environment:
  – Define optimization targets
  – Set constraints (things that are absolutely required)

• Define an optimal system that is optimized to meet the constraints

• Computer Architecture:
  – Instruction Set Architecture (ISA)
  – Computer organization (system architecture: e.g. memory sub-system, communication, standard interfaces, CPU...)
  – Hardware – the next design level (specifics, partitioning, logic design, packaging, etc.)
Context for Designing New Architectures

- **Application Area**
  - Special Purpose (e.g., DSP) / General Purpose
  - Scientific (FP intensive) / Commercial (Mainframe)

- **Level of Software Compatibility**
  - Object Code/Binary Compatible (cost HW vs. SW; IBM S/360)
  - Assembly Language (dream to be different from binary)
  - Programming Language; Why not?

- **Operating System Requirements for General Purpose Applications**
  - Size of Address Space
  - Memory Management/Protection
  - Context Switch
  - Interrupts and Traps
  - Multi-tasking, threads

- **Standards: Innovation vs. Competition**
  - IEEE 754 Floating Point
  - I/O Bus
  - Networks
  - Operating Systems / Programming Languages ...

- **Constraints**
- **Optimization targets**
Metrics

Performance Relative to a specific machine, Answers per unit time
ISA Million Instruction Per Second (MIPS)
Instruction per Cycles (IPC = 1/CPI)
Frequency Cycles per second (e.g. Mhz, Ghz)
Communication bits/second (e.g. 1Gb/s)
Memory Bytes (e.g. MB, GB)
Power envelope Watts
VLSI Feature Size Length (um, nm)

\[
\text{Performance} = \text{Frequency} \times \text{IPC}
\]
Technology Trends

- Feature size
- Transistors on a chip
- Performance
- Die size
- Frequency
- Power
- BW vs. Latency

Moore’s law: number of transistors will double every 18 months
Transistors on a Chip

2X growth in 1.96 years!

Transistors on a chip doubled every two years
Microprocessor Performance Trends

Number of transistors:
• Intel core 2 Duo: ~300M
• Pentium III – ~30 million
Die Size Growth

Die side (mm) = $\sqrt{\text{Area}}$

Die size grows? Is it saturated?

Year

Frequency

Lead Microprocessor frequency used to be doubled every 2 years; Not any more…
Lead Microprocessors power increased exponentially but the trend has changed…
Following Moore’s Law
BW vs. Latency Improvements (~1980-2005)

H&P4e

Fig. 1.18
Micropororessor Perf. Trends

Number of transistors:
- Intel core 2 Duo: ~300M
- Pentium III – ~30 million

CMOS improvements:
- Die size: 2X every 3 yrs
- Line width: halve / 7 yrs

Architectural trends:
- Chip Multi-Processing (CMP)
- Network on Chip (NOC)
## Hardware Technology (History)

<table>
<thead>
<tr>
<th></th>
<th>1980</th>
<th>1990</th>
<th>2006</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory chips</td>
<td>64 K</td>
<td>4 M</td>
<td>256 M-1 G</td>
</tr>
<tr>
<td>Speed</td>
<td>1-2</td>
<td>20-40</td>
<td>400-1000</td>
</tr>
<tr>
<td>Disks</td>
<td>40 M (5.25”)</td>
<td>1 GB</td>
<td>750 GB (3.5”)</td>
</tr>
<tr>
<td>Floppies</td>
<td>.256 M</td>
<td>1.5 M</td>
<td>500-2,000 M</td>
</tr>
<tr>
<td>LAN</td>
<td>2-10 Mb/s</td>
<td>10 (100)</td>
<td>1-10 Gb/s Ethernet</td>
</tr>
<tr>
<td>Busses</td>
<td>2-20 MB/s</td>
<td>40-400 (PCI)</td>
<td>5000 (PCI-Xpress)</td>
</tr>
<tr>
<td>I/O interconnects</td>
<td>various “channels”</td>
<td>30Gb/s (Infiniband)</td>
<td></td>
</tr>
<tr>
<td>On-chip interconnects</td>
<td>Bus</td>
<td>Network on chip</td>
<td></td>
</tr>
</tbody>
</table>
# Technology Trends (Historical Summary)

