Amdahl’s Law

How is system performance altered when some component is changed?

Example 1:
Program execution time is made up of 75% CPU time and 25% I/O time. Which is the better enhancement:
(a) Increasing the CPU speed by 50% or (b) reducing I/O time by half?

Execution model: No overlap between CPU and I/O operations

Program execution time \( T = T_{\text{cpu}} + T_{\text{io}} \)

\( T_{\text{cpu}} / T = 0.75 \) and \( T_{\text{io}} / T = 0.25 \)
Amdahl’s Law

(a) Increasing the CPU speed by 50%

Program execution time \( T = T_{\text{cpu}} + T_{\text{io}} \) \( T_{\text{old}} = T \)

\[
T_{\text{cpu}} / T = 0.75 \\
T_{\text{io}} / T = 0.25
\]

Program execution time \( T_{\text{new}} = T_{\text{cpu}} / 1.5 + T_{\text{io}} \)

\[
T_{\text{new}} = T_{\text{cpu}} / 1.5 + T_{\text{io}} = 0.75 T / 1.5 + 0.25T = 0.75T
\]

For a 50% improvement in CPU speed: Execution time decreases by 25%

\[
\text{Speedup} = T_{\text{old}} / T_{\text{new}} = T / 0.75T = 1.33
\]
Amdahl’s Law

(b) Halve the IO Time

Program execution time $T = T_{cpu} + T_{io}$  $T_{old} = T$

$T_{cpu} / T = 0.75$

$T_{io} / T = 0.25$

Program execution time $T_{new} = T_{cpu} + T_{io} / 2$

$T_{new} = 0.75 T + 0.25T / 2 = 0.875T$

For a 100% improvement in IO speed: Execution time decreases by 12.5%

Speedup = $T_{old} / T_{new} = T / 0.875T = 1.14$
Amdahl’s Law

Limiting Cases

• CPU speed improved infinitely so $T_{CPU}$ tends to zero
  $T_{new} = T_{IO} = 0.25T$  Speedup limited to 4

• IO speed improved infinitely so $T_{IO}$ tends to zero
  $T_{new} = T_{CPU} = 0.75T$  Speedup limited to 1.33
Amdahl’s Law

Example 2: Parallel Programming (Multicore execution)

A program made up of 10% serial initialization and finalization code. The remainder is a fully parallelizable loop of $N$ iterations.

```
for (j = 0; j < N; j++) {
    a[j] = b[j] + c[j];
    d[j] = d[j] * c;
}
```

$$T = T_{INIT} + T_{LOOP} + T_{FINAL} = T_{SERIAL} + T_{LOOP}$$
Amdahl’s Law
Each iteration can be executed in parallel with the other iterations
Assuming p = 4

```c
for (j = 0; j < 25; j++)  {
    a[j] = b[j] + c[j];
    d[j] = d[j] * c;
}
for (j = 25; j < 50; j++)  {
    a[j] = b[j] + c[j];
    d[j] = d[j] * c;
}
for (j = 50; j < 75; j++)  {
    a[j] = b[j] + c[j];
    d[j] = d[j] * c;
}
for (j = 75; j < 100; j++)  {
    a[j] = b[j] + c[j];
    d[j] = d[j] * c;
}
```
Amdahl’s Law

Example 2: Parallel Programming (Multicore execution)

```
for (j = 0; j < 25; j++) {
    a[j] = b[j] + c[j];
    d[j] = d[j] * c;
}
```

```
for (j = 25; j < 50; j++) {
    a[j] = b[j] + c[j];
    d[j] = d[j] * c;
}
```

```
for (j = 50; j < 75; j++) {
    a[j] = b[j] + c[j];
    d[j] = d[j] * c;
}
```

```
for (j = 75; j < 100; j++) {
    a[j] = b[j] + c[j];
    d[j] = d[j] * c;
}
```
Amdahl’s Law

Performance Model

Assume

- System Calls for FORK/JOIN incur zero overhead
- Execution time for parallel loop scales linearly with the number of iterations in the loop
  - With $p$ processors executing the loop in parallel
    - Each processor executes $N/p$ iterations
      - Parallel time for executing the loop is: $T_{\text{LOOP}} / p$

