Definition: Performance

- Performance is in units of things per second
  - bigger is better
- If we are primarily concerned with response time

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

"X is n times faster than Y" means

\[
n = \frac{\text{Performance}(X)}{\text{Performance}(Y)} = \frac{\text{Execution\_time}(Y)}{\text{Execution\_time}(X)}
\]
The bottom line: Performance

<table>
<thead>
<tr>
<th>Car</th>
<th>Time to Boston</th>
<th>mph</th>
<th>passengers</th>
<th>Throughput (person, mile/hour)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ferrari</td>
<td>3</td>
<td>150</td>
<td>2</td>
<td>100</td>
</tr>
<tr>
<td>Greyhound</td>
<td>7</td>
<td>65</td>
<td>60</td>
<td>560</td>
</tr>
</tbody>
</table>

- Time to do the task
  - execution time, response time, latency
- Tasks per day, hour, week, sec, ns ...
  - throughput, bandwidth

Comparing performance of two machines

- Definition: Performance is equal to 1 divided by execution time

- Problem: How to measure execution time?
What is time?

• Unix time command example:
  – 90.7u 12.9s 2:39 65%
  – The user used the CPU for 90.7 seconds (user CPU time)
  – The system used it for 12.9 seconds (system CPU time)
  – Elapsed time from the user’s request to completion of the task was 2 minutes, 39 seconds (159 seconds)
  – And (90.7 + 12.9)/159 = 65%
    » the rest of the time was spent waiting for I/O or running other programs

Time (cont.)

• Usual measurements of time:
  – system performance measures the elapsed time on unloaded (single user) system
  – CPU performance measures user CPU time on unloaded system
Relative Performance

\[
\text{Relative Performance} \left( \frac{X}{Y} \right) = \frac{\text{Performance}_X}{\text{Performance}_Y} \frac{\text{Execution Time}_Y}{\text{Execution Time}_X} = n
\]

- X is n times faster than Y
- X is n times as fast as Y
- From Y to X, speedup is n

What is time?

CPU Execution Time = CPU clock cycles * Clock cycle time

- Every conventional processor has a clock with an associated clock cycle time or clock rate
- Every program runs in an integral number of clock cycles
- Cycle Time
  - MHz = millions of cycles/second, GHz = billions of cycles/second
  - X MHz = 1000/X nanoseconds cycle time
  - Y GHz = 1/Y nanoseconds cycle time
How many clock cycles?

Number of CPU cycles = Instructions executed * Average Clock Cycles per Instruction (CPI)

Example: Computer A runs program C in 3.6 billion cycles. Program C consists of 2 billion dynamic instructions. What is the CPI?

\[
\text{3.6 billion cycles / 2 billion instruction} = 1.8 \text{ CPI}
\]

How many clock cycles?

Number of CPU cycles = Instructions executed * Average Clock Cycles per Instruction (CPI)

Example: A computer is running a program with CPI = 2.0, and executes 24 million instructions, how long will it run?

\[
24 \text{ millions instruction} * 2 \text{ CPI} = 48 \text{ millions cycles}
\]
Example

- IC = 1 billion, 500 MHz processor, execution time of 3 seconds. What is the CPI for this program?
  - \(3 \text{ s} = 1 \text{b} \times \text{CPI} \times \frac{1}{500 \text{M s}}\)  
    \[\text{CPI} = 1.5\]

- Suppose we reduce CPI to 1.2 (through an architectural improvement). What is the new execution time?
  - \(\text{Exe Time} = 1.2 \times 1 \times \frac{1}{500} \text{ s} = 2.4 \text{ s}\)
  - \(\frac{\text{Performance}_{\text{new}}}{\text{performance}_{\text{old}}} = \frac{3}{2.4} = 1.25\)

Who Affects Performance?

