Two Notions of Performance

<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>Concorde</td>
<td>3 hours</td>
<td>1350 mph</td>
<td>132</td>
<td>178,200</td>
</tr>
</tbody>
</table>

- Which has higher performance?
- What is performance?
  - Time to completion (latency)? *This is our focus.*
  - Throughput?
- We’re concerned with performance, but there are other important metrics:
  - Cost
  - Power
  - Footprint
Amdahl’s Law

How much extra performance can you get if you speed up part of your program? There are two factors:

- How much better is it? (s)
- How often is it used? (p)

\[
\text{Speedup}(E) = \frac{\text{Exec Time without } E}{\text{Exec Time with } E} = \frac{\text{Perf with } E}{\text{Perf without } E}
\]

\[
\text{Exec Time}_{\text{new}} = \text{Exec Time}_{\text{old}} \times [(1 - p) + p/s]
\]

\[
\text{Speedup}(E) = \frac{\text{Exec Time}_{\text{old}}}{\text{Exec Time}_{\text{new}}} = \frac{1}{(1 - p) + p/s}
\]
Example 1

Suppose:
- Floating point instructions could be improved by 2X.
- But, only 10% of instructions are floating point.

\[
\text{Exec Time}_{new} = \text{Exec Time}_{old} \times [0.9 + 0.1/2] = 0.95 \times \text{Exec Time}_{old}
\]

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

Speedup is bounded by:

\[
\frac{1}{\text{fraction of time not enhanced}}
\]
Example 2

Assume you can parallelize some portion of your program to make it 100X faster.

How much faster does the whole program get?

\[ T_1 = T_0 \left[ (1 - p) + \frac{p}{S} \right] \]
Example 3

Suppose:

- Memory operations currently take 30% of execution time.
- A new L1 cache speeds up 80% of memory operations by a factor of 4.
- A second new L2 cache speeds up 1/2 of the remaining 20% by a factor of 2.

What is the total speedup?
Applying the two optimizations sequentially:

<table>
<thead>
<tr>
<th></th>
<th>L1</th>
<th>L2</th>
<th>Not memory</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory time</td>
<td>0.24</td>
<td>0.03</td>
<td>0.03</td>
<td>0.7</td>
</tr>
<tr>
<td>L1 sped up</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>L2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>24%</td>
<td>3%</td>
<td>3%</td>
<td>70%</td>
</tr>
</tbody>
</table>

|                | 0.06 | 0.03| 0.03       | 0.7   |
| L1 sped up     |      |     |            |       |
| L2             |      |     |            |       |
|                | 8.6% | 4.2%| 4.2%       | 85%   |

Speed up = 1.242
Make the common case fast!

Examples

- All instructions require instruction fetch, only some require data memory access. *Improve instruction fetch performance first.*
- Programs exhibit locality (spatial and temporal) and smaller memories are faster than larger memories.
  - Incorporate small, fast caches into processor design.
  - Manage caches to exploit locality.

Amdahl provided a quantitative basis for making these decisions.
Amdahl’s law does not bound slowdown!

Things can only get so fast, but they can get arbitrarily slow.

*Don’t do things that hurt the non-common case too much!*