Understanding Parallelism

Parallelism

To find parallelism in a computation, we have to look at:

- **Task decomposition** - based on processing to be done
- **Data decomposition** - based on the data to be processed

The decomposition determines the **potential parallelism**.

Task dependency graph

We represent the decomposition of a computation as a **directed acyclic graph (DAG)**.

- Node represents a **task**
    - Its weight represents the amount of work to be done by that task
- Edge represents a **dependence**

Execution time

T_P represents the execution time of the computation using P identical processors.

A **sequential** execution takes $T_1 = \sum_{i=i}^{nodes} work\_done_i$.

The **critical path** is the longest series of tasks that HAVE to be done sequentially due to depedencies.

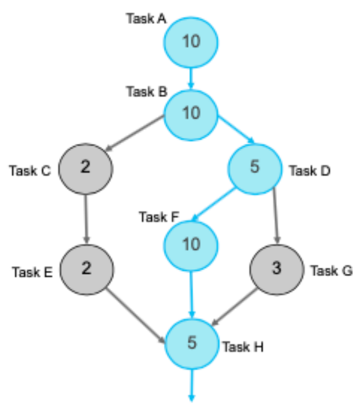
If we have infinite resources, $T_\infty = \sum_{i \in criticalpath} work\_done_i$

Computing the parallelism

$Parallelism = \frac{T_1}{T_\infty}$ and tells us how fast the computation would be if we had sufficient processors.

P_{min} is the minimum number of processors necessary to achieve $Parallelism$.

Example



- $T_1 = Tasks_{ABCDEFGH} = 47$
- Possible paths:

  $Tasks_{ABCEH} = 29$
  $Tasks_{ABDFH} = 40$
  $Tasks_{ABDGH} = 33$

- $T_\infty = Tasks_{ABDFH} = 40$
- $Parallelism = 47/40 = 1.175$
- $P_{min} = 2$

Granularity

The **granularity** of a decomposition is determined by the size of each node in the graph.

Fine-grained vs coarse-grained tasks $\Rightarrow$ Trade-off between more parallelism or less overhead

It may determine **performance bounds**.

Speed-up

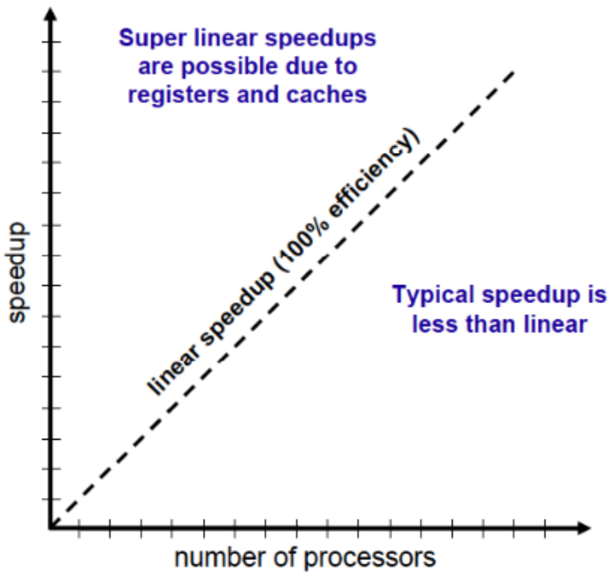
Speed-up on P processors is $S_P = \frac{T_1}{T_P}$.

Lower bounds:

- $T_P \geq \frac{T_1}{P}$
- $T_P \geq T_\infty$

Speed-up vs efficiency

Relative reduction in execution time when using P procesors with respect to the sequential execution.



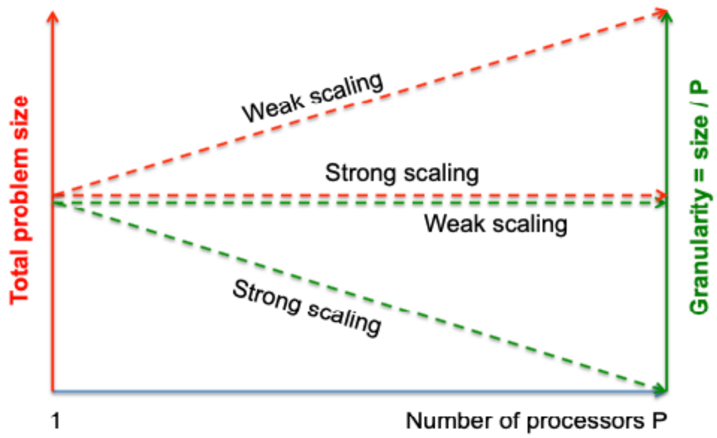
Efficiency: a measure of the fraction of time for which a processing unit is usefully employed.