- Programmer
- Compiler
- instruction-set architect
- hardware designer
- materials scientist/physicist/silicon engineer
### Performance Variation

CPU Execution Time = Instruction Count × CPI × Clock Cycle Time

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Number of Instructions</th>
<th>CPI</th>
<th>Clock Cycle Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Same machine, different programs</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Same program, different machine, same ISA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Same program, different machines</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

### Performance Variation

CPU Execution Time = Instruction Count × CPI × Clock Cycle Time

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Number of Instructions</th>
<th>CPI</th>
<th>Clock Cycle Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Same machine, different programs</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>Same program, different machine, same ISA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Same program, different machines</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Performance Variation

**Formula:**

\[
\text{CPU Execution Time} = \text{Instruction Count} \times \text{CPI} \times \text{Clock Cycle Time}
\]

<table>
<thead>
<tr>
<th></th>
<th>Number of Instructions</th>
<th>CPI</th>
<th>Clock Cycle Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Same machine, different programs</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>Same program, different machine, same ISA</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Same program, different machines</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
</tbody>
</table>
Other Performance Metrics

- MIPS
  - Millions of Instructions Per Second = instruction count/ (exe time * 10^6)
  - Program-independent?
  - Deceptive

- FLOPS
  - Floating-point Operations Per Second
  - How does execution time depend on FLOPS?

Which Programs

- Real applications

- SPEC (System Performance Evaluation Cooperative)
  - Provides a common set of real applications along with strict guidelines for how to run them.
  - Provides a relatively unbiased means to compare machines.
How to measure CPU performance

- **Benchmark**: a program used to measure performance
  - real programs - what is reality?
  - kernels - loops in which most of time is spent in a real program
  - toy programs
  - synthetic programs
- **Fact**: Computer manufacturers tune their product to the popular benchmarks
  - "Your results may vary," unless you run benchmark programs and nothing else
  - See Figure 1.13 listing programs in the SPEC CPU2006 benchmark suite
Performance: What to measure

- Usually rely on benchmarks vs. real workloads
- To increase predictability, collections of benchmark applications, called benchmark suites, are popular
- **SPECCPU**: popular desktop benchmark suite
  - CPU only, split between integer and floating point programs
  - SPEC2006 has 12 integer, 17 floating-point C/C++/Fortran programs
  - SPECSFS (NFS file server) and SPECWeb (WebServer) added as server benchmarks
- **Transaction Processing Council** measures server performance and cost-performance for databases
  - TPC-C Complex query for Online Transaction Processing
  - TPC-W a transactional web benchmark
  - TPC-App application server and web services benchmark

### SPEC CPU 2006 (Integer)

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Language</th>
<th>Application Area</th>
</tr>
</thead>
<tbody>
<tr>
<td>400.perlbench</td>
<td>C</td>
<td>Programming Language</td>
</tr>
<tr>
<td>401.bzip2</td>
<td>C</td>
<td>Compression</td>
</tr>
<tr>
<td>403.gcc</td>
<td>C</td>
<td>C Compiler</td>
</tr>
<tr>
<td>429.mcf</td>
<td>C</td>
<td>Combinatorial Optimization</td>
</tr>
<tr>
<td>445.gobmk</td>
<td>C</td>
<td>Artificial Intelligence: Go</td>
</tr>
<tr>
<td>456.hmmer</td>
<td>C</td>
<td>Search Gene Sequence</td>
</tr>
<tr>
<td>458.sjeng</td>
<td>C</td>
<td>Artificial Intelligence: chess</td>
</tr>
<tr>
<td>462.libquantum</td>
<td>C</td>
<td>Physics / Quantum Computing</td>
</tr>
<tr>
<td>464.h264ref</td>
<td>C</td>
<td>Video Compression</td>
</tr>
<tr>
<td>471.omnetpp</td>
<td>C++</td>
<td>Discrete Event Simulation</td>
</tr>
<tr>
<td>473.astar</td>
<td>C++</td>
<td>Path-finding Algorithms</td>
</tr>
<tr>
<td>483.xalanckbmk</td>
<td>C++</td>
<td>XML Processing</td>
</tr>
</tbody>
</table>
### SPEC CPU 2006 (Floating Point)

