Advanced Computer Architecture

Tran Ngoc Thinh
HCMC University of Technology
http://www.cse.hcmut.edu.vn/~tnthinh/aca

Administrative Issues

• Class
  – Time and venue: Thursdays, 6:30am - 09:00am, 605B4
  – Web page:
    • http://www.cse.hcmut.edu.vn/~tnthinh/aca

• Textbook:
  – Research papers on Computer Design and Architecture from IEEE and ACM conferences, transactions and journals
• Grades
  – 10% homeworks
  – 20% presentations
  – 20% midterm exam
  – 50% final exam

• Personnel
  – Instructor: Dr. Tran Ngoc Thinh
    • Email: tnthinh@cse.hcmut.edu.vn
    • Phone: 8647256 (5843)
    • Office: A3 building
    • Office hours: Thursdays, 09:00-11:00

  – TA: Mr. Tran Huy Vu
    • Email:vutran@cse.hcmut.edu.vn
    • Phone: 8647256 (5843)
    • Office: A3 building
    • Office hours:
Course Coverage

• Introduction
  – Brief history of computers
  – Basic concepts of computer architecture.

• Instruction Set Principle
  – Classifying Instruction Set Architectures
  – Addressing Modes, Type and Size of Operands
  – Operations in the Instruction Set, Instructions for Control Flow, Instruction Format
  – The Role of Compilers

Course Coverage

• Pipelining: Basic and Intermediate Concepts
  – Organization of pipelined units,
  – Pipeline hazards,
  – Reducing branch penalties, branch prediction strategies.

• Instructional Level Parallelism
  – Temporal partitioning
  – List-scheduling approach
  – Integer Linear Programming
  – Network Flow
  – Spectral methods
  – Iterative improvements
Course Coverage

• Memory Hierarchy Design
  – Memory hierarchy
  – Cache memories
  – Virtual memories
  – Memory management.

• SuperScalar Architectures
  – Instruction level parallelism and machine parallelism
  – Hardware techniques for performance enhancement
  – Limitations of the superscalar approach

• Vector Processors

Course Requirements

• Computer Organization & Architecture
  – Comb./Seq. Logic, Processor, Memory, Assembly Language

• Data Structures / Algorithms
  – Complexity analysis, efficient implementations

• Operating Systems
  – Task scheduling, management of processors, memory, input/output devices
## Computer Architecture's Changing Definition

- **1950s to 1960s:** Computer Architecture Course: Computer Arithmetic
- **1970s to mid 1980s:** Computer Architecture Course: Instruction Set Design, especially ISA appropriate for compilers
- **1990s:** Computer Architecture Course: Design of CPU, memory system, I/O system, Multiprocessors, Networks
- **2000s:** Multi-core design, on-chip networking, parallel programming paradigms, power reduction
- **2010s:** Computer Architecture Course: Self adapting systems? Self organizing structures? DNA Systems/Quantum Computing?

## Computer Architecture

- Role of a computer architect:
  - To design and engineer the various levels of a computer system to maximize **performance** and **programmability** within limits of **technology** and **cost**
Levels of Abstraction

- S/W and H/W consists of hierarchical layers of abstraction, each hides details of lower layers from the above layer.
- The instruction set arch. abstracts the H/W and S/W interface and allows many implementation of varying cost and performance to run the same S/W.

The Task of Computer Designer

- determine what attribute are important for a new machine.
- design a machine to maximize cost performance.
- What are these Task?
  - instruction set design
  - function organization
  - logic design
  - implementation
  - IC design, packaging, power, cooling….
  - …
History

• **Big Iron” Computers:**
  – Used vacuum tubes, electric relays and bulk magnetic storage devices. No microprocessors. No memory.

• Example: ENIAC (1945), IBM Mark 1 (1944)

---

History

• **Von Newmann:**
  – Invented EDSAC (1949).
  – First Stored Program Computer. Uses Memory.

• Importance: We are still using The same basic design.
The Processor Chip

Intel 4004 Die Photo

- Introduced in 1970
  - First microprocessor
- 2,250 transistors
- 12 mm$^2$
- 108 KHz
**Intel 8086 Die Scan**

- 29,000 transistors
- 33 mm$^2$
- 5 MHz
- Introduced in 1979
  - Basic architecture of the IA32 PC

**Intel 80486 Die Scan**

- 1,200,000 transistors
- 81 mm$^2$
- 25 MHz
- Introduced in 1989
  - 1st pipelined implementation of IA32
Pentium Die Photo

- 3,100,000 transistors
- 296 mm\(^2\)
- 60 MHz
- Introduced in 1993
  - 1st superscalar implementation of IA32

Pentium III

- 9,5000,000 transistors
- 125 mm\(^2\)
- 450 MHz
- Introduced in 1999
Moore’s Law

- “Cramming More Components onto Integrated Circuits”
  - Gordon Moore, Electronics, 1965
- # on transistors on cost-effective integrated circuit double every 18 months

Performance Trend

- In general, tradeoffs should improve performance
- The natural idea here… HW cheaper, easier to manufacture → can make our processor do more things…
Price Trends (Pentium III)

- Diagram showing Intel flat price (1000 units) vs. date, with price points at 400 MHz, 500 MHz, 600 MHz, 735 MHz, 867 MHz, and 1000 MHz over time.