<table>
<thead>
<tr>
<th></th>
<th>Capacity</th>
<th>Speed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logic</td>
<td>2x in 3 years</td>
<td>2x in 3 years</td>
</tr>
<tr>
<td>DRAM</td>
<td>4x in 3 years</td>
<td>1.4x in 10 years</td>
</tr>
<tr>
<td>Disk</td>
<td>2x in 3 years</td>
<td>1.4x in 10 years</td>
</tr>
</tbody>
</table>

Leads to:

**Computing capacity:** 4x per 3 years

- If can keep all the transistors busy all the time
- Actual: 3.3x per 3 years
Trends - Summary

- **Technology**
  - Density keeps improving
  - Clock rate increases at a slower pace
  - Latency is not keeping up

- **Parallelism**
  - Further increasing the exploitation of Instruction-level parallelism (ILP) using pipeline, out-of-order execution, etc. has become inefficient
  - Thread multi-processing and chip multiprocessors (“multi-core”) are here, presenting a dramatic software challenge

- **Complete systems on a chip**
  - 100+ Million Transistors
  - Integration is key to cost and power reduction + performance.

- **Power is a key issue**
  - Heat dissipation:
    - Reached the current die power envelope (chip level)
    - Reached the Watts/m² and Watts/m³ limits (heat removal from boxes and racks)
  - Total power consumption:
    - Computing power often limited by available Watts: computer + air conditioning
    - Cost of power often dominates the total cost of ownership
  - Battery lifetime (or weight and cost) of portable devices
Platform Trends

• Growing independence and modularity of subsystems
  – Communication
  – Storage
    » Storage servers (e.g., EMC)
    » Storage-Area Networks (SAN)
    » Network-attached Storage (NAS)

• Scalable Technologies for Computing, Networking, and Information Systems
  – Systems that scale DOWN as well as UP
  – High performance workstations
  – Clusters and distributed systems
  – Massively parallel I/O and compute servers
  – National Information Infrastructure
  – “Cloud computing”;  Grid
Trends - Implications

• Individual technologies advance “smoothly”, but:
  – The crossing of certain barriers can result in leaps at the higher levels:
    » A certain transistor count permitted 32 bit CPU on a chip
    » A certain transistor count permitted cache on CPU chip
    » Obviating the need to go off chip sharply reduces cost and power, and can dramatically increase performance.
  – Differences in rates of improvement of different technologies can result in “inversions”: a different system components becomes the bottleneck ⇒ dramatic change in “optimal” architecture

• Useful approach: spend abundant resources in order to reduce the requirement for scarce ones
Performance Measures and Calculation
Metrics of Performance
(what we measure)

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)

Operations per Joule
Marketing Metrics

MIPS = Millions of instructions per second

- Machines with different instruction sets?
- Programs with different instruction mixes?
  - Dynamic frequency of instructions
- Uncorrelated with performance

MFLOP/s = FP Operations / Time * 10^6

- Machine dependent
- Often not where time is spent

<table>
<thead>
<tr>
<th>Normalized:</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>add, sub, compare, mult</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>divide, sqrt</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>exp, sin, . . .</td>
<td>8</td>
<td></td>
</tr>
</tbody>
</table>
Performance Terminology

“X is n% faster than Y” means:

\[
\frac{\text{Execution Time}(Y)}{\text{Execution Time}(X)} = \frac{\text{Performance}(X)}{\text{Performance}(Y)} = 1 + \frac{n}{100}
\]

\[
n = 100 \times \frac{(\text{Performance}(X) - \text{Performance}(Y))}{\text{Performance}(Y)}
\]
Performance Sensitivity

• Definitions:
  – $C_i$ contributor $i$ to performance
  – $P$ performance
  – $\Delta x$ absolute change in $x$
  – $\Delta x/x$ relative change in $x$

