Speedup and Efficiency are simple, but useful measures of a parallel system, as long as you take care over using them.
Analysis

Other measures

Speedup and Efficiency are simple, but useful measures of a parallel system, as long as you take care over using them.

There are many other measures that are occasionally used, but they are of lesser importance.
Sometimes people use the *Karp-Flatt metric* as a measure of an implementation to see how well it is doing
Sometimes people use the *Karp-Flatt metric* as a measure of an implementation to see how well it is doing.

This is essentially an empirical measure of the sequential fraction of a computation (important for the Amdahl limit).
Sometimes people use the *Karp-Flatt metric* as a measure of an implementation to see how well it is doing.

This is essentially an empirical measure of the sequential fraction of a computation (important for the Amdahl limit)

\[
e = \frac{1}{S_p} - \frac{1}{p} \frac{1}{1 - \frac{1}{p}}
\]

where \( S_p \) is the measured speedup and \( p \) the number of processors.
Analysis
Karp-Flatt

A larger $e$ indicates a larger sequential part
A larger $e$ indicates a larger sequential part

If we have perfect speedup, $S_p = p$, then $e = 0$
A larger $e$ indicates a larger sequential part

If we have perfect speedup, $S_p = \rho$, then $e = 0$

If we have no speedup, $S_p = 1$, then $e = 1$

Exercise. Calculate Karp-Flatt for the pipeline. What does it tell us?

Exercise. Some people use the phrase "negative speedup" rather than "slowdown". Why is that a bad idea?
Analysis
Karp-Flatt

A larger $e$ indicates a larger sequential part

If we have perfect speedup, $S_p = p$, then $e = 0$

If we have no speedup, $S_p = 1$, then $e = 1$

If we have slowdown, e.g., $S_p = 1/2$, then $e \approx 2$
A larger $e$ indicates a larger sequential part

If we have perfect speedup, $S_p = p$, then $e = 0$

If we have no speedup, $S_p = 1$, then $e = 1$

If we have slowdown, e.g., $S_p = 1/2$, then $e \approx 2$

(If we have superlinear speedup, $S_p > p$, then $e < 0$)
A larger $e$ indicates a larger sequential part

If we have perfect speedup, $S_p = p$, then $e = 0$

If we have no speedup, $S_p = 1$, then $e = 1$

If we have slowdown, e.g., $S_p = 1/2$, then $e \approx 2$

(If we have superlinear speedup, $S_p > p$, then $e < 0$)

Exercise. Calculate Karp-Flatt for the pipeline. What does it tell us?
Analysis
Karp-Flatt

A larger $e$ indicates a larger sequential part

If we have perfect speedup, $S_p = p$, then $e = 0$

If we have no speedup, $S_p = 1$, then $e = 1$

If we have slowdown, e.g., $S_p = 1/2$, then $e \approx 2$

(If we have superlinear speedup, $S_p > p$, then $e < 0$)

Exercise. Calculate Karp-Flatt for the pipeline. What does it tell us?

Exercise. Some people use the phrase “negative speedup” rather than “slowdown”. Why is that a bad idea?
A parallel algorithm is *work efficient* (*cost efficient*) if the number of operations it performs is no more than the sequential algorithm.
A parallel algorithm is *work efficient* (*cost efficient*) if the number of operations it performs is no more than the sequential algorithm.

For example, a parallel algorithm might duplicate some operations on separate processors as it is more convenient, or reduces communications.
A parallel algorithm is *work efficient* (*cost efficient*) if the number of operations it performs is no more than the sequential algorithm.

For example, a parallel algorithm might duplicate some operations on separate processors as it is more convenient, or reduces communications.

The *parallel overhead* is

\[ T_o = pT_p - T_s \]

where \( T_s \) is the sequential time and \( T_p \) is the parallel time.
Analysis

Work Efficient

This measures the amount of extra work we are doing to get the parallelism
Analysis

Work Efficient

This measures the amount of extra work we are doing to get the parallelism

A measure of the extra energy expended in the parallel algorithm or implementation
Analysis
Work Efficient

This measures the amount of extra work we are doing to get the parallelism

A measure of the extra energy expended in the parallel algorithm or implementation

And the cost of the overheads (e.g., communication) when we measure a real implementation
Analysis
Work Efficient

This measures the amount of extra work we are doing to get the parallelism

A measure of the extra energy expended in the parallel algorithm or implementation

And the cost of the overheads (e.g., communication) when we measure a real implementation

Exercise. Calculate the parallel overhead for the pipeline. What does it tell us?
Analysis

Isoefficiency

Another question is “how scalable is this algorithm?”
Another question is “how scalable is this algorithm?”