Price Trends (DRAM memory)

- Diagram showing dollars per DRAM chip vs. year, with price points at 64K bits, 256K bits, 1M bits, 4M bits, and 16M bits over time.
Technology constantly on the move!

- Num of transistors not limiting factor
  - Currently ~ 1 billion transistors/chip
  - Problems:
    - Too much Power, Heat, Latency
    - Not enough Parallelism
- 3-dimensional chip technology?
  - Sandwiches of silicon
  - “Through-Vias” for communication
- On-chip optical connections?
  - Power savings for large packets
- The Intel® Core™ i7 microprocessor (“Nehalem”)
  - 4 cores/chip
  - 45 nm, Hafnium hi-k dielectric
  - 731M Transistors
  - Shared L3 Cache - 8MB
  - L2 Cache - 1MB (256K x 4)

Crossroads: Uniprocessor Performance


- VAX: 25%/year 1978 to 1986
- RISC + x86: 52%/year 1986 to 2002
- RISC + x86: ??%/year 2002 to present
Limiting Force: Power Density

Moore’s Law Extrapolation:
Power Density for Leading Edge Microprocessors

- 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
  ⇒ Sea change in chip design: multiple “cores” (2X processors per chip / ~ 2 years)
    - More power efficient to use a large number of simpler processors rather than a small number of complex processors
Sea Change in Chip Design

- Intel 4004 (1971):
  - 4-bit processor,
  - 2312 transistors, 0.4 MHz,
  - 10 μm PMOS, 11 mm² chip
- RISC II (1983):
  - 32-bit, 5 stage
  - pipeline, 40,760 transistors, 3 MHz,
  - 3 μm NMOS, 60 mm² chip
- 125 mm² chip, 65 nm CMOS
  = 2312 RISC II+FPU+Icache+Dcache
  - RISC II shrinks to ~ 0.02 mm² at 65 nm
  - Caches via DRAM or 1 transistor SRAM (www.t-ram.com) ?
  - Proximity Communication via capacitive coupling at > 1 TB/s ?
  (Ivan Sutherland @ Sun / Berkeley)
- Processor is the new transistor?

ManyCore Chips: The future is here

- Intel 80-core multicore chip (Feb 2007)
  - 80 simple cores
  - Two FP-engines / core
  - Mesh-like network
  - 100 million transistors
  - 65nm feature size
  - 24 “tiles” with two IA cores per tile
  - 24-router mesh network with 256 GB/s bisection
  - 4 integrated DDR3 memory controllers
  - Hardware support for message-passing
- “ManyCore” refers to many processors/chip
  - 64? 128? Hard to say exact boundary
- How to program these?
  - Use 2 CPUs for video/audio
  - Use 1 for word processor, 1 for browser
  - 76 for virus checking???
- Something new is clearly needed here…
The End of the Uniprocessor Era

Single biggest change in the history of computing systems

• Multiprocessors imminent in 1970s, ‘80s, ‘90s, …
• “… today’s processors … are nearing an impasse as technologies approach the speed of light…”

• ⇒ Custom multiprocessors strove to lead uniprocessors
  ⇒ Procrastination rewarded: 2X seq. perf. / 1.5 years
• “We are dedicating all of our future product development to multicore designs. … This is a sea change in computing”
  Paul Otellini, President, Intel (2004)

• Difference is all microprocessor companies switch to multicore (AMD, Intel, IBM, Sun; all new Apples 2-4 CPUs)
  ⇒ Procrastination penalized: 2X sequential perf. / 5 yrs
  ⇒ Biggest programming challenge: 1 to 2 CPUs
Problems with Sea Change

- Algorithms, Programming Languages, Compilers, Operating Systems, Architectures, Libraries, … not ready to supply Thread Level Parallelism or Data Level Parallelism for 1000 CPUs / chip
  - Need whole new approach
  - People have been working on parallelism for over 50 years without general success
- Architectures not ready for 1000 CPUs / chip
  - Unlike Instruction Level Parallelism, cannot be solved by just by computer architects and compiler writers alone, but also cannot be solved without participation of computer architects
- PARLab: Berkeley researchers from many backgrounds meeting since 2005 to discuss parallelism
  - Krste Asanovic, Ras Bodik, Jim Demmel, Kurt Keutzer, John Kubiatowicz, Edward Lee, George Necula, Dave Patterson, Koushik Sen, John Shalf, John Wawrzynek, Kathy Yelick, …
  - Circuit design, computer architecture, massively parallel computing, computer-aided design, embedded hardware and software, programming languages, compilers, scientific programming, and numerical analysis

Computer Design Cycle

- Evaluate Existing Systems for Bottlenecks
- Simulate New Designs and Organizations
- Benchmark Workloads
- Implementation Complexity
The computer design is evaluated for bottlenecks using certain benchmarks to achieve the optimum performance.

**Performance** (Metric)

- **Time/Latency**: The wall clock or CPU elapsed time.
- **Throughput**: The number of results per second.

