Complexity Measures for Parallel Computation

Complexity Measures for Parallel Computation

Problem parameters:

- n index of problem size
- p number of processors

Algorithm parameters:

- t_p running time on p processors
- t_1 time on 1 processor = sequential time = "work"
- t_∞ time on unlimited procs = critical path length = "span"
- v total communication volume

Performance measures

- speedup $s = t_1 / t_p$
- efficiency $e = t_1 / (p^*t_p) = s / p$
- (potential) parallelism $pp = t_1 / t_\infty$
- computational intensity $q = t_1 / v$

Several possible models!

- Execution time and parallelism:
  - Work / Span Model

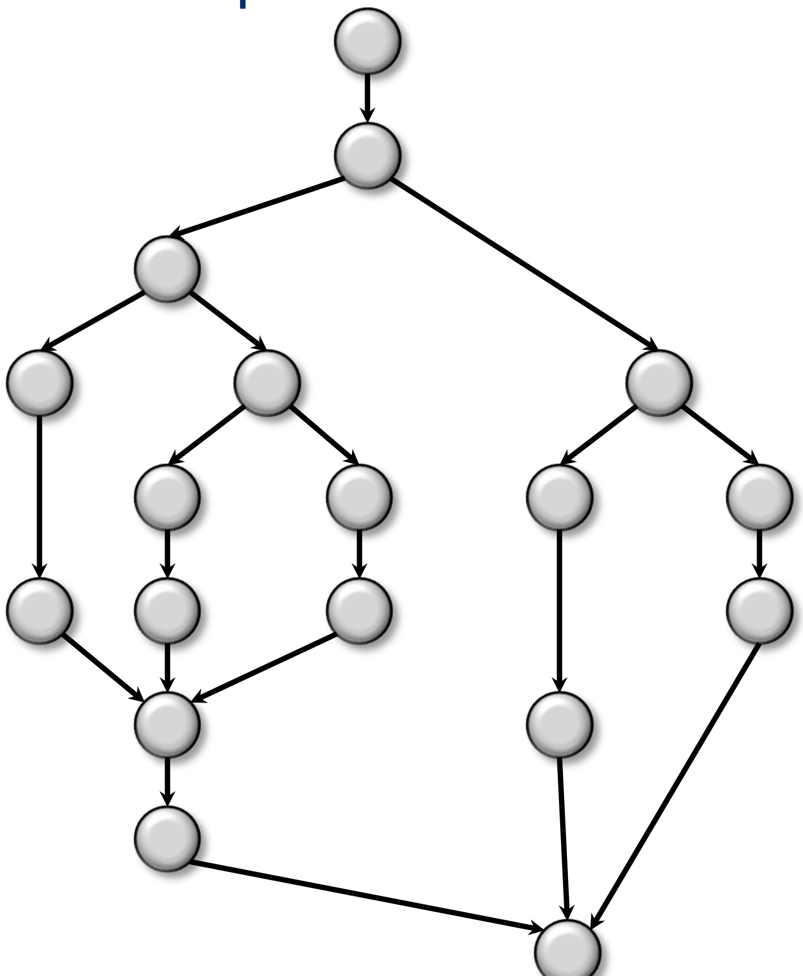
- Total <u>cost</u> of moving data:
  - Communication Volume Model

- Detailed models that try to capture <u>time</u> for moving data:
  - Latency / Bandwidth Model (for message-passing)
  - Cache Memory Model (for hierarchical memory)

  - Other detailed models we won't discuss: LogP, UMH, ….

