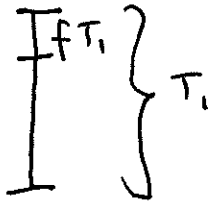
Reasoning about performance

Amdahl's Law (1967)

A program runs in time T_1 one one processor. A fraction f, $0 \le f \le 1$, of it is sequential. Let T_p be the runtime on p processors. Then



$$T_p \geq \frac{(1-f)T_1}{p} + fT_1$$

Speedup: $S_p = \frac{T_1}{T_p} \leq \frac{1}{\frac{1-f}{p} + f}$ ("linear" speedup means $S_p \approx p$)

Efficiency: $E_p = \frac{S_p}{p} \leq \frac{1}{1-f+fp}$ ($S_p \approx p \Rightarrow E_p \approx 1$)

$p \to \infty$: $T_\infty \geq fT_1$

$S_\infty \leq \frac{1}{f}$

$E_\infty = 0$ if $f \neq 0$

Is Amdahl's Law optimistic or pessimistic?

Pessimistic

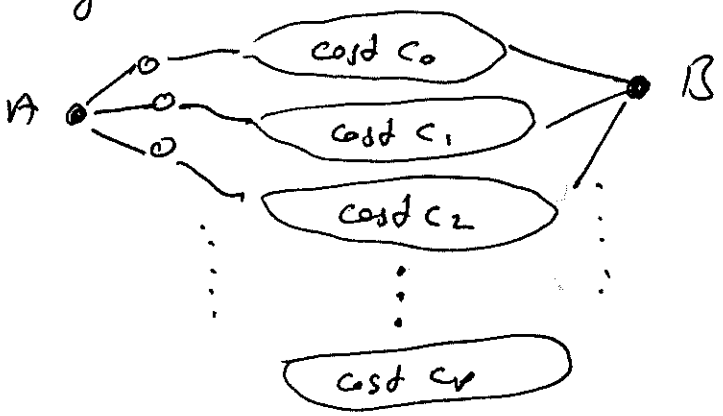
- AL fixes the problem size n. But with more processors people will want to solve larger problems. Thus take n into account above: $T_1(n)$, $f(n)$, ...
  For example (Gustafson's Law, 1988):

$$S_p(n) \leq \frac{1}{f(n)} \xrightarrow[n\to\infty]{} \infty \quad \text{if } f(n) \to 0$$

(sequential fraction decreases with problem size)

Terms:
- strong scaling: behavior of $S_p(n)$ for fixed n and $p \to \infty$
- weak scaling: " for $p, n \to \infty$

- AL assumes that by increasing p, all other resources remain constant. If this is not the case, superlinear speedup is possible
  - data caches private to a processor (for $p=2$, working set can suddenly fit into cache
  - memory bandwidth $p=2$ threads on 2 processors may get twice the bandwidth
- AL does not allow algorithmic superlinear speedup
  - variant of traveling salesman: go from A to B visiting all cities



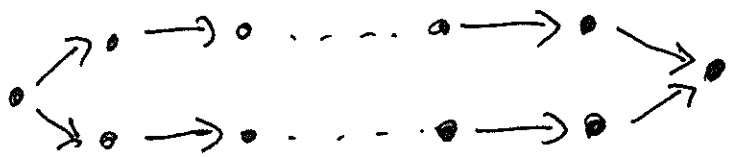
Sequential: $\sum c_i$
On p processors: $\min c_i$

Optimistic

- Ignores overhead of parallelization (e.g., creating threads) which increases with p.
- Assumes load balancing. In reality some processors idle.

So in reality: $S_p(n) = \dfrac{T_1(n)}{T_p(n) + A_p(n)}$ ← overhead

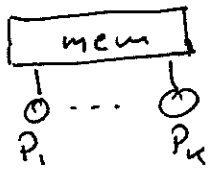
- Many programs have no sequential or infinitely parallelizable part. Example:



⇒ Need better models that take graph structure into account.