• Sensitivity of $P$ to change in $C_i$:
  – to absolute change in $C_i$: $\frac{\partial P(C_1, \ldots, C_i, \ldots, C_N)}{\partial C_i}$
  – to relative change in $C_i$: $\frac{\partial P(C_1, \ldots, C_i, \ldots, C_N)}{C_i}$
Relative Sensitivity

- Relative sensitivity of $P$ to *relative* changes in $C_i$ versus *its* relative sensitivity to relative changes in $C_j$:

\[
\frac{\partial P(C_1, \ldots, C_i, \ldots C_N)}{\partial C_i} \cdot \frac{C_i}{P(C_1, \ldots, C_i, \ldots C_N)} = \frac{\partial P(C_1, \ldots, C_i, \ldots C_N)}{\partial C_j} \cdot \frac{C_i}{C_j}
\]
Effect of changes on relative sensitivity - example

• Let $P = C_1 + C_2$ (larger $P$ is better)

• How does increasing $C_1$ affect the relative benefit of increasing $C_1$ rather than $C_2$ by the same percentage?

• Solution:

  relSens = $C_1 / C_2$

  so, the relative benefit of increasing $C_1$ grows.

• Underlying theory: Amdahl’s law.
Power & Performance

- Performance = 1/Execution Time
  Execution time = #-of-inSTRUCTIONS-in-Task \times Time-per-instruction
  Time-per-instruction = T \times CPI = 1/(Frequency \times IPC) \quad T=\text{cycle time}
  Performance = (IPC \times Frequency) / #-of-inSTRUCTIONS-in-Task

- For a given instruction stream:
  - Relative performance depends only on # of instr. executed per time-unit: 
    \text{IPC} \times \text{Frequency}
  - Sometimes, Measured in $\text{MIPS}$ - Million Instructions Per Second

- \text{Power} = \frac{1}{2} \times C \times V^2 \times Frequency \times \text{activity factor}
  - C = overall capacitance: for a given technology, \sim\text{proportional to # of transistors.}

- “Energy Efficiency” = \frac{\text{Performance}}{\text{Power}} <\text{MIPS/Watt}>

Caution: with pipelining, latency per instruction \neq 1/throughput
CPI = \text{Avg. No. of clock cycles between instruction completions}
Power & Performance

- Performance = \( \frac{1}{\text{Execution Time}} = \frac{(\text{IPC} \times \text{Frequency})}{\text{#-of-instructions-in-Task}} \).

- For a given instruction stream:

  \[
  \text{Performance} = \frac{\text{IPC} \times \text{Frequency}}{\text{#-of-instructions-in-Task}}
  \]

- \( \frac{1}{2} \times C \times V^2 \times \text{Frequency} \)
Brainiacs and Speed demons

Brainiacs

Speed demons

Source: ISCA 95, p. 174
Trends of Future Processors

Landscape of Microprocessor Families

** Data source www.spec.org
Cycles Per Instruction

Average Cycles per Instruction (CPI)

Average CPI = total number of clock cycles/ # of instructions executed

Execution time [sec] = Clock cycle time * \( \sum_{i=1}^{n} CPI_i \cdot I_i \)

\( I_i \) = number of times instruction \( i \) is executed in a program

\( CPI_i \) = Average number of clocks to complete per instruction \( i \)

Instruction Relative Frequency (\( F_i \))

Average CPI = \( \sum_{i=1}^{n} CPI_i \cdot F_i \)

where \( F_i = \frac{I_i}{\text{instruction count}} \)

\( F_i \) = relative frequency of appearance of instruction \( i \) in a program

Invest Resources where time is Spent!

Caution: in pipelined machines,

CPI = mean time between instruction completions!