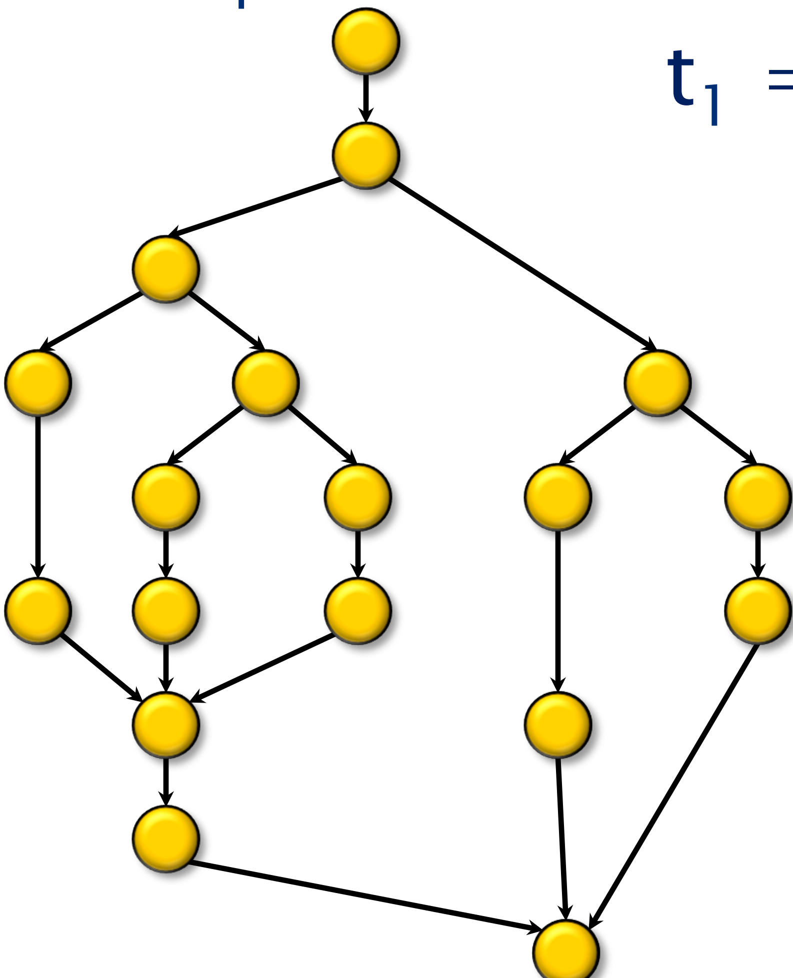
Work / Span Model

t_p = execution time on p processors



Work / Span Model

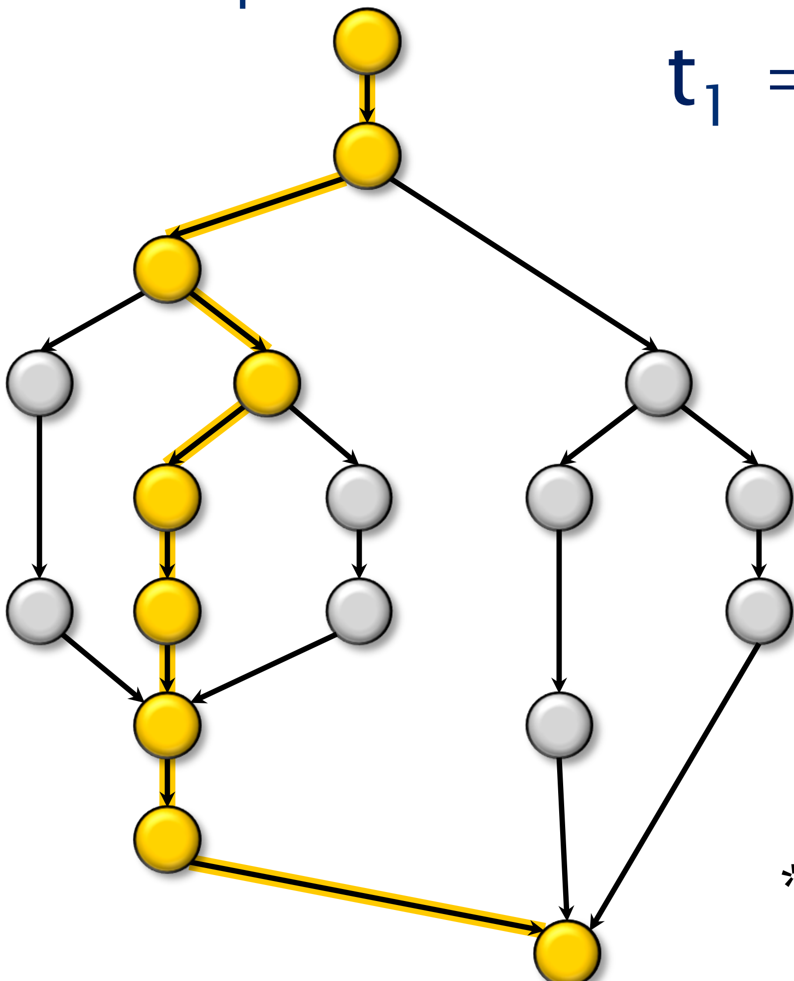
t_p = execution time on p processors



t_1 = *work*

Work / Span Model

t_p = execution time on p processors



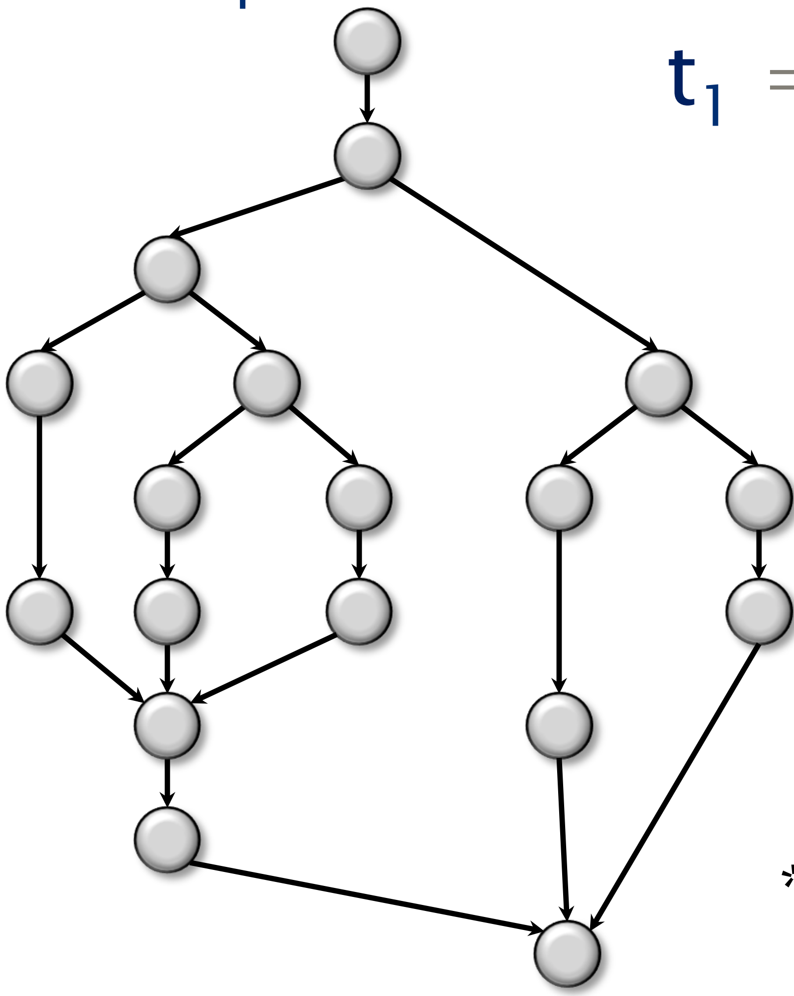
t_1 = *work*

t_∞ = *span* *

