Lecture 3: Architectural Performance Laws and Rules of Thumb

Prof. Randy H. Katz
Computer Science 252
Spring 1996
Measurement and Evaluation

Architecture is an iterative process:
• Searching the space of possible designs
• At all levels of computer systems
Measurement Tools

- Benchmarks, Traces, Mixes
- Cost, delay, area, power estimation
- Simulation (many levels)
  - ISA, RT, Gate, Circuit
- Queuing Theory
- Rules of Thumb
- Fundamental Laws
The Bottom Line: Performance (and Cost)

<table>
<thead>
<tr>
<th>Plane</th>
<th>DC to Paris</th>
<th>Speed</th>
<th>Passengers</th>
<th>Throughput (pmph)</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 Concodre</td>
<td>3 hours</td>
<td>1350 mph</td>
<td>132</td>
<td>178,200</td>
</tr>
</tbody>
</table>

- **Time to run the task (ExTime)**
  - Execution time, response time, latency
- **Tasks per day, hour, week, sec, ns ... (Performance)**
  - Throughput, bandwidth
The Bottom Line: Performance (and Cost)

"X is n times faster than Y" means

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

- Speed of Concorde vs. Boeing 747
- Throughput of Boeing 747 vs. Concorde
Performance Terminology

“X is n% faster than Y” means:

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

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

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

\[
\frac{\text{ExTime}(Y)}{\text{ExTime}(X)} = \frac{15}{10} = \frac{1.5}{1.0} = \frac{\text{Performance (X)}}{\text{Performance (Y)}}
\]

\[
n = \frac{100 (1.5 - 1.0)}{1.0}
\]

\[
n = 50\%
\]
Amdahl's Law

Speedup due to enhancement E:

\[
\text{Speedup}(E) = \frac{\text{ExTime w/o E}}{\text{ExTime w/ E}} = \frac{\text{Performance w/ E}}{\text{Performance w/o E}}
\]

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

\[
\text{ExTime}(E) = \\
\text{Speedup}(E) =
\]
Amdahl’s Law

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

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

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

\[
\text{ExTime}_{\text{new}} = \text{ExTime}_{\text{new}} \\
\text{Speedup}_{\text{overall}} = \text{Speedup}_{\text{overall}}
\]
Amdahl’s Law

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

\[ \text{ExTime}_{\text{new}} = \text{ExTime}_{\text{old}} \times (0.9 + \frac{1}{2}) = 0.95 \times \text{ExTime}_{\text{old}} \]

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

  ![Spatial Locality](image1)

  ![Temporal Locality](image2)

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

  ![Diagram of storage hierarchy](image3)
Occam's Toothbrush

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

• Do simple, fast things in hardware and be sure the rest can be handled correctly in software
Metrics of Performance

Application

Programming Language

Compiler

Datapath

Control

Function Units

Transistors Wires Pins

ISA

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)
Aspects of CPU Performance

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

<table>
<thead>
<tr>
<th></th>
<th>Instr. Cnt</th>
<th>CPI</th>
<th>Clock Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Compiler</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Instr. Set</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Organization</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Technology</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Marketing Metrics

**MIPS** = Instruction Count / Time * 10^6 = Clock Rate / CPI * 10^6

- 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>
</tr>
</thead>
<tbody>
<tr>
<td>add, sub, compare, mult</td>
</tr>
<tr>
<td>divide, sqrt</td>
</tr>
<tr>
<td>exp, sin, . . .</td>
</tr>
</tbody>
</table>

RHK.S96
Cycles Per Instruction

Average Cycles per Instruction”

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

\[ \text{CPU time} = \text{CycleTime} \times \sum_{i=1}^{n} CPI_i \times I_i \]

‘Instruction Frequency”

\[ CPI = \sum_{i=1}^{n} CPI_i \times F_i \quad \text{where} \quad F_i = \frac{I_i}{\text{Instruction Count}} \]

Invest Resources where time is Spent!
Organizational Trade-offs
Example: Calculating CPI

<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>

Typical Mix

\[
\text{CPI(i)} = 1.5
\]
Example

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