Unambiguous alternative: IPC
## Example: Calculating CPI

### Typical Mix

<table>
<thead>
<tr>
<th>Op</th>
<th>Freq</th>
<th>Cycles</th>
<th>CPI(i)</th>
<th>(% Time)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALU</td>
<td>50%</td>
<td>1</td>
<td>.5</td>
<td>(33%)</td>
</tr>
<tr>
<td>Load</td>
<td>20%</td>
<td>2</td>
<td>.4</td>
<td>(27%)</td>
</tr>
<tr>
<td>Store</td>
<td>10%</td>
<td>2</td>
<td>.2</td>
<td>(13%)</td>
</tr>
<tr>
<td>Branch</td>
<td>20%</td>
<td>2</td>
<td>.4</td>
<td>(27%)</td>
</tr>
</tbody>
</table>

\[
\text{Avg CPI} = \sum CPI_i = 1.5
\]

\[
CPI_t = \text{Avg CPI} + CPI_m + CPI_x
\]

- \( CPI_m \): Average memory penalty access clocks per instruction
- \( CPI_x \): Any other penalty normalize per instruction!
Example (cont.)

Add register / memory operations:
   – One source operand in memory
   – One source operand in register
   – Cycle count of 2

Branch cycle count to increase to 3.

What fraction of the loads must be eliminated for this to pay off?

Base Machine (Reg / Reg)

<table>
<thead>
<tr>
<th>Op</th>
<th>Freq</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALU</td>
<td>50%</td>
<td>1</td>
</tr>
<tr>
<td>Load</td>
<td>20%</td>
<td>2</td>
</tr>
<tr>
<td>Store</td>
<td>10%</td>
<td>2</td>
</tr>
<tr>
<td>Branch</td>
<td>20%</td>
<td>2</td>
</tr>
</tbody>
</table>

Typical Mix
## Example Solution

**Exec Time** = Instr Cnt x CPI x Clock period

<table>
<thead>
<tr>
<th>Op</th>
<th>Freq</th>
<th>Cycles</th>
<th>Freq</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALU</td>
<td>.50</td>
<td>1</td>
<td>.5 – X</td>
<td>1</td>
</tr>
<tr>
<td>Load</td>
<td>.20</td>
<td>2</td>
<td>.2 – X</td>
<td>2</td>
</tr>
<tr>
<td>Store</td>
<td>.10</td>
<td>2</td>
<td>.1</td>
<td>2</td>
</tr>
<tr>
<td>Branch</td>
<td>.20</td>
<td>2</td>
<td>.2</td>
<td>3</td>
</tr>
<tr>
<td>Reg/Mem</td>
<td>X</td>
<td>2</td>
<td>2X</td>
<td>2</td>
</tr>
</tbody>
</table>

\[
\text{CPI}_{\text{New}} \text{ must be normalized to new instruction frequency}
\]

\[
\text{Cycles}_{\text{New}} = \frac{1.7 - X}{1 - X}
\]

\[
\text{Instructions}_{\text{New}} = \frac{1 - X}{(1 - X)}
\]
**Example Solution (cont.)**

Exec Time = Instr Cnt \( \times \) CPI \( \times \) Clock

<table>
<thead>
<tr>
<th>Op</th>
<th>Freq</th>
<th>Cycles</th>
<th>Freq</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALU</td>
<td>.50</td>
<td>1</td>
<td>.5 – X</td>
<td>1</td>
</tr>
<tr>
<td>Load</td>
<td>.20</td>
<td>2</td>
<td>.2 – X</td>
<td>2</td>
</tr>
<tr>
<td>Store</td>
<td>.10</td>
<td>2</td>
<td>.1</td>
<td>2</td>
</tr>
<tr>
<td>Branch</td>
<td>.20</td>
<td>2</td>
<td>.2</td>
<td>3</td>
</tr>
<tr>
<td>Reg/Mem</td>
<td></td>
<td></td>
<td>X</td>
<td>2</td>
</tr>
</tbody>
</table>

\[
\text{Instr Cnt}_{\text{Old}} \times \text{CPI}_{\text{Old}} \times \text{Clock}_{\text{Old}} = \text{Instr Cnt}_{\text{New}} \times \text{CPI}_{\text{New}} \times \text{Clock}_{\text{New}}
\]

\[
1.00 \times 1.5 = (1 - X) \times \frac{(1.7 - X)}{(1 - X)}
\]
### Example Solution (cont.)