*Also called *critical-path length* or *computational depth*.

Work / Span Model

t_p = execution time on p processors



t_1 = ***work*** $\quad$ t_∞ = ***span*** *

WORK LAW

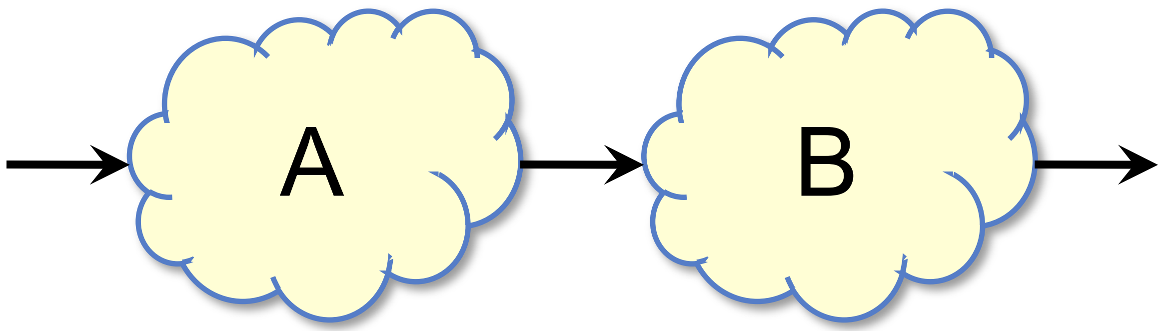
- $t_p \geq t_1 / p$

SPAN LAW

- $t_p \geq t_\infty$

*Also called *critical-path length* or *computational depth*.