PRAM machine model & DAG-based computation model

PRAM:



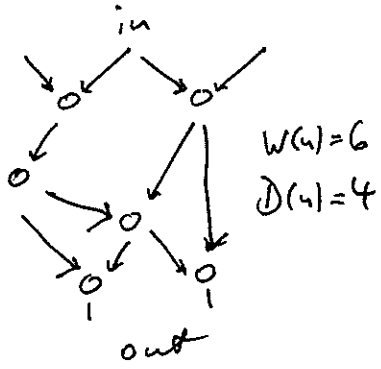
all processors can access every memory location simultaneously and in unit time

Computation as directed acyclic graph (DAG)

nodes: unit time operations

edges: dependencies

n: input size



$W(n)$ (work, op count, cost): number of nodes

$D(n)$ (depth, span, length of critical path): longest path between any input and output (measured in # nodes on that path)

Average parallelism: $\frac{W(n)}{D(n)}$

Examples:

1.) Reduction: $x_0 + x_1 + \ldots + x_{n-1}$ — average parallelism

sequential: $W(n) = O(n)$, $D(n) = O(n)$ — $O(1)$

binary tree: $W(n) = O(n)$, $D(n) = O(\log(n))$ — $O(\frac{n}{\log n})$

2.) Mergesort: L a list of length n

```
sort(L)
    if length(L) = 1 return L
    L1 = sort(left(L))
    L2 = sort(right(L))
    return merge(L1, L2)
```

$W(n) = O(n \log(n))$

$D(n) = D(\frac{n}{2}) + O(n) = O(n)$

average parallelism: $O(\log(n))$

Note: Parallel merge exists → shorter $D(n)$ → more parallelism

3.) Scan: input: list $x = (x_0, x_1, \ldots, x_{n-1})$

output: $(0, x_0, x_0+x_1, x_0+x_1+x_2, \ldots, x_0+\ldots+x_{n-2})$

sequential: $W(n) = D(n) = O(n)$

parallel: $x = (x_0, \dots, x_{n-1})$

```
scan(x)
    if length(x) = 1 return (0)
    sums = (x_0 + x_1, ..., x_{n-2} + x_{n-1})
    evens = scan(sums)      // (0, x_1 + x_2, x_1 + x_2 + x_3 + x_4, ...)
    odds = (evens[i] + x[2i] | i = 0 ... ⌈n/2 - 1⌉)
    return interleave(evens, odds)
```

$W(n) = W(\frac{n}{2}) + O(n) = O(n)$ average parallelism: $O(\frac{n}{\log n})$

$D(n) = D(\frac{n}{2}) + O(1) = O(\log n)$

Reasoning with W and D

Given a DAG with $W(n)$ nodes and depth $D(n)$.

Sequential runtime: $T_1(n) = W(n)$

Time on ∞ processors: $T_\infty(n) = D(n)$

" P " : $T_P(n) = ?$

$T_P(n) \geq D(n)$, $\geq \frac{W(n)}{P}$

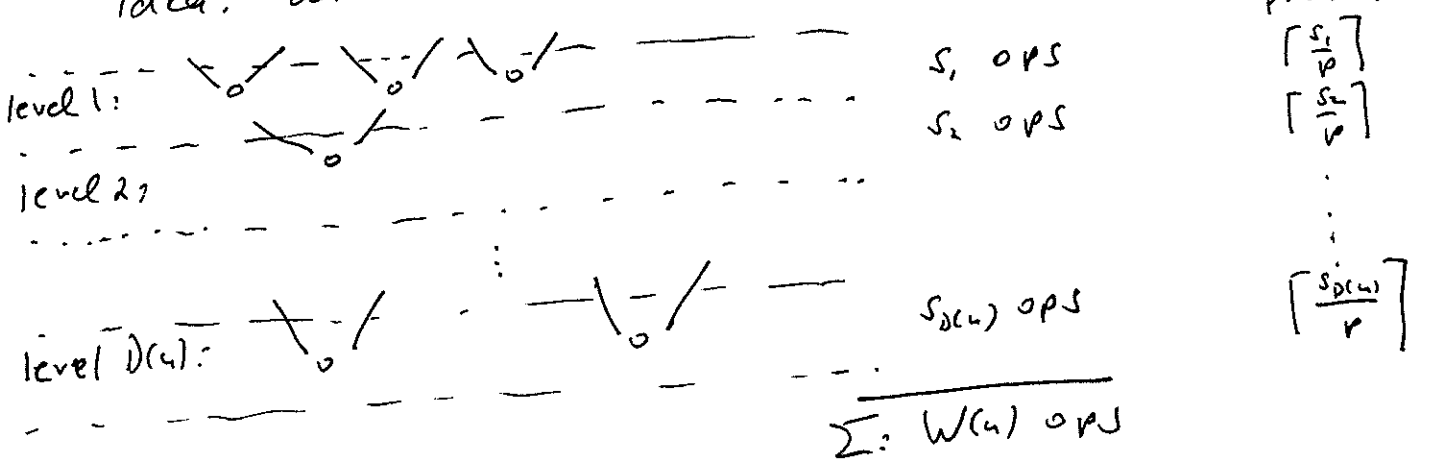
Brent's theorem (1974) $\rightarrow T_P(n) \leq D(n) + \frac{W(n) - D(n)}{P}$