**Exec Time = Instr Cnt × CPI × Clock**

<table>
<thead>
<tr>
<th>Op</th>
<th>Freq</th>
<th>Cycles</th>
<th>Freq</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALU</td>
<td>.50</td>
<td>.5</td>
<td>.5 – X</td>
<td>1</td>
</tr>
<tr>
<td>Load</td>
<td>.20</td>
<td>.4</td>
<td>.2 – X</td>
<td>2</td>
</tr>
<tr>
<td>Store</td>
<td>.10</td>
<td>.2</td>
<td>.1</td>
<td>2</td>
</tr>
<tr>
<td>Branch</td>
<td>.20</td>
<td>.3</td>
<td>.2</td>
<td>3</td>
</tr>
<tr>
<td>Reg/Mem</td>
<td>X</td>
<td>2</td>
<td></td>
<td>2X</td>
</tr>
<tr>
<td></td>
<td>1.00</td>
<td>1.5</td>
<td>1 – X</td>
<td>(1.7 – X)/(1 – X)</td>
</tr>
</tbody>
</table>

\[
\text{Instr Cnt}_{\text{Old}} \times \text{CPI}_{\text{Old}} \times \text{Clock}_{\text{Old}} = \text{Instr Cnt}_{\text{New}} \times \text{CPI}_{\text{New}} \times \text{Clock}_{\text{New}}
\]

\[
1.00 \times 1.5 = (1 - X) \times \frac{1.7 - X}{1 - X}
\]

\[
1.5 = 1.7 - X
\]

\[
0.2 = X
\]

**ALL** loads must be eliminated for this to be a win!
Performance Measurement and Summary
Measurement / Estimation Tools

• Run:
  – Benchmarks (“representative” programs)
  – Traces (of actual programs)
  – Kernels (just the core of a real program)
  – Mixes of the above

• Estimation using models: Cost, delay, area, power

• Simulation: e.g. ISA, RT, Gate, Circuit

• Queuing Theory

• Rules of Thumb/Fundamental Laws
Benchmarking Games

- Different configurations used to run the same workload on two systems
- Compiler wired to optimize the workload
- Test specification written to be biased towards one machine
- Synchronized CPU/IO intensive job sequence used
- Workload arbitrarily picked
- Very small benchmarks used
- Benchmarks manually translated to optimize performance
Common Benchmarking Mistakes

- Only average behavior represented in test workload
- Skew of device demands ignored
- Loading level controlled inappropriately
- Caching effects ignored
- Buffer sizes not appropriate
- Inaccuracies due to sampling ignored
Common Benchmarking Mistakes

• Ignoring monitoring overhead
• Not validating measurements
• Not ensuring same initial conditions
• Not measuring transient (cold start) performance
• Using device utilizations for performance comparisons
• Collecting too much data but doing too little analysis
Summarizing Performance

• Two main methods for representing performance of each machine for each program:
  – Absolute number (e.g., execution time)
  – Normalized to a reference machine (makes one believe that he can more easily predict performance of a new machine from that of the reference machine)

• Weighted arithmetic mean of execution time:
  \[ \sum (W_i * T_i); \ (\sum W_i = 1) \]

• Geometric mean: \[ \prod \text{(execution time ratios)}^{1/n}, \] where \( n \) is the number of programs used in the comparison and the ratio is the execution time of the evaluated processor over that of the reference machine

• Remark: advantage of geometric mean: comparison among machines independent of ref. Drawback: independent of weights!