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Language</th>
<th>Application Area</th>
</tr>
</thead>
<tbody>
<tr>
<td>410.bwaves</td>
<td>Fortran</td>
<td>Fluid Dynamics</td>
</tr>
<tr>
<td>416.gamess</td>
<td>Fortran</td>
<td>Quantum Chemistry</td>
</tr>
<tr>
<td>433.milc</td>
<td>C</td>
<td>Physics / QCD</td>
</tr>
<tr>
<td>434.zeusmp</td>
<td>Fortran</td>
<td>Physics / CFD</td>
</tr>
<tr>
<td>435.gromacs</td>
<td>C,Fortran</td>
<td>Molecular Dynamics</td>
</tr>
<tr>
<td>436.cactusADM</td>
<td>C,Fortran</td>
<td>Physics</td>
</tr>
<tr>
<td>437.leslie3d</td>
<td>Fortran</td>
<td>Fluid Dynamics</td>
</tr>
<tr>
<td>444.namd</td>
<td>C++</td>
<td>Biology / Molecular Dynamics</td>
</tr>
<tr>
<td>447.dealII</td>
<td>C++</td>
<td>Finite Element Analysis</td>
</tr>
<tr>
<td>450.soplex</td>
<td>C++</td>
<td>Linear Programming</td>
</tr>
<tr>
<td>453.povray</td>
<td>C++</td>
<td>Image processing / ray-tracing</td>
</tr>
<tr>
<td>454.calculix</td>
<td>C,Fortran</td>
<td>Structural mechanics</td>
</tr>
<tr>
<td>459.GemsFDTD</td>
<td>Fortran</td>
<td>Computational E&amp;M</td>
</tr>
<tr>
<td>465.tonto</td>
<td>Fortran</td>
<td>Quantum Chemistry</td>
</tr>
<tr>
<td>470.lbm</td>
<td>C</td>
<td>Fluid Dynamics</td>
</tr>
<tr>
<td>481.wrf</td>
<td>C,Fortran</td>
<td>Weather simulation</td>
</tr>
<tr>
<td>482.sphinx3</td>
<td>C</td>
<td>Speech recognition</td>
</tr>
</tbody>
</table>

### SPEC CPU Change Over Time (Int)

<table>
<thead>
<tr>
<th>SPEC2006 benchmark description</th>
<th>SPEC2006</th>
<th>SPEC2000</th>
<th>SPEC95</th>
<th>SPEC92</th>
<th>SPEC99</th>
</tr>
</thead>
<tbody>
<tr>
<td>GNU C compiler</td>
<td></td>
<td></td>
<td>gcc</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Interpreted string processing</td>
<td></td>
<td></td>
<td>perl</td>
<td>espresso</td>
<td></td>
</tr>
<tr>
<td>Combinatorial optimization</td>
<td></td>
<td></td>
<td>mcf</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Block-sorting compression</td>
<td></td>
<td></td>
<td>bzip2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Go game (AI)</td>
<td>go</td>
<td></td>
<td>vortex</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Video compression</td>
<td>h264avc</td>
<td>gimp</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Games/path finding</td>
<td>astar</td>
<td>ucin</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Search gene sequences</td>
<td>hammer</td>
<td>twolf</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Quantum computer simulation</td>
<td>libquantum</td>
<td>vortex</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Discrete event simulation</td>
<td>omnetpp</td>
<td>vpr</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Chess game (AI)</td>
<td>speng</td>
<td>crafty</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>XML parsing</td>
<td>xalanbank</td>
<td>parser</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**Reproducibility**

- Benchmarking is a laboratory experiment, and needs to be documented as fully as a well-run chemistry experiment
  - Identify each variable hardware component
  - Identify compiler flags and measure variability
  - Verify reproducibility and provide data for others to reproduce the benchmarking results
IMPROVING COMPUTER PERFORMANCE – AMDAHL’S LAW

How to make computers faster

- Make the common case faster!
- Example: Put more effort and funds into optimizing the hardware for addition than to optimize square root
- Amdahl's law quantifies this principle:
  - Define speedup as the time the task took originally divided by the time the task takes after improvement
Amdahl’s Law

- Suppose that the original task runs for 1 second so it takes $f$ seconds in the critical piece, and $1-f$ in other things
- Then the task on the improved machine will take only $f/s$ seconds in the critical piece, but will still take $1-f$ seconds in other things
- \[
    \text{speedup} = \frac{\text{old\_time}}{\text{new\_time}} = \frac{1}{(1-f) + \frac{f}{s}}
\]