Other measures such as MIPS, MFLOPS, clock frequency (MHz), cache size do not make any sense.
Performance (Measuring Tools)

- Benchmarks:
- **Hardware:** Cost, delay, area, power consumption
- **Simulation** (at levels - ISA, RT, Gate, Circuit)
- Queuing Theory
- Fundamental “Laws”/Principles

Computer Design Cycle

1: **Performance**
Evaluate Existing Systems for Bottlenecks using Benchmarks

2: **Technology**
Workloads

Simulate New Designs and Organizations

The Technology Trends motivate new designs. These designs are simulated to evaluate the performance for different levels of workloads. Simulation helps in keeping the result verification
### Technology Trends: Computer Generations

<table>
<thead>
<tr>
<th>Technology</th>
<th>Generation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vacuum tube</td>
<td>1946-1957 1&lt;sup&gt;st&lt;/sup&gt; Gen.</td>
</tr>
<tr>
<td>Transistor</td>
<td>1958-1964 2&lt;sup&gt;nd&lt;/sup&gt; Gen.</td>
</tr>
<tr>
<td>Small scale integration</td>
<td>1965-1968</td>
</tr>
<tr>
<td>- Up to 100 devices/chip</td>
<td></td>
</tr>
<tr>
<td>Medium scale integration</td>
<td>1969-1971 3&lt;sup&gt;rd&lt;/sup&gt; Gen.</td>
</tr>
<tr>
<td>- 100-3,000 devices/chip</td>
<td></td>
</tr>
<tr>
<td>Large scale integration</td>
<td>1972-1977</td>
</tr>
<tr>
<td>- 3,000 - 100,000 devices/chip</td>
<td></td>
</tr>
<tr>
<td>Very large scale integration</td>
<td>1978 on.. 4&lt;sup&gt;th&lt;/sup&gt; Gen.</td>
</tr>
<tr>
<td>- 100,000 - 100,000,000 devices/chip</td>
<td></td>
</tr>
<tr>
<td>Ultra large scale integration</td>
<td></td>
</tr>
<tr>
<td>- Over 100,000,000 devices/chip</td>
<td></td>
</tr>
</tbody>
</table>

### Computer Design Cycle

1: Performance  
2: Technology  
3: Cost  
Implementation Complexity

The systems are implemented using the latest technology to obtain cost effective, high performance solution - the implementation complexities are given due consideration.
The relationship between cost and price is a complex one. The cost is the total amount spent to produce a product. The price is the amount for which a finished good is sold. The cost passes through different stages before it becomes price. A small change in cost may have a big impact on price.

**Price Verses Cost**

- **Manufacturing Costs**: Total amount spent to produce a component
  - **Component Cost**: Cost at which the components are available to the designer. It ranges from 40% to 50% of the list price of the product.
  - **Direct cost (Recurring costs)**: Labor, purchasing, scrap, warranty – 4% - 16% of list price
  - **Gross margin** – Non-recurring cost: R&D, marketing, sales, equipment, rental, maintenance, financing cost, pre-tax profits, taxes
**Price vs. Cost**

- **List Price:**
  - Amount for which the finished good is sold;
  - it includes *Average Discount* of 15% to 35% of the as volume discounts and/or retailer markup

---

**Cost-effective IC Design: Price-Performance Design**

- **Yield:** Percentage of manufactured components surviving testing
- **Volume:** increases manufacturing hence decreases the list price and improves the purchasing efficiency
- **Feature Size:** the minimum size of a transistor or wire in either x or y direction
Cost-effective IC Design: Price-Performance Design

- Reduction in feature size from 10 microns in 1971 and 0.045 in 2008 has resulted in:
  - Quadratic rise in transistor count
  - Linear increase in performance
  - 4-bit to 64-bit microprocessor
  - Desktops have replaced time-sharing machines

Cost of Integrated Circuits

**Manufacturing Stages:**

The Integrated circuit manufacturing passes through many stages:

- Wafer growth and testing
- Wafer chopping it into *dies*
- Packaging the *dies to chips*
- Testing a chip.
Cost of Integrated Circuits

Die: is the square area of the wafer containing the integrated circuit

See that while fitting dies on the wafer the small wafer area around the periphery goes waist

Cost of a die: The cost of a die is determined from cost of a wafer; the number of dies fit on a wafer and the percentage of dies that work, i.e., the yield of the die.

The cost of integrated circuit can be determined as ratio of the total cost; i.e., the sum of the costs of die, cost of testing die, cost of packaging and the cost of final testing a chip; to the final test yield.

Cost of IC = \[ \frac{\text{die cost} + \text{die testing cost} + \text{packaging cost} + \text{final testing cost}}{\text{final test yield}} \]

- The cost of die is the ratio of the cost of the wafer to the product of the dies per wafer and die yield

\[ \text{Die cost} = \frac{\text{Cost of wafer}}{\text{dies per wafer} \times \text{die yield}} \]
Cost of Integrated Circuits

• The number of dies per wafer is determined by the dividing the wafer area (minus the waist wafer area near the round periphery) by the die area

\[
\text{Dies per wafer} = \frac{\pi \left(\frac{\text{wafer diameter}}{2}\right)^2}{\text{die area}} - \frac{\pi \left(\frac{\text{wafer diameter}}{2}\right)}{\sqrt{2 \times \text{die area}}}
\]

Example: For die of 0.7 cm on a side, find the number of dies per wafer of 30 cm diameter

