If we are lucky enough that $S_p$ increases with $p$ we can make our program get faster by adding more processors.
If we are lucky enough that $S_p$ increases with $p$ we can make our program get faster by adding more processors.

But at what cost?
If we are lucky enough that $S_p$ increases with $p$ we can make our program get faster by adding more processors.

But at what cost?

If we can double the speed of a program using 4 processors we feel we are doing better than if we used a different approach that needed 8 processors for the same speedup.
If we are lucky enough that $S_p$ increases with $p$ we can make our program get faster by adding more processors.

But at what cost?

If we can double the speed of a program using 4 processors we feel we are doing better than if we used a different approach that needed 8 processors for the same speedup.

*Efficiency* measures this.
Efficiency is speedup per processor:

\[ E_p = \frac{S_p}{p} = \frac{\text{time on a sequential processor}}{p \times \text{time on } p \text{ parallel processors}} \]
Analysis

Efficiency

Efficiency is speedup per processor:

\[ E_p = \frac{S_p}{p} = \frac{\text{time on a sequential processor}}{p \times \text{time on } p \text{ parallel processors}} \]

Usually 0 \leq E_p \leq 1, and is often written as a percentage.
Efficiency is speedup per processor:

\[ E_p = \frac{S_p}{p} = \frac{\text{time on a sequential processor}}{p \times \text{time on } p \text{ parallel processors}} \]

Usually \( 0 \leq E_p \leq 1 \), and is often written as a percentage.

\( E_p = 0.5 \) (50\%) means we are using only half of the processors’ capabilities on our computation; half is lost in overheads or idling while waiting for something.
Efficiency is speedup per processor:

\[ E_p = \frac{S_p}{p} = \frac{\text{time on a sequential processor}}{p \times \text{time on } p \text{ parallel processors}} \]

Usually \( 0 \leq E_p \leq 1 \), and is often written as a percentage

\( E_p = 0.5 \) (50%) means we are using only half of the processors’ capabilities on our computation; half is lost in overheads or idling while waiting for something

\( E_p = 1 \) (100%) means we are using all the processors all the time on our computation
Efficiency is speedup per processor:

$$E_p = \frac{S_p}{p} = \frac{\text{time on a sequential processor}}{p \times \text{time on } p \text{ parallel processors}}$$

Usually $0 \leq E_p \leq 1$, and is often written as a percentage

$E_p = 0.5$ (50%) means we are using only half of the processors’ capabilities on our computation; half is lost in overheads or idling while waiting for something

$E_p = 1$ (100%) means we are using all the processors all the time on our computation

$E_p > 1$ indicate superlinear speedup: we are using more than 100% of the processors!
Efficiency is useful when we need to gauge the cost of a parallel system: the higher the efficiency the better the utilisation of the processors.
Efficiency is useful when we need to gauge the cost of a parallel system: the higher the efficiency the better the utilisation of the processors.

When $E_p < 1$ this indicates that somewhere at some point a processor not working on the computation. Perhaps it is occupied in communication; or possibly just lying idle waiting.
Typical efficiency graph on a fixed size problem:
As an example of calculating speedup and efficiency we consider a pipeline (systolic array):
Analysis

Speedup and Efficiency

As an example of calculating speedup and efficiency we consider a pipeline (systolic array)

Data moves from one processor to the next being transformed at each stage: we assume one time step per transform
As an example of calculating speedup and efficiency we consider a pipeline (systolic array)

Data moves from one processor to the next being transformed at each stage: we assume one time step per transform

This could equally be a CPU instruction pipeline
A $p$-stage pipeline will take $p$ time steps to fill; after that it produces one result per time step.
A $p$-stage pipeline will take $p$ time steps to fill; after that it produces one result per time step.

So it can produce $n$ results in $p + (n - 1)$ time steps.
A $p$-stage pipeline will take $p$ time steps to fill; after that it produces one result per time step.

So it can produce $n$ results in $p + (n - 1)$ time steps.

A sequential system will take $np$ time steps to do the $p$ steps on the $n$ computations.
The speedup is

\[ S_p = \frac{np}{p + n - 1} = \frac{p}{(p - 1)/n + 1} \]
The speedup is

\[ S_p = \frac{np}{p + n - 1} = \frac{p}{(p - 1)/n + 1} \]

As time passes, the number of tasks \( n \) gets large, and \( S_p \to p \)
The speedup is

\[ S_p = \frac{np}{p + n - 1} = \frac{p}{(p - 1)/n + 1} \]

As time passes, the number of tasks \( n \) gets large, and \( S_p \to p \)

A \( p \)-stage pipeline has a speedup is less than \( p \), but that gets closer to \( p \) as time progresses
Analysis
Speedup and Efficiency

The speedup is

\[ S_p = \frac{np}{p + n - 1} = \frac{p}{(p - 1)/n + 1} \]

As time passes, the number of tasks \( n \) gets large, and \( S_p \rightarrow p \)

A \( p \)-stage pipeline has a speedup is less than \( p \), but that gets closer to \( p \) as time progresses

Also, the speedup starts low (for \( n = 1 \), \( S_p = p/(p + 1 - 1) = 1 \)) and increases over time, getting closer and closer to \( p \)
The efficiency is

\[ E_p = \frac{S_p}{p} = \frac{n}{p + n - 1} = \frac{1}{(p - 1)/n + 1} \]
The efficiency is

\[ E_p = \frac{S_p}{p} = \frac{n}{p + n - 1} = \frac{1}{(p - 1)/n + 1} \]

As \( n \) gets large, \( E_p \rightarrow 1 \).
Analysis

Speedup and Efficiency

The efficiency is

$$E_p = \frac{S_p}{p} = \frac{n}{p + n - 1} = \frac{1}{(p - 1)/n + 1}$$

As $n$ gets large, $E_p \rightarrow 1$

Eventually we are (close to) using all the processors all the time: perfect efficiency!
The efficiency is

$$E_p = \frac{S_p}{p} = \frac{n}{p + n - 1} = \frac{1}{(p - 1)/n + 1}$$

As $n$ gets large, $E_p \to 1$

Eventually we are (close to) using all the processors all the time: perfect efficiency!

Also, the efficiency starts low (for $n = 1$, $E_p = 1/(p + 1 - 1) = 1/p$) and increases over time
Pipelines are a really good way of making something parallel: both great speedup and great efficiency.
Pipelines are a really good way of making something parallel: both great speedup and great efficiency

As long as we can keep the pipeline full: in a CPU each time we take a jump the instruction pipeline breaks, empties and needs to refill
Pipelines are a really good way of making something parallel: both great speedup and great efficiency

As long as we can keep the pipeline full: in a CPU each time we take a jump the instruction pipeline breaks, empties and needs to refill

To keep high efficiency we need to avoid this: thus the complications in the designs of modern processors that are aimed at keeping the pipeline full
Pipelines are a really good way of making something parallel: both great speedup and great efficiency.

As long as we can keep the pipeline full: in a CPU each time we take a jump the instruction pipeline breaks, empties and needs to refill.

To keep high efficiency we need to avoid this: thus the complications in the designs of modern processors that are aimed at keeping the pipeline full.

(Things like speculative evaluation, using many transistors. . .)