$Eff_P = \frac{T_1}{T_P \times P} = \frac{S_P}{P}$

Strong vs weak scalability

Strong scalability: increase the number of processors P with constant problem size.

Weak scaling: increase the number of processors P with problem size proportional to P.



Amdahl's Law

The **performace improvement** to be gained from using some faster mode of execution is limited by the **fraction** of the time the faster mode can be used.

Parallel fraction: $\varphi = \frac{T_{par}}{T_1}$

We have that $T_P = (1 - \varphi) \times T_1 + (\varphi \times T_1/P)$, hence

$$S_P = \frac{T_1}{T_P} = \frac{T_1}{(1-\varphi) \times T_1 + (\varphi \times T_1/P)} = \frac{1}{((1-\varphi) + \varphi/P)}$$

Special cases:

- $\varphi = 0 \rightarrow S_P = 1$
- $\varphi = 1 \rightarrow S_P = P$
- $P \rightarrow \infty, S_P \rightarrow \frac{1}{1-\varphi}$

Overhead

Parallel computing is **not free**, we should account overheads.

Sources of overhead:

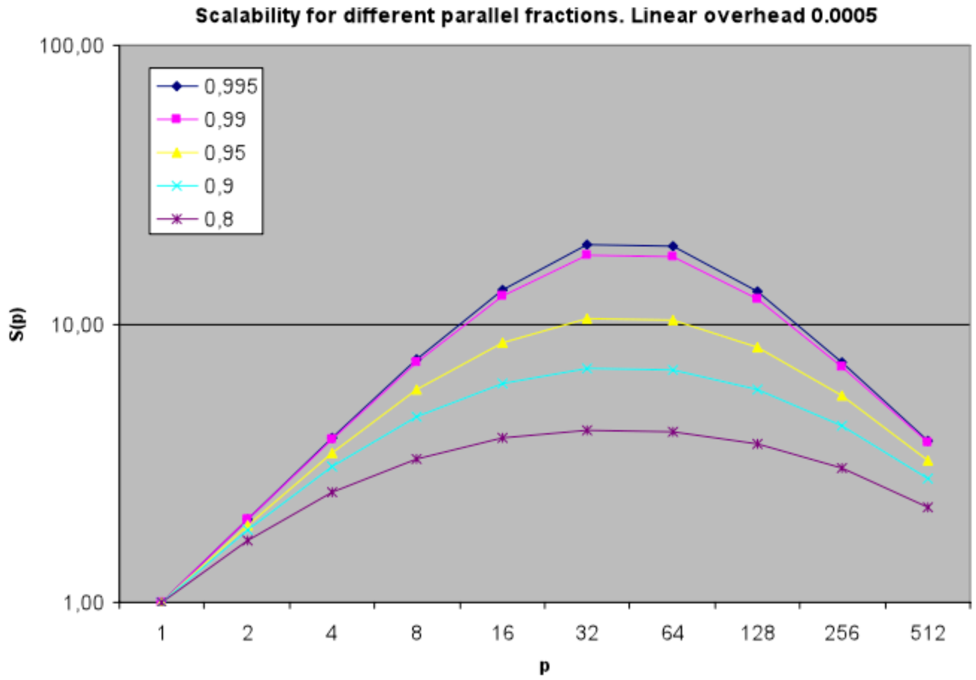
- **Task creation**
- **Task synchronisation**
- **Data sharing**
- **Idleness:** threads cannot find useful work to do
- **Computation:** extra work added to obtain parallel algorithm
- **Memory**
- **Contention:** competition for the access to shared resources

Amdahl's Law with **constant overhead**:

$$S_P = \frac{T_1}{(1-\varphi) \times T_1 + (\varphi \times T_1/P) + overhead}$$

Amdahl's Law with **linear overhead**:

$$S_P = \frac{T_1}{(1-\varphi)\times T_1 + (\varphi \times T_1/P) + overhead(P)}$$



Data sharing modeling

Each processor P_i has its **own memory**, interconnected with the other processors'.

Processors access local data with **zero overhead**.

Processors can access **remote data** using message-passsing model.

1. Start-up: time spent preparing the remote access (t_s)
2. Transfer: time spent transfering the message (m bytes, t_w time per byte)

$$T_{access} = t_s + m \times t_w$$

At any given moment, a processor P_i can at most execute a remote memory access to P_j and serve a memory access to P_k.