Answer:

\[
\left[\frac{\text{Wafer area}}{\text{Die Area}}\right] - \text{Wafer Waist area} = \frac{\pi \left(30/2\right)^2 / 0.49 - \pi (30) / \sqrt{2 \times 0.49}}{\sqrt{2 \times 0.49}} = 1347 \text{ dies}
\]

Calculating Die Yield

• Die yield is the fraction or percentage of good dies on a wafer number
• Wafer yield accounts for completely bad wafers so need not be tested
• Wafer yield corresponds to defect density by \( \alpha \) which depends on number of masking levels good estimate for CMOS is 4.0

\[
\text{Die Yield} = \text{Wafer Yield} \times \left(1 + \frac{\text{(Defect/Unit Area)} \times \text{Die Area}}{\alpha}\right)^{-\alpha}
\]

Example:
The yield of a die, 0.7cm on a side, with defect density of 0.6/cm²

\[
= (1 + [0.6 \times 0.49]/4.0)^{-4} = 0.75
\]
Volume vs. Cost

- Rule of thumb on applying learning curve to manufacturing:
  
  "When volume doubles, costs reduce 10%"


<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>PC</td>
<td>23,880,898</td>
<td>33,547,589</td>
<td>44,006,000</td>
<td>65,480,000</td>
</tr>
<tr>
<td>WS</td>
<td>407,624</td>
<td>584,544</td>
<td>679,320</td>
<td>978,585</td>
</tr>
<tr>
<td>Ratio</td>
<td>59</td>
<td>57</td>
<td>65</td>
<td>67</td>
</tr>
</tbody>
</table>

- $2^x = 65 \Rightarrow x = 6.0$
- Since doubling value reduces cost by 10%, costs reduces to $(0.9)^{6.0} = 0.53$ of the original price.
- PC costs are 47% less than workstation costs for whole market.

High Margins on High-End Machines

- R&D considered return on investment (ROI) 10% 
  - Every $1$ R&D must generate $7$ to $13$ in sales
- **High end machines need more $ for R&D**
- Sell fewer high end machines 
  - Fewer to amortize R&D
  - Much higher margins
- Cost of 1 MB Memory (January 1994):
  
<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>PC</td>
<td>$40  (Mac Quadra)</td>
</tr>
<tr>
<td>WS</td>
<td>$42  (SS-10)</td>
</tr>
<tr>
<td>Mainframe</td>
<td>$1920 (IBM 3090)</td>
</tr>
<tr>
<td>Supercomputer</td>
<td>$600 (M90 DRAM)</td>
</tr>
<tr>
<td></td>
<td>$1375 (C90 15 ns SRAM)</td>
</tr>
</tbody>
</table>
Recouping Development Cost on Low Volume Microprocessors?

- Hennessy says MIPS R4000 cost $30M to develop
- Intel rumored to invest $100M on 486
- SGI/MIPS sells 300,000 R4000s over product lifetime?
- Intel sells 50,000,000 486s?
- Intel must get $100M from chips ($2/chip)
- SGI/MIPS can get $30M from margin of workstations vs. chips vs. $100/chip
- Alternative: SGI buys chips vs. develops them

Metrics of Performance

- **MIPS**: Millions of Instructions per second
- **MFLOPS**: millions of FP operations per sec.

```
Application
Programming
Language
Compiler

Instruction Set Architecture

Datapath
Control
Function Units
Transistors
Pins/Wire – I/O

Answers per month
Operations per second
Megabytes per second
Cycles per second (clock rate)
```
Does Anybody Really Know What Time it is?

**UNIX Time Command:** 90.7u 12.9s 2:39 65%

- User CPU Time (Time spent in program): 90.7 sec
- System CPU Time (Time spent in OS): 12.9 sec
- Elapsed Time (Response Time = 2 min 39 sec = 159 Sec.)
  - \((90.7 + 12.9) / 159 \times 100 = 65\%\), % of lapsed time that is CPU time. 45% of the time spent in I/O or running other programs

---

**Time**

**CPU time**
- time the CPU is computing
- not including the time waiting for I/O or running other program

**User CPU time**
- CPU time spent in the program

**System CPU time**
- CPU time spent in the operating system performing task requested by the program decrease execution time

**CPU time = User CPU time + System CPU time**
Performance

System Performance
- elapsed time on unloaded system

CPU performance
- user CPU time on an unloaded system

Two notions of “performance”

<table>
<thead>
<tr>
<th>Plane</th>
<th>DC to Paris</th>
<th>Speed</th>
<th>Passengers</th>
<th>Throughput</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boeing 747</td>
<td>6.5 hours</td>
<td>610 mph</td>
<td>470</td>
<td>286,700</td>
</tr>
<tr>
<td>BAD/Sud Concorde</td>
<td>3 hours</td>
<td>1350 mph</td>
<td>132</td>
<td>178,200</td>
</tr>
</tbody>
</table>

Which has higher performance?