Here we ask for a relationship between $p$, the number of processors and $n$ the size of the problem for a given efficiency.
Another question is “how scalable is this algorithm?”

Here we ask for a relationship between $p$, the number of processors and $n$ the size of the problem for a given efficiency.

If we increase $p$, how much to we have to increase $n$ to maintain a given efficiency?
Increasing $p$ will generally decrease efficiency (Amdahl).
Increasing $p$ will generally decrease efficiency (Amdahl)

Increasing $n$ will generally increase efficiency (Gustafson)
Analysis
Isoefficiency

Increasing $p$ will generally decrease efficiency (Amdahl)

Increasing $n$ will generally increase efficiency (Gustafson)

A poorly scalable algorithm will need to increase $n$ a lot to maintain efficiency as we increase $p$
Increasing $p$ will generally decrease efficiency (Amdahl)

Increasing $n$ will generally increase efficiency (Gustafson)

A poorly scalable algorithm will need to increase $n$ a lot to maintain efficiency as we increase $p$

This relationship is called the *isoefficiency*, and expresses $n$ as a function of $p$
Analysis

Isoefficiency

Increasing \( p \) will generally decrease efficiency (Amdahl)

Increasing \( n \) will generally increase efficiency (Gustafson)

A poorly scalable algorithm will need to increase \( n \) a lot to maintain efficiency as we increase \( p \)

This relationship is called the *isoefficiency*, and expresses \( n \) as a function of \( p \)

It quantifies the balance between Amdahl and Gustafson
Computing the isoefficiency can be a bit fiddly, but often it is easiest to start by looking at the parallel overhead.
Computing the isoefficiency can be a bit fiddly, but often it is easiest to start by looking at the parallel overhead.

We have efficiency \( E = T_s / p T_p \) and overhead \( T_o = p T_p - T_s \). Combining these:

\[
E = \frac{T_s}{p \left( \frac{T_o + T_s}{p} \right)} = \frac{T_s}{T_o + T_s} = \frac{1}{1 + T_o / T_s}
\]
Computing the isoefficiency can be a bit fiddly, but often it is easiest to start by looking at the parallel overhead.

We have efficiency \( E = \frac{T_s}{pT_p} \) and overhead \( T_o = pT_p - T_s \). Combining these:

\[
E = \frac{T_s}{p \left( \frac{T_o + T_s}{p} \right)} = \frac{T_s}{T_o + T_s} = \frac{1}{1 + T_o/T_s}
\]

So to keep \( E \) constant, we need to keep \( T_o/T_s \) constant.
So we must have

\[ T_s = cT_o \]

for some constant \( c \)
So we must have

\[ T_s = c T_o \]

for some constant \( c \)

As both \( T_s \) and \( T_o \) depend on \( n \) and \( p \), this equation generally gives us enough to solve for \( n \) in terms of \( p \)
Example. The $p$-stage pipeline had efficiency $E = n/(p + n - 1)$ on a problem of size $n$. 
Example. The $p$-stage pipeline had efficiency

$$E = \frac{n}{p + n - 1}$$

on a problem of size $n$

The overhead

$$T_o = pT_p - T_s = p(p + n - 1) - np = p^2 - p$$

independent of $n$
Example. The $p$-stage pipeline had efficiency

$$E = \frac{n}{(p + n - 1)}$$
on a problem of size $n$

The overhead

$$T_o = pT_p - T_s = p(p + n - 1) - np = p^2 - p$$

independent of $n$

This fixed overhead again tells us it is a good idea to keep the pipeline full!
We want $T_s = cT_o$ which is

$$np = c(p^2 - p)$$
We want $T_s = cT_o$ which is

$$np = c(p^2 - p)$$

We solve for $n$

$$n = c(p - 1)$$
We want $T_s = cT_o$ which is

$$np = c(p^2 - p)$$

We solve for $n$

$$n = c(p - 1)$$

Thus the isoefficiency is

$$n = O(p)$$
This is linear in $p$: if we double $p$ we need only double $n$ to maintain efficiency
This is linear in $p$: if we double $p$ we need only double $n$ to maintain efficiency

So this tells us pipelines are very scalable
There are many ways we can measure if our parallel program is performing well, or poorly.
Analysis
Other measures

There are many ways we can measure if our parallel program is performing well, or poorly

But we do need to be careful that we are making meaningful comparisons of parallel and sequential algorithms
There are many ways we can measure if our parallel program is performing well, or poorly.

But we do need to be careful that we are making meaningful comparisons of parallel and sequential algorithms.

Exercise. Compute these measures for adding $n$ numbers on $p$ processors.