• See H&P e3 p. 35-38 + exercise 1.10 and others for more.
Architectural Performance Laws and Rules of Thumb
Amdahl's Law

Speedup due to enhancement (E):

\[
\text{Speedup}(E) = \frac{\text{ExTime without } E}{\text{ExTime with } E} = \frac{\text{Performance with } E}{\text{Performance without } E}
\]

When enhancement E accelerates a fraction F of the task by a factor S, (the remainder of the task is unaffected) then:

\[
F = \frac{q}{h}
\]
Amdahl’s Law

\[ ExTime_{\text{new}} = ExTime_{\text{old}} \cdot \left( (1 - F) + \frac{F}{s} \right) \]

\[ Speedup_{\text{overall}} = \frac{ExTime_{\text{old}}}{ExTime_{\text{new}}} = \frac{1}{(1 - F) + \frac{F}{s}} \]

0<F<1: fraction of execution time to which acceleration was applied
s: speedup factor (of accelerated part)
Amdahl’s Law - Example

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

\[
\text{ExTime}_{\text{new}} = \text{ExTime}_{\text{old}} \times (0.9 + 0.5 \times 0.1) = 0.95 \times \text{ExTime}_{\text{old}}
\]

\[
\text{Speedup}_{\text{overall}} = \frac{1}{0.95} = 1.053
\]

Caution: This assumes that originally the execution time of an FP instruction was equal to that of others, and that instructions are executed sequentially (on at a time). Otherwise, the fraction of execution time taken by the FP instructions should be used instead of the fraction of FP instructions among the instruction executed!!!
Corollary: Make The Common Case Fast

- All instructions require an instruction fetch, only a fraction require a data fetch/store.
  - Optimize instruction access over data access
- Programs exhibit *locality*

  ![Diagram of spatial and temporal locality](image)

  - Access to small memories is faster
    - Provide a *storage hierarchy* such that the most frequent accesses are to the smallest (closest) memories.
Additional Rules of Thumb

- Avg. IPC = 0.5 – 2 (higher in FP)
- “Balanced” computer: 1 MIPS, 1 Megabyte, 1Mbit/sec
- Frequency of cache misses (<5%)
- Program executes 90% of its instruction in 10% of its code (“10:90” rule)
- Memory access: access of small memory is faster
Additional Rules of Thumb (cont.)

- The simple case is usually the most frequent and the easiest to optimize!

- Do simple, fast things in hardware and be sure that the rest can be handled correctly in software.

- (Over)spend abundant resources in order to reduce the requirement for scarce ones.
Cost, Price, and Price for Performance
Cost/Performance
What is Relationship of Cost to Price?

- **Component Costs**

- **Direct Costs** (add 25% to 40%) recurring costs: labor, purchasing, scrap, warranty

- **Gross Margin** (add 82% - 186%) nonrecurring costs: R&D, marketing, sales, equipment maintenance, rental, financing cost, pretax profits, taxes

- **Average Discount** to get List Price (add 33% to 66%): volume discounts and/or retailer markup

<table>
<thead>
<tr>
<th>List Price</th>
<th>Average Discount</th>
<th>Gross Margin</th>
</tr>
</thead>
<tbody>
<tr>
<td>Avg. Selling Price</td>
<td>25% to 40%</td>
<td>34% to 39%</td>
</tr>
</tbody>
</table>

| Component Cost | 6% to 8% | 15% to 33% |
$1k$ PC – ~Cost Breakdown (2001)

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

Figure 1.9 Estimated distribution of costs of the components in a $1000$ PC in 2001.
Learning Curve

- Production costs
- Volume
- Time to introduce new product

Years
Making a Profit

• High Margins:
  – No competition
  – Highly efficient production
  – High-End Machines
  – Custom machines

• Huge volume:
  – R&D and other fixed costs are negligible

<table>
<thead>
<tr>
<th></th>
<th>MIPS R4000</th>
<th>Intel Pentium IV</th>
</tr>
</thead>
<tbody>
<tr>
<td>Development cost</td>
<td>~$30M</td>
<td>~$800M</td>
</tr>
<tr>
<td>Units sold</td>
<td>300K</td>
<td>500M</td>
</tr>
<tr>
<td>Development cost per die</td>
<td>$100</td>
<td>$1.6</td>
</tr>
</tbody>
</table>