° **Time to do the task (Execution Time)**
  - execution time, response time, latency

° **Tasks per day, hour, week, sec, ns. ..**
  - throughput, bandwidth

Response time and throughput often are in opposition
Example

- Time of Concorde vs. Boeing 747?
  - Concord is 1350 mph / 610 mph = 2.2 \textit{times faster} = 6.5 hours / 3 hours
- Throughput of Concorde vs. Boeing 747?
  - Concord is 178,200 pmph / 286,700 pmph = 0.62 "times faster"
  - Boeing is 286,700 pmph / 178,200 pmph = 1.6 "times faster"

- Boeing is 1.6 times ("60\%") faster in terms of throughput
- Concord is 2.2 times ("120\%") faster in terms of flying time

We will focus primarily on execution time for a single job

Computer Performance Measures: Program Execution Time (1/2)

- For a specific program compiled to run on a specific machine (CPU) "A", the following parameters are provided:
  - The total instruction count of the program.
  - The average number of cycles per instruction (average CPI).
  - Clock cycle of machine "A"
Computer Performance Measures: Program Execution Time (2/2)

- How can one measure the performance of this machine running this program?
  - Intuitively the machine is said to be faster or has better performance running this program if the total execution time is shorter.
  - Thus the inverse of the total measured program execution time is a possible performance measure or metric:

\[
\text{Performance}_A = \frac{1}{\text{Execution Time}_A}
\]

How to compare performance of different machines? What factors affect performance? How to improve performance?!!!!

Comparing Computer Performance Using Execution Time

- To compare the performance of two machines (or CPUs) “A”, “B” running a given specific program:

  \[
  \text{Performance}_A = \frac{1}{\text{Execution Time}_A}
  \]
  \[
  \text{Performance}_B = \frac{1}{\text{Execution Time}_B}
  \]

- Machine A is \( n \) times faster than machine B means (or slower? if \( n < 1 \)):

\[
\text{Speedup} = n = \frac{\text{Performance}_A}{\text{Performance}_B} = \frac{\text{Execution Time}_B}{\text{Execution Time}_A}
\]
Example

For a given program:

Execution time on machine A: \( \text{Execution}_A = 1 \) second
Execution time on machine B: \( \text{Execution}_B = 10 \) seconds

\[
\text{Speedup} = \frac{\text{Performance}_A}{\text{Performance}_B} = \frac{\text{Execution Time}_B}{\text{Execution Time}_A} = \frac{10}{1} = 10
\]

The performance of machine A is 10 times the performance of machine B when running this program, or: Machine A is said to be 10 times faster than machine B when running this program.

The two CPUs may target different ISAs provided the program is written in a high level language (HLL).

CPU Execution Time: The CPU Equation

- A program is comprised of a number of instructions executed, \( I \)
  - Measured in: instructions/program
- The average instruction executed takes a number of cycles per instruction (CPI) to be completed.
  - Measured in: cycles/instruction, CPI
- CPU has a fixed clock cycle time \( C = 1 / \text{clock rate} \)
  - Measured in: seconds/cycle
- CPU execution time is the product of the above three parameters as follows:

\[
T = \frac{\text{Seconds}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Cycle}}
\]

\[
T = \frac{I \times \text{CPI} \times C}{\text{Execution Time per program in seconds}} = \frac{I \times \text{Average CPI for program}}{\text{CPU Clock Cycle}}
\]
CPU Execution Time: Example

• A Program is running on a specific machine with the following parameters:
  – Total executed instruction count: 10,000,000 instructions
    Average CPI for the program: 2.5 cycles/instruction.
  – CPU clock rate: 200 MHz. (clock cycle = 5x10^-9 seconds)

• What is the execution time for this program:

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

  = 10,000,000 \times 2.5 \times \frac{1}{5x10^{-9}}

  = 10,000,000 \times 2.5 \times 5x10^{-9}

  = 0.125 \text{ seconds}

Factors Affecting CPU Performance

<table>
<thead>
<tr>
<th></th>
<th>Instruction Count</th>
<th>CPI</th>
<th>Clock Cycle</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Compiler</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Instruction Set</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Architecture (ISA)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Organization</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(CPU Design)</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Technology</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>(VLSI)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Performance Comparison: Example

- From the previous example: A Program is running on a specific machine with the following parameters:
  - Total executed instruction count, \( I \): 10,000,000 instructions
  - Average CPI for the program: 2.5 cycles/instruction.
  - CPU clock rate: 200 MHz.
- Using the same program with these changes:
  - A new compiler used: New instruction count 9,500,000
    New CPI: 3.0
  - Faster CPU implementation: New clock rate = 300 MHz
- What is the speedup with the changes?

\[
\text{Speedup} = \frac{I_{\text{old}} \times \text{CPI}_{\text{old}} \times \text{Clock cycle}_{\text{old}}}{I_{\text{new}} \times \text{CPI}_{\text{new}} \times \text{Clock Cycle}_{\text{new}}}
\]

\[
\text{Speedup} = \frac{(10,000,000 \times 2.5 \times 5 \times 10^{-9})}{(9,500,000 \times 3 \times 3.33 \times 10^{-9})}
\]

\[
= \frac{.125}{.095} = 1.32
\]

or 32% faster after changes.