Sequential time: $T_{\text{SEQ}} = T$

$$T = T_{\text{SERIAL}} + T_{\text{LOOP}}$$

$T_{\text{SERIAL}} = 0.1 \, T$
$T_{\text{LOOP}} = 0.9 \, T$

Parallel Time with $p$ processors:

$$T_p = T_{\text{SERIAL}} + T_{\text{LOOP}} / p$$

$$= 0.1T + 0.9T/p$$
Amdahl’s Law

Performance Model

Parallel Time with $p$ processors:

$$T_p = T_{\text{SERIAL}} + \frac{T_{\text{LOOP}}}{p}$$

$$T_p = 0.1T + \frac{0.9T}{p}$$

$p = 2$: $T_p = 0.1T + \frac{0.9T}{2} = 0.55T$  Speedup $= \frac{T}{0.55T} = 1.8$

$p = 4$: $T_p = 0.1T + \frac{0.9T}{4} = 0.325T$  Speedup $= \frac{T}{0.325T} = 3.0$

$p = 8$: $T_p = 0.1T + \frac{0.9T}{8} = 0.2125T$  Speedup $= \frac{T}{0.2125T} = 4.7$

$p = 16$: $T_p = 0.1T + \frac{0.9T}{16} = 0.15625T$  Speedup $= \frac{T}{0.15625T} = 6.4$

**Limiting Case:**  $p$ so large that $T_{\text{LOOP}}$ is negligible (assume 0)

$$T_p = 0.1T$$  and Maximum Speedup is 10!!

Program with a fraction $f$ of serial (non-parallelizable) code will have a maximum speedup of $1/f$
Amdahl’s Law

Diminishing Returns

- Adding more processors leads to successively smaller returns in terms of speedup
- Using 16 processors does not result in an anticipated 16-fold speedup
- The Non-parallelizable sections of code takes a larger percentage of the execution time as the loop time is reduced
- Maximum Speedup is theoretically limited by fraction f of serial code
  - So even 1% serial code implies speedup of 100 at best!

Q: In the light of this pessimistic assessment:
   Why is multicore alive and well and even becoming the dominant paradigm?
Amdahl’s Law

Why is multicore alive and well and even becoming the dominant paradigm?

1. **Throughput Computing**: Run large numbers of independent computations (e.g. Web or Database transactions) on different cores

2. **Scaling Problem Size**:
   - Use parallel processing to solve larger problem sizes in a given amount of time
   - Different from solving a small problem even faster

In many situations scaling the problem size (N in our example) does not imply a proportionate increase in the serial portion.

Serial fraction f drops as problem size is increased

**Examples**:
- Opening a file is a fixed serial overhead independent of problem size
  - The fraction it represents decreases as the problem size is increased
- Parallel IO is routinely available today while it used to be a serialized overhead
- Sophisticated parallel algorithms / compiler techniques are able to parallelize what used to be considered intrinsically serial in the past
Amdahl’s Law Summary

- How is **system performance** altered when some **component** of the design is changed?

- **Performance Gains (Speedup)** by enhancing some **design feature**
  - **Base design time:** $T_{\text{base}}$
  - Several design components $C_1, C_2 \ldots C_n$
  - Component $C_k$ takes fraction $f_k$ of the total time
  - Suppose $C_k$ speeded up by factor $S$; others remain the same
  - **Enhanced design time:** $T_{\text{enhanced}}$

<table>
<thead>
<tr>
<th></th>
<th>Base Design</th>
<th>Enhanced Design</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time for $C_k$</td>
<td>$T_{\text{base}} \times f_k$</td>
<td>$T_{\text{base}} \times f_k / S$</td>
</tr>
<tr>
<td>Time for rest</td>
<td>$T_{\text{base}} \times (1 - f_k)$</td>
<td>$T_{\text{base}} (1 - f_k)$</td>
</tr>
<tr>
<td>Total Time</td>
<td>$T_{\text{base}}$</td>
<td>$T_{\text{base}} (f_k / S + 1 - f_k)$</td>
</tr>
</tbody>
</table>

\[
\text{Speedup} = \frac{T_{\text{base}}}{T_{\text{enhanced}}} = \frac{T_{\text{base}}}{T_{\text{base}} (f_k / S + 1 - f_k)}
\]

\[
= \frac{1}{( (1 - f_k) + f_k / S )}
\]

- As $S$ becomes large Speedup tends to $1/(1-f)$ asymptotically