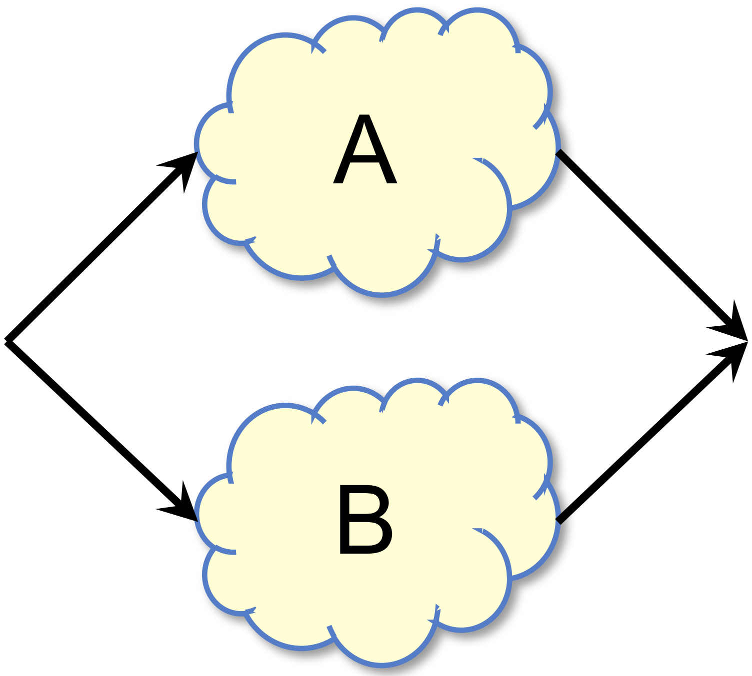
Series Composition



Work: $t_1(A \cup B) = t_1(A) + t_1(B)$

Span: $t_\infty(A \cup B) = t_\infty(A) + t_\infty(B)$

Parallel Composition



Work: $t_1(A \cup B) = t_1(A) + t_1(B)$

Span: $t_\infty(A \cup B) = \max\{t_\infty(A), t_\infty(B)\}$

Speedup

Def. $t_1 / t_P =$ ***speedup*** on p processors.