Instruction Types & CPI

- Given a program with \( n \) types or classes of instructions executed on a given CPU with the following characteristics:

\[
C_i = \text{Count of instructions of type}_i
\]
\[
\text{CPI}_i = \text{Cycles per instruction for type}_i \quad i = 1, 2, \ldots, n
\]

Then:

\[
\text{CPI} = \frac{\text{CPU Clock Cycles}}{\text{Instruction Count} \ I}
\]

Where:

\[
\text{CPU clock cycles} = \sum_{i=1}^{n} (\text{CPI}_i \times C_i)
\]

\[
\text{Instruction Count} \ I = \sum C_i
\]
Instruction Types & CPI: An Example

- An instruction set has \( n = \text{three} \) instruction classes:

<table>
<thead>
<tr>
<th>Instruction class</th>
<th>CPI</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
</tr>
<tr>
<td>B</td>
<td>2</td>
</tr>
<tr>
<td>C</td>
<td>3</td>
</tr>
</tbody>
</table>

- Two code sequences have the following instruction counts:

<table>
<thead>
<tr>
<th>Code Sequence</th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

- CPU cycles for sequence 1: \( 2 \times 1 + 1 \times 2 + 2 \times 3 = 10 \) cycles
  CPI for sequence 1: \( \frac{10}{5} = 2 \)
- CPU cycles for sequence 2: \( 4 \times 1 + 1 \times 2 + 1 \times 3 = 9 \) cycles
  CPI for sequence 2: \( \frac{9}{6} = 1.5 \)

Instruction Frequency & CPI

- Given a program with \( n \) types or classes of instructions with the following characteristics:
  \( C_i \) = Count of instructions of type \( i \)
  \( CPI_i \) = Average cycles per instruction of type \( i \)
  \( F_i \) = Frequency or fraction of instruction type \( i \) executed
    \( = \frac{C_i}{\text{total executed instruction count}} = \frac{C_i}{I} \)

Then:

\[
\text{CPI} = \sum_{i=1}^{n} \left( CPI_i \times F_i \right)
\]

Fraction of total execution time for instructions of type \( i \) = \( \frac{\text{CPI}_i \times F_i}{\text{CPI}} \)
### Program Profile or Executed Instructions Mix

<table>
<thead>
<tr>
<th>Op</th>
<th>Freq, F</th>
<th>CPI</th>
<th>CPI x F</th>
<th>% Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALU</td>
<td>50%</td>
<td>.5</td>
<td>.5/2.2</td>
<td>23%</td>
</tr>
<tr>
<td>Load</td>
<td>20%</td>
<td>1.0</td>
<td>1/2.2</td>
<td>45%</td>
</tr>
<tr>
<td>Store</td>
<td>10%</td>
<td>.3</td>
<td>.3/2.2</td>
<td>14%</td>
</tr>
<tr>
<td>Branch</td>
<td>20%</td>
<td>.4</td>
<td>.4/2.2</td>
<td>18%</td>
</tr>
</tbody>
</table>

Typical Mix

\[
\text{CPI} = \sum_{i=1}^{n} (CPI_i \times F_i)
\]

\[
\text{Sum} = 2.2
\]

### Performance Terminology

“X is n% faster than Y” means:

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

\[
\text{ExTime}(X) \times \text{Performance}(Y) = 100
\]

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

\[
n = \frac{100(\text{ExTime}(Y) - \text{ExTime}(X))}{\text{ExTime}(X)}
\]

Example: Y takes 15 seconds to complete a task, X takes 10 seconds. What % faster is X?

\[
n = \frac{100(15 - 10)}{10} = 50%
\]
Speedup

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 of the task by a factor Speedup\text{enhanced}, and the remainder of the task is unaffected, then what is ExTime(E)?

Speedup(E) = ?

Amdahl’s Law

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

\[
\text{Speedup} = \frac{\text{Performance for entire task using the enhancement}}{\text{Performance for the entire task without using the enhancement}}
\]

or \[
\text{Speedup} = \frac{\text{Execution time without the enhancement}}{\text{Execution time for entire task using the enhancement}}
\]

\[
\text{ExTime}_{\text{new}} = \text{ExTime}_{\text{old}} \times \left( 1 - \frac{\text{Fraction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}} \right) + \frac{\text{Fraction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}}
\]

\[
\text{Speedup}_{\text{overall}} = \frac{\text{ExTime}_{\text{old}}}{\text{ExTime}_{\text{new}}} = \frac{1}{\left( 1 - \frac{\text{Fraction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}} \right) + \frac{\text{Fraction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}}}
\]
Example of Amdahl’s Law

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

\[
\text{ExTime}_{\text{new}} = \text{ExTime}_{\text{old}} \times (0.9 + 0.1/2) = 0.95 \times \text{ExTime}_{\text{old}}
\]

\[
\text{Speedup}_{\text{overall}} = \frac{1}{0.95} = 1.053
\]
Pictorial Depiction of Amdahl’s Law

Enhancement E accelerates fraction F of original execution time by a factor of S

Before:
Execution Time without enhancement E: (Before enhancement is applied)
- shown normalized to 1 = (1-F) + F = 1

Unaffected fraction: (1-F)
Affected fraction: F

After:
Execution Time with enhancement E:

\[
\text{Speedup}(E) = \frac{\text{Execution Time without enhancement E}}{\text{Execution Time with enhancement E}} = \frac{1}{(1 - F) + \frac{F}{S}}
\]

Performance Enhancement Example

- For the RISC machine with the following instruction mix given earlier:

<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>23%</td>
</tr>
<tr>
<td>Load</td>
<td>20%</td>
<td>5</td>
<td>1.0</td>
<td>45%</td>
</tr>
<tr>
<td>Store</td>
<td>10%</td>
<td>3</td>
<td>.3</td>
<td>14%</td>
</tr>
<tr>
<td>Branch</td>
<td>20%</td>
<td>2</td>
<td>.4</td>
<td>18%</td>
</tr>
</tbody>
</table>

- If a CPU design enhancement improves the CPI of load instructions from 5 to 2, what is the resulting performance improvement from this enhancement:

\[
\text{Fraction enhanced} = F = 45\% = .45
\]
\[
\text{Unaffected fraction} = 100\% - 45\% = 55\% = .55
\]
\[
\text{Factor of enhancement} = \frac{5}{2} = 2.5
\]

Using Amdahl’s Law:
\[
\text{Speedup}(E) = \frac{1}{(1 - F) + \frac{F}{S}} = \frac{1}{.55 + .45/2.5} = 1.37
\]
An Alternative Solution Using CPU Equation

<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>23%</td>
</tr>
<tr>
<td>Load</td>
<td>20%</td>
<td>5</td>
<td>1.0</td>
<td>45%</td>
</tr>
<tr>
<td>Store</td>
<td>10%</td>
<td>3</td>
<td>.3</td>
<td>14%</td>
</tr>
<tr>
<td>Branch</td>
<td>20%</td>
<td>2</td>
<td>.4</td>
<td>18%</td>
</tr>
</tbody>
</table>

- If a CPU design enhancement improves the CPI of load instructions from 5 to 2, what is the resulting performance improvement from this enhancement:

  Old CPI = 2.2
  New CPI = \( .5 \times 2 + .2 \times 2 + .1 \times 3 + .2 \times 2 = 1.6 \)

  \[
  \text{Speedup}(E) = \frac{\text{Original Execution Time} \times \text{Instruction count} \times \text{old CPI} \times \text{clock cycle}}{\text{New Execution Time} \times \text{Instruction count} \times \text{new CPI} \times \text{clock cycle}}
  \]

  Old CPI \( \frac{2.2}{1.6} = 1.37 \)

Which is the same speedup obtained from Amdahl's Law in the first solution.

Extending Amdahl's Law To Multiple Enhancements

- Suppose that enhancement \( E_i \) accelerates a fraction \( F_i \) of the execution time by a factor \( S_i \) and the remainder of the time is unaffected then:

  \[
  \text{Speedup} = \frac{1}{\left(1 - \sum_i F_i \right) \times \text{Original Execution Time}}
  \]

  \[
  \text{Speedup} = \frac{1}{\left(1 - \sum_i F_i \right) \times \sum_i \frac{F_i}{S_i}}
  \]

  Note: All fractions \( F_i \) refer to original execution time before the enhancements are applied.
Amdahl's Law With Multiple Enhancements: Example

- Three CPU performance enhancements are proposed with the following speedups and percentage of the code execution time affected:
  - Speedup\(_1\) = S\(_1\) = 10 \(\text{Percentage}_1 = F\(_1\) = 20\%\)
  - Speedup\(_2\) = S\(_2\) = 15 \(\text{Percentage}_2 = F\(_2\) = 15\%\)
  - Speedup\(_3\) = S\(_3\) = 30 \(\text{Percentage}_3 = F\(_3\) = 10\%\)
- While all three enhancements are in place in the new design, each enhancement affects a different portion of the code.
- What is the resulting overall speedup?

\[
\text{Speedup} = \frac{1}{(1 - \sum F_i) + \sum \left(\frac{F_i}{S_i}\right)}
\]

\[
\text{Speedup} = 1 / [(1 - .2 - .15 - .1) + .2/10 + .15/15 + .1/30]
\]
\[
= 1 / [ .55 + .0333 ]
\]
\[
= 1 / .5833 = 1.71
\]

Pictorial Depiction of Example

Before:
Execution Time with no enhancements: 1

Unaffected, fraction: \(.55\)

Unchanged

Unaffected, fraction: \(.55\)

After:
Execution Time with enhancements: \(.55 + .02 + .01 + .00333 = .5833\)

Speedup = \(1 / .5833 = 1.71\)

Note: All fractions \((F_i, i = 1, 2, 3)\) refer to original execution time.
## Computer Performance Measures: MIPS Rating (1/3)

- For a specific program running on a specific CPU the MIPS rating is a measure of how many millions of instructions are executed per second:

\[
\text{MIPS Rating} = \frac{\text{Instruction count}}{(\text{Execution Time} \times 10^6)}
\]

\[
= \frac{\text{Instruction count}}{(\text{CPU clocks} \times \text{Cycle time} \times 10^6)}
\]

\[
= \frac{(\text{Instruction count} \times \text{Clock rate})}{(\text{Instruction count} \times \text{CPI} \times 10^6)}
\]

\[
= \frac{\text{Clock rate}}{(\text{CPI} \times 10^6)}
\]

- **Major problem with MIPS rating:** As shown above the MIPS rating does not account for the count of instructions executed (I).
  - A higher MIPS rating in many cases may not mean higher performance or better execution time. i.e. due to compiler design variations.