$\leq D(n) + \frac{W(n)}{P}$ (approximation)

yields: $W(n)/P \leq T_P(n) \leq \frac{W(n)}{P} + D(n)$

Proof of Brent's theorem:

Idea: divide DAG into $D(n)$ levels:



$$T_P(n) \leq \sum_{i=1}^{D(n)} \lceil \frac{s_i}{P} \rceil \leq \sum_{i=1}^{D(n)} \frac{s_i + P - 1}{P} = \frac{1}{P} W(n) + D(n) \frac{P-1}{P}$$

$$= D(n) + \frac{W(n) - D(n)}{P}$$

↑ makes bound non-sharp

Note: level shifts (e.g. node from level i → level $i+1$) are not considered

Speedups: $S_p(n) = \frac{T_1(n)}{T_p(n)}$

$$S_p(n) \leq \frac{W(n)}{D(n)}, \quad \leq P$$

$$S_p(n) \geq \frac{P}{\frac{D(n)}{W(n)} P + 1} \xrightarrow{P \to \infty} \frac{W(n)}{D(n)}$$

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

If n is fixed, speedup is limited. For $n \to \infty$, speedup can be unbounded.

Strong scaling: Behaviour of $S_p(n)$ for fixed n and increasing p.

Weak scaling: Behaviour of $S_p(n)/p$ for increasing p and increasing n (~~explicitly~~ proportionally to p)

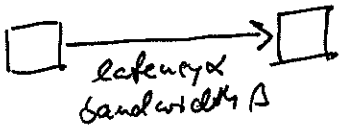
Example: LINPACK benchmark for TOP500 list
→ weak scaling

Example: tree reduction $W(n) = n$, $D(n) = \lceil \log_2(n) \rceil$

$$S_p(n) \geq \frac{P}{\frac{\lceil \log_2(n) \rceil}{n} P + 1} \xrightarrow{P \to \infty} \frac{n}{\lceil \log_2(n) \rceil}$$

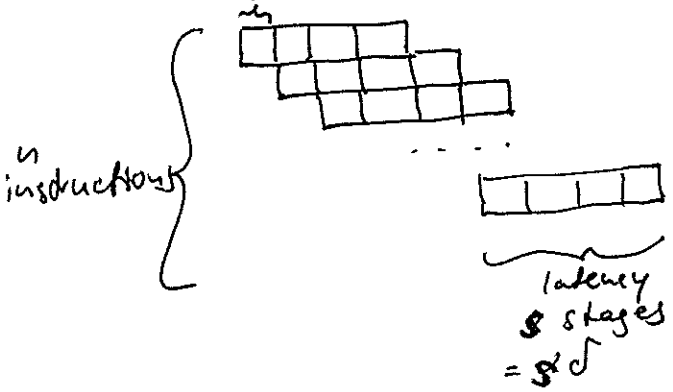
Next: reasoning about latency and throughput

α-β Model

Situation: 

How long does it take to send a message of size n?

Intuition: pipelining of instructions



τ = time per stage
s = # stages per instruction
$\beta = 1/\tau$ = bandwidth

Time for n instructions:

$$T(n) = (s-1)\tau + n \cdot \tau$$
$$= (s-1)\tau + \frac{n}{\beta}$$
$$\approx \text{latency} + \frac{n}{\text{bandwidth}}$$

Model: $T(n) = \alpha + \frac{n}{\beta}$