Speedup

- An application runs in time $T_1$ on one processor, $T_p$ for $p$ processors.
- Measured speedup on $p$ processors: $\frac{T_1}{T_p}$
- Can we bound/estimate speedup without running the application?

Amdahl’s Law

- Given $f \in [0, 1]$ the sequential fraction of an application:
  $$T_p \geq \frac{(1-f)T_1}{p} + f \cdot T_1$$

Scaling Models – Intuition

Consider the following application:

```c
auto A = InitializeArray(random_seed, ... ++j)
C[i][j] = Compute(B[i][j]);
WriteResults(C);
Sequential
Sequential
Sequential
Parallel
Parallel
```

Scaling Models – Intuition (revisited)

Consider the following application:

```c
auto A = InitializeArray(random_seed, ... ++j)
C[i][j] = Compute(B[i][j]);
WriteResults(C);
Sequential
Sequential
Sequential
Parallel
Parallel
```

Amdahl and Gustafson’s law

$$T_p \geq \frac{(1-f)T_1}{p} + f \cdot T_1$$
Amdahl’s Law

- Given \( f \in [0, 1] \) the sequential fraction of an application:
  \[ T_p \geq \frac{(1 - f) T_1}{p} + f \cdot T_1 \]

- Speedup can be modeled as:
  \[ S_p \leq \frac{T_1}{T_p} \leq \frac{1}{1 - f + f \cdot p} \]

- And efficiency:
  \[ E_p = \frac{S_p}{p} \leq \frac{1}{1 - f + f \cdot p} \]

Amdahl’s Law: Runtime

- As \( p \to \infty \):
  \[ T_\infty \geq f \cdot T_1 \]

  \[ S_\infty \leq \frac{1}{f} \]

  \[ E_\infty = 0 \quad (\text{if } f \neq 0) \]

Amdahl’s Law is Pessimistic

- Fixes the problem size

- More processors \( \Rightarrow \) Larger problems

- \( f \) is not a constant fraction, but \( f(n) \)
  - The same applies for \( T_1 \) and \( T_1(n) \)

Question

Is Amdahl’s law pessimistic or optimistic?
Case Study: Weather Forecasting

- Grid point climate/atmospheric models
  - GFS
  - GME
  - COSMO

- The denser the grid, the more accurate prediction becomes

- Important for anticipating natural disasters!

Gustafson’s Law

- States:
  \[ S_\infty \leq \frac{1}{f(n)} \rightarrow \infty \] \[ \text{if } f(n) \rightarrow 0 \]

→ Larger problems can be solved (with more resources) at the same time.

Strong and Weak Scaling

- **Strong scaling:**
  - Behavior of \( S_p(n) \) for a fixed \( n \), and \( p \rightarrow \infty \)
    
    Fixed problem size as \( p \) increases

- **Weak scaling:**
  - Behavior of \( S_p(n) \) for \( n, p \rightarrow \infty \)
    
    Increasing problem size as \( p \) increases

Pessimism/Optimism Revisited

Could both models be optimistic?

Other Issue: Load Balancing

- Amdahl’s Law assumes that computation divides evenly
- Processor waiting time is another form of overhead

Runtime With Parallel Overhead

<table>
<thead>
<tr>
<th>#Processors/Cores</th>
<th>Runtime</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><img src="image.png" alt="Image of runtime with parallel overhead" /></td>
</tr>
</tbody>
</table>
Runtime with Parallel Overhead and Work Imbalance

Real Speedup Graphs often Look like This:

Additional Resources
- Assumption in both laws: increasing $p$ does not change other resources (e.g. cache)
- It is thus possible to achieve super-linear speedup:
  - Example 1: Cache size increase leads to larger working set
  - Example 2: Probabilistic convergent algorithms

Amdahl’s Law: Summary
- Strong, simple models for scaling
- Ignore overhead incurred by parallelization and load imbalance
- In reality: $S_p(n) = \frac{T_1(n)}{T_p(n) + A_p(n)}$
- Programs are not always that simple
  - $f$ does not always exist

PRAM Model
- Parallel Random Access Machine
- Abstract model for shared memory parallelism
- Any processor can access the memory at unit time
- What about access conflicts?
  - Exclusive read exclusive write (EREW)
  - Concurrent read exclusive write (CREW)
  - Concurrent read concurrent write (CRCW)

PRAM Model: Program Representation
- Programs represented as DAGs (Directed Acyclic Graphs)
  - Nodes: Unit-time operations
  - Edges: Dependencies
- We can now define:
  - $W(n) = \#nodes$
  - $D(n) =$longest path from input to output
  - Average parallelism: $W(n)/D(n)$
**PRAM Model: Advantages**

