Exercise: You have a task X with two component parts A and B, each of which takes 30 minutes. What is the latency of X?

Suppose that you can speedup part B by a factor of 2. What is the latency now? What is the overall speedup?

Answer: It really depends on whether A and B are concurrent or sequential. If sequential, you can answer using Amdahl’s Law.
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? \( (S, k) \)
- How often is it used? \( (\alpha) \)

\[
T_{new} = (1 - \alpha) T_{old} + (\alpha T_{old})/k = T_{old}[(1 - \alpha) + \alpha/k]
\]

\[
S = \frac{T_{old}}{T_{new}} = \frac{1}{(1 - \alpha) + \alpha/k}
\]

Example 1

Your pipeline has one very slow stage that consumes 60% of the time to process an instruction. You discover that you can speed it up by a factor of 3. What is the improvement in the latency?

\[
S = \frac{T_{old}}{T_{new}} = \frac{1}{(1 - \alpha) + \alpha/k}
\]

Hence,

\[
S = \frac{1}{(1 - 0.6) + 0.6/3} = 1.67X
\]

Suppose you could make that stage arbitrarily fast. How would that improve latency?

Example 2

Suppose:

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

\[
T_{new} = T_{old} \times [0.9 + 0.1/2] = 0.95 \times T_{old}
\]

\[
S_{total} = \frac{1}{0.95} = 1.053
\]

Speedup is bounded by:

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

Example 3

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

How much faster does the whole program get?

\[
T_{1} = T_{0} \left[ (1 - p) + \frac{p}{S} \right]
\]

Speedup vs. Vector Fraction

Fraction of code vectorizable

Speedup

0 10 20 30 40 50 60 70 80 90 100
0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1
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?

### Example 4 Answer

Applying the two optimizations sequentially:

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

<table>
<thead>
<tr>
<th></th>
<th>L1 Speed up</th>
<th>L2</th>
<th>Not memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>Speed up</td>
<td>0.06</td>
<td>0.03 0.03</td>
<td>0.7</td>
</tr>
<tr>
<td>Total</td>
<td>8.6% 4.2% 4.2%</td>
<td>85%</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>L1 Speed up</th>
<th>L2</th>
<th>Not memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>Speed up</td>
<td>0.015</td>
<td>0.03</td>
<td>0.7</td>
</tr>
<tr>
<td>Total</td>
<td>0.805</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Speed up = 1.242

### Summary Message

**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 Non-Corollary

**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!*