Example 1

- Suppose we work very hard improving the square root hardware, and that our task originally spends 1% of its time doing square roots. Even if the improvement reduces the square root time to zero, the speedup is no better than
- \[
    \text{speedup} = \frac{1}{1 - f} = \frac{1}{.99} = 1.01
    \]
- And we might be better off putting our effort into a more important part of the task
Example 2

- Suppose that, for the same cost, we can speed up integer arithmetic by a factor of 20, or speed up floating point arithmetic by a factor of 2. If our task spends 10% of its time in integer arithmetic, and 40% of its time in floating point arithmetic, which should we do?

Example 2 (cont.)

- Option 1: speedup = \( \frac{1}{.9 + \frac{1}{20}} = 1.105 \)

- Option 2: speedup = \( \frac{1}{.6 + \frac{4}{2}} = 1.25 \)
CPU Performance

• More jargon ...

• Something new happens in a computer during every clock cycle
• The clock rate is what manufacturers usually advertise to indicate a chip's speed:
  – e.g., a 2.8GHz Pentium 4
• But how fast the machine is depends on the clock rate and the number of clock cycles per instruction (CPI)

CPU Performance (cont.)

• So total CPU time = instruction count (IC) times CPI divided by clock rate (MHz/GHz)
  – IC * CPI / GHz
• There’s an example on p. 50 that illustrates the overall effect
• Note: CPI is not a good measure of performance all by itself since it varies by type of instruction!
How to design a fast computer

• Amdahl’s law says put your effort into optimizing instructions that are often used.
• The locality of reference rule-of-thumb says that a program spends 90% of its time in 10% of the code, so we should put effort into taking advantage of this, through a memory hierarchy
• For data accesses, 2 types of locality
  – temporal
  – spatial

Take advantage of parallelism

• At digital design level
  – e.g., set associative caches (Chapter 2)
• At processor level
  – pipelining (Appendix C)
  – among instructions (Chapter 3)
• At system level
  – multiple cores (Chapter 4)
  – multiple CPUs (Chapter 5)
  – disks (Appendix D)
Define and quantify reliability (1/3)

- How decide when a system is operating properly?
- Infrastructure providers now offer Service Level Agreements (SLA) to guarantee that their networking or power service would be dependable
- Systems alternate between 2 states of service with respect to an SLA:
  1. **Service accomplishment**, where the service is delivered as specified in SLA
  2. **Service interruption**, where the delivered service is different from the SLA
- **Failure** = transition from state 1 to state 2
- **Restoration** = transition from state 2 to state 1
Define and quantify reliability (2/3)

- **Module reliability** = measure of continuous service accomplishment (or time to failure).
  - 2 metrics
    1. **Mean Time To Failure (MTTF)** measures Reliability
    2. **Failures In Time (FIT)** = 1/MTTF, the rate of failures
      - Traditionally reported as failures per billion hours of operation
    - **Mean Time To Repair (MTTR)** measures Service Interruption
      - **Mean Time Between Failures (MTBF)** = MTTF + MTTR
    - **Module availability** measures service as alternate between the 2 states of accomplishment and interruption (number between 0 and 1, e.g. 0.9)
      - **Module availability** = MTTF / (MTTF + MTTR)

Example calculating reliability

- If modules have *exponentially distributed lifetimes* (age of module does not affect probability of failure), overall failure rate is the sum of failure rates of the modules
- Calculate FIT and MTTF for 10 disks (1M hour MTTF per disk), 1 disk controller (0.5M hour MTTF), and 1 power supply (0.2M hour MTTF):
  
  \[ \text{FailureRate} = 10 \times \left( \frac{1}{1,000,000} \right) + 1 \times \frac{1}{500,000} + 1 \times \frac{1}{200,000} \]
  
  \[ = (10 + 2 + 5) / 1,000,000 \]
  
  \[ = 17 / 1,000,000 \]
  
  \[ = 17,000 \text{FIT} \]

  \[ \text{MTTF} = 1,000,000,000 / 17,000 \]
  
  \[ = 59,000 \text{hours} \]