- Powerful tool for analyzing parallel algorithms
- No need to manage communication and synchronization
  - Focus on algorithmic aspects
- Time-unit and processor abstraction is robust w.r.t. architectures

**PRAM Model: Examples**

- **Merge-sort:**
  ```
  def MergeSort(L):  
    if len(L) > 1:  
      mid = len(L) // 2  
      L1 = MergeSort(L[:mid])  
      L2 = MergeSort(L[mid:])  
      return merge(L1, L2)  
  ```

- **Scan (prefix sum):**
  ```
  def Scan(L):  
    W(n) = \Theta(n\log n)  
    D(n) = D\left(\frac{n}{2}\right) + \Theta(n) = \Theta(n)  
    Parallellism = \Theta(1)  
  ```

- **Binary tree:**
  ![Binary tree diagram]

**Reasoning in PRAM**

- **Given a DAG with \(W(n)\) nodes and \(D(n)\) depth:**
  - Sequential runtime: \(T_S(n) = W(n)\)
  - Time on \(p\) processors: \(T_P(n) = D(n)\)
  - Time on \(p\) processors: \(T_P(n) = ?\)

- Time bound: \(T_P(n) \geq D(n)\), \(T_P(n) \geq W(n)/p\)

- Can we bound \(T_P\) from above?
Brent's Theorem

\[ T_p(n) \leq D(n) + \frac{W(n) - D(n)}{p} \]

- Intuition: Bound parallelism at each level of the DAG

Brent's Theorem: Proof

- Observe: a DAG of depth D can be partitioned to D levels, s.t. there are no interdependencies between nodes at the same level.

  - At each DAG level \( i \), we define \( T_p^i \) as the time for layer \( i \) and \( p \) processors, \( s_i (i = 1 \ldots D) \) denotes the number of nodes in level \( i \), thus:
  
  \[ T_p^i = \frac{s_i}{p} \leq \frac{s_i + p - 1}{p} \]

  - It follows that:
  
  \[ T_p(n) \leq \sum_{i=1}^{D} \frac{s_i + p - 1}{p} \]

  \[ T_p(n) \leq \frac{D + W(n) - D(n)}{p} \]

  \[ T_p(n) \leq \frac{D + W(n) - D(n)}{p} \]

  \[ \sum_{i=1}^{D} s_i = W \]

Speedups

- \( S_p(n) = \frac{T_p(n)}{T_1(n)} \leq \frac{W(n)}{D(n)} \)
- \( S_p(n) \leq p \)
- \( S_p(n) \geq \frac{p}{W(n)D(n)} \rightarrow \frac{W(n)}{D(n)} \)

\[ S_{\infty}(n) = \frac{W(n)}{D(n)} \]

- For fixed \( n \), speedup is limited. For \( n \rightarrow \infty \), speedup can be unbounded
  - Example: Tree reduction

Communication Models: \( \alpha-\beta \) Model

- Gist: \( \alpha = \) Latency (cycles), \( \beta = \) Bandwidth (units/cycle)

  \[ \frac{T_1(n)}{p} \leq T_p(n) \leq \frac{W(n)}{p} + D(n) = \frac{T_1}{p} + T_{\infty} \]

- How long does it take to send a message of size \( n \)?
Little’s Law: Primer

- **Problem 1:**
  - In Tannenbar, every minute on average 2 students come and leave
  - Each student spends 8 minutes inside
  - Q: How many students are in Tannenbar at a given time?
    - A: 16

- **Problem 2:**
  - In your fridge, you have 150 bottles of Vivi Kola (on average)
  - You drink and buy on average 25 bottles per week
  - Q: How many weeks does every bottle last?
    - A: 6

Little’s Law and the α-β Model

- Queueing theory
  - In a stationary system (input rate = output rate), the number of customers \( N \) in a queue is given by:
    \[ N = \lambda W \]
  - Where \( \lambda \) = arrival rate and \( W \) = average time spent in the system
  - Simple rule, yet powerful: \( N \) is independent of I/O distribution
  - Connection: in the \( \alpha-\beta \) model, arrival rate is \( \beta \) and time spent in system is \( \alpha \) cycles

Example: Memory Systems

- **Latency** × **Throughput** = **Concurrency**
  - Latency is reduced by half every 9 years
  - Throughput doubles every 3 years
  - Parallel processing beneficial?

- Real-world statistics:
  - Intel Core 2 (2006): \( \alpha = 100 \) cycles, \( \beta = 2 \) bytes/cycle \( \rightarrow N = 200 \)
  - Intel Haswell (2014): \( \alpha = 63 \) cycles, \( \beta = 23 \) bytes/cycle \( \rightarrow N = 1449 \)