## Computer Performance Measures: MIPS Rating (2/3)

- In addition the MIPS rating:
  - **Does not account for the instruction set architecture (ISA) used.**
    - Thus it cannot be used to compare computers/CPUs with different instruction sets.
  - **Easy to abuse:** Program used to get the MIPS rating is often omitted.
    - Often the **Peak MIPS rating** is provided for a given CPU which is obtained using a program comprised entirely of **instructions with the lowest CPI** for the given CPU design which **does not represent real programs**.
Computer Performance Measures: MIPS Rating (3/3)

• Under what conditions can the MIPS rating be used to compare performance of different CPUs?

• The MIPS rating is only valid to compare the performance of different CPUs provided that the following conditions are satisfied:
  1. The same program is used (actually this applies to all performance metrics)
  2. The same ISA is used
  3. The same compiler is used

⇒ (Thus the resulting programs used to run on the CPUs and obtain the MIPS rating are identical at the machine code level including the same instruction count)

A MIPS Example (1)

• Consider the following computer:

<table>
<thead>
<tr>
<th>Instruction counts (in millions) for each instruction class</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code from:</td>
</tr>
<tr>
<td>Compiler 1</td>
</tr>
<tr>
<td>Compiler 2</td>
</tr>
</tbody>
</table>

The machine runs at 100MHz.

Instruction A requires 1 clock cycle, Instruction B requires 2 clock cycles, Instruction C requires 3 clock cycles.

\[ \text{CPI} = \frac{\sum_{i=1}^{n} \text{CPU Clock Cycles}}{\text{Instruction Count}} = \frac{\sum_{i=1}^{n} \text{CPI}_i \times C_i}{\text{Instruction Count}} \]

Note important formula!
A MIPS Example (2)

\[ CPI_1 = \frac{\left(5 \times 1\right) + \left(1 \times 2\right) + \left(1 \times 3\right)}{5 + 1 + 1} \times 10^6 = \frac{10}{7} = 1.43 \]

\[ MIPS_1 = \frac{100 \text{ MHz}}{1.43} = 69.9 \]

\[ CPI_2 = \frac{\left(10 \times 1\right) + \left(1 \times 2\right) + \left(1 \times 3\right)}{10 + 1 + 1} \times 10^6 = \frac{15}{12} = 1.25 \]

\[ MIPS_2 = \frac{100 \text{ MHz}}{1.25} = 80.0 \]

So, compiler 2 has a higher MIPS rating and should be faster?

A MIPS Example (3)

- Now let’s compare CPU time:

\[ CPU \text{ Time} = \frac{\text{Instruction Count} \times CPI}{\text{Clock Rate}} \]

\[ CPU \text{ Time}_1 = \frac{7 \times 10^6 \times 1.43}{100 \times 10^6} = 0.10 \text{ seconds} \]

\[ CPU \text{ Time}_2 = \frac{12 \times 10^6 \times 1.25}{100 \times 10^6} = 0.15 \text{ seconds} \]

Therefore program 1 is faster despite a lower MIPS!
Computer Performance Measures : MFLOPS (1/2)

- A floating-point operation is an addition, subtraction, multiplication, or division operation applied to numbers represented by a single or a double precision floating-point representation.
- MFLOPS, for a specific program running on a specific computer, is a measure of millions of floating point-operation (megaflops) per second:

\[
MFLOPS = \frac{\text{Number of floating-point operations}}{(\text{Execution time} \times 10^6)}
\]

- MFLOPS rating is a better comparison measure between different machines (applies even if ISAs are different) than the MIPS rating.
  - Applicable even if ISAs are different

Computer Performance Measures : MFLOPS (2/2)

- Program-dependent: Different programs have different percentages of floating-point operations present. i.e compilers have no floating-point operations and yield a MFLOPS rating of zero.
- Dependent on the type of floating-point operations present in the program.
  - Peak MFLOPS rating for a CPU: Obtained using a program comprised entirely of the simplest floating point instructions (with the lowest CPI) for the given CPU design which does not represent real floating point programs.
CPU Benchmark Suites

- **Performance Comparison**: the execution time of the same *workload* running on two machines without running the actual programs
- **Benchmarks**: the programs specifically chosen to measure the performance.
- **Five levels of programs**: in the decreasing order of accuracy
  - *Real Applications*
  - *Modified Applications*
  - *Kernels*
  - *Toy benchmarks*
  - *Synthetic benchmarks*

SPEC: System Performance Evaluation Cooperative

- **SPECCPU**: popular desktop benchmark suite
  - CPU only, split between integer and floating point programs
- **First Round 1989**: 10 programs yielding a single number – SPECmarks
- **Second Round 1992**: SPECInt92 (6 integer programs) and SPECfp92 (14 floating point programs)
- **Third Round 1995**
  - new set of programs: SPECInt95 (8 integer programs) and SPECfp95 (10 floating point)
  - “benchmarks useful for 3 years”
  - Single flag setting for all programs: SPECInt_base95, SPECfp_base95
- **SPECInt2000** has 12 integer, **SPECfp2000** has 14 integer pgms
- **SPECCPU2006** to be announced Spring 2006
- **SPECSFS** (NFS file server) and **SPECWeb** (WebServer) added as server benchmarks