If t_1 / t_P $= \Theta(p)$, we have *linear speedup*,

$= p$, we have *perfect linear speedup*,

$> p$, we have *superlinear speedup*,

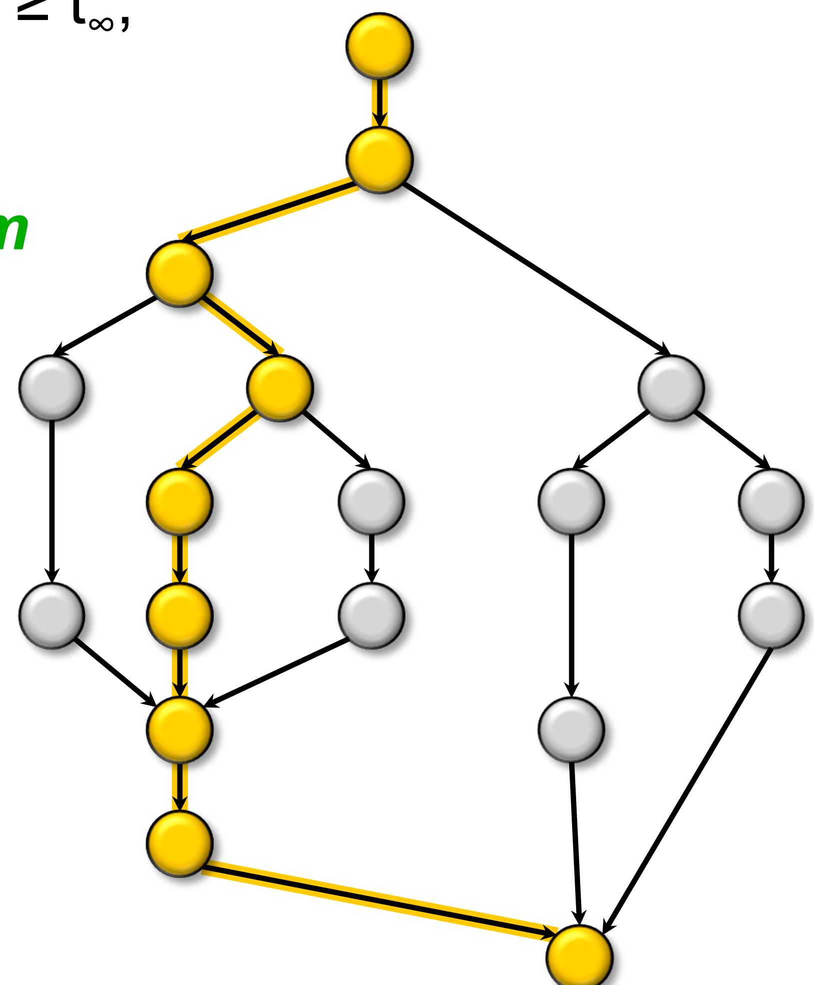
(which is not possible in this model, because of the Work Law $t_p \geq t_1 / p$)

Parallelism

Because the Span Law requires $t_p \geq t_\infty$, the maximum possible speedup is

t_1/t_∞ = ***(potential) parallelism***

= the average amount of work per step along the span.



Laws of Parallel Complexity

- Work law: $t_p \geq t_1 / p$
- Span law: $t_p \geq t_\infty$
- Amdahl's law:
  - If a fraction f, between 0 and 1, of the work must be done sequentially, then

    $$\text{speedup} \leq 1 / f$$

  - Exercise: prove Amdahl's law from the span law.

Communication Volume Model

- Network of p processors
  - Each with local memory
  - Message-passing
- Communication volume (v)
  - Total size (words) of all messages passed during computation
  - Broadcasting one word costs volume p (actually, p-1)
- No explicit accounting for communication time
  - Thus, can't really model parallel efficiency or speedup;
    for that, we'd use the latency-bandwidth model (see later slide)

Complexity Measures for Parallel Computation

Problem parameters:

- n index of problem size
- p number of processors

Algorithm parameters:

- t_p running time on p processors
- t_1 time on 1 processor = sequential time = "work"
- t_∞ time on unlimited procs = critical path length = "span"
- v total communication volume

Performance measures

- speedup $s = t_1 / t_p$
- efficiency $e = t_1 / (p*t_p) = s / p$
- (potential) parallelism $pp = t_1 / t_\infty$
- computational intensity $q = t_1 / v$

Detailed complexity measures for data movement I: Latency/Bandwith Model

<u>Moving data between processors by message-passing</u>

- Machine parameters:
  - α or $t_{startup}$ latency (message startup time in seconds)
  - β or t_{data} inverse bandwidth (in seconds per word)
  - between nodes of Triton, $\alpha \sim 2.2 \times 10^{-6}$ and $\beta \sim 6.4 \times 10^{-9}$
- Time to send & recv or bcast a message of w words: $\alpha + w^*\beta$
- t_{comm} total commmunication time
- t_{comp} total computation time
- Total parallel time: $t_p = t_{comp} + t_{comm}$

Detailed complexity measures for data movement II: Cache Memory Model

Moving data between cache and memory on one processor:

- Assume just two levels in memory hierarchy, fast and slow
- All data initially in slow memory
  - m = number of memory elements (words) moved between fast and slow memory
  - t_m = time per slow memory operation
  - f = number of arithmetic operations
  - t_f = time per arithmetic operation, $t_f << t_m$
  - $q = f / m$ *(computational intensity)* flops per slow element access
- Minimum possible time = $f * t_f$ when all data in fast memory
- Actual time
  - $f * t_f + m * t_m = f * t_f * (1 + t_m/t_f * 1/q)$
- Larger q means time closer to minimum $f * t_f$