

COL380

Introduction to  
Parallel & Distributed Programming

# Agenda

- Performance Metrics

- A function of: input size, number of processors
  - ➔ communication, dependencies
- Scaling
- How fast does a job complete?
  - ➔ Elapsed wall time (Latency)
    - ▶ compute + communicate + synchronize
- How many jobs complete in a given time?
  - ➔ Throughput

$$\text{Speedup, } S_p = \frac{\text{Exec time using 1 processor system } (t_1)}{\text{Exec time using p processors } (t_p)}$$

$$\text{Speedup, } S_p = \frac{\text{Exec time using 1 processor system } (t_1)}{\text{Exec time using p processors } (t_p)}$$

$$\text{Efficiency, } \mathcal{E}_p = \frac{S_p}{p}$$



## Performance Metrics

$$\text{Speedup, } S_p = \frac{\text{Exec time using 1 processor system } (t_1)}{\text{Exec time using } p \text{ processors } (t_p)}$$

$$\text{Efficiency, } \mathcal{E}_p = \frac{S_p}{p}$$

$$\text{Cost, } C_p = p \times t_p$$

Cost Optimal if  $C_p = t_1$

## Performance Metrics

$$\text{Speedup, } S_p = \frac{\text{Exec time using 1 processor system } (t_1)}{\text{Exec time using } p \text{ processors } (t_p)}$$

$$\text{Efficiency, } \mathcal{E}_p = \frac{S_p}{p}$$

$$\text{Cost, } C_p = p \times t_p$$

Cost Optimal if  $C_p = t_1$

Parallelization Overhead

$$\overline{o}_p = p \times t_p - t_1$$

$$\text{Speedup, } S_p = \frac{\text{Exec time using 1 processor system } (t_1)}{\text{Exec time using } p \text{ processors } (t_p)}$$

$$\text{Efficiency, } \mathcal{E}_p = \frac{S_p}{p}$$

$$\text{Cost, } C_p = p \times t_p$$

Cost Optimal if  $C_p = t_1$

Look out for inefficiency:

$$t_1 = n^3$$

$$t_p = n^2, \text{ for } p = n^2$$

$$C_p = n^4$$

Parallelization Overhead

$$\overline{o}_p = p \times t_p - t_1$$



Parameterize with “p”

- Algorithms scale up to a processor count,  $p$ 
  - A larger value of  $p$  raises the expectation that the algorithm scales well

- Algorithms scale up to a processor count,  $p$ 
  - ➔ A larger value of  $p$  raises the expectation that the algorithm scales well
- To execute using fewer processors  $p^\# < p$ :
  - ➔ Simulate each step (that uses up to  $p$  processors) thus:
    - ▶ Group up to  $p$  operations into  $p^\#$  sets, one for each available processor
    - ▶ Each set takes  $\lfloor p/p^\# \rfloor$  or  $\lceil p/p^\# \rceil$  steps on one processor
    - ▶ The time taken for original step =  $O(\lceil p/p^\# \rceil)$
- The total time taken  $t(n, p^\#) = O(t(n, p) * \lceil p/p^\# \rceil)$

Parameterize with “p”

- Algorithms scale up to a processor count,  $p$ 
  - ➔ A larger value of  $p$  raises the expectation that the algorithm scales well

- To execute using fewer processors  $p^\# < p$ :

➔ Simulate each step (that uses up to  $p$  processors) thus:

- ▶ Group up to  $p$  operations into  $p^\#$  sets, one for each available processor
- ▶ Each set takes  $\lfloor p/p^\# \rfloor$  or  $\lceil p/p^\# \rceil$  steps on one processor
- ▶ The time taken for original step =  $O(\lceil p/p^\# \rceil)$

- The total time taken  $t(n, p^\#) = O(t(n, p) * \lceil p/p^\# \rceil)$

$$\text{Work} = \sum_{i=0}^{t(n,p)} p_i$$

number of processors active at step  $i$



## Amdahl's Law

- $f$  = fraction of the problem that is sequential
  - ➔  $\Rightarrow (1 - f)$  = fraction that is parallel

- Best parallel time
$$t_{par} = t_{seq} \left( f + \frac{1-f}{p} \right)$$
  - ➔ Fraction  $(1-f)$  equally shared by  $p$  processors

- Speedup with  $p$  processors:
$$S_p = \frac{1}{f + \frac{1-f}{p}}$$

## Amdahl's Law

$$S_p = \frac{1}{f + \frac{1-f}{p}} \rightarrow 0$$

- Speed-up due to  $p$  processors
- Upper bound on speedup at  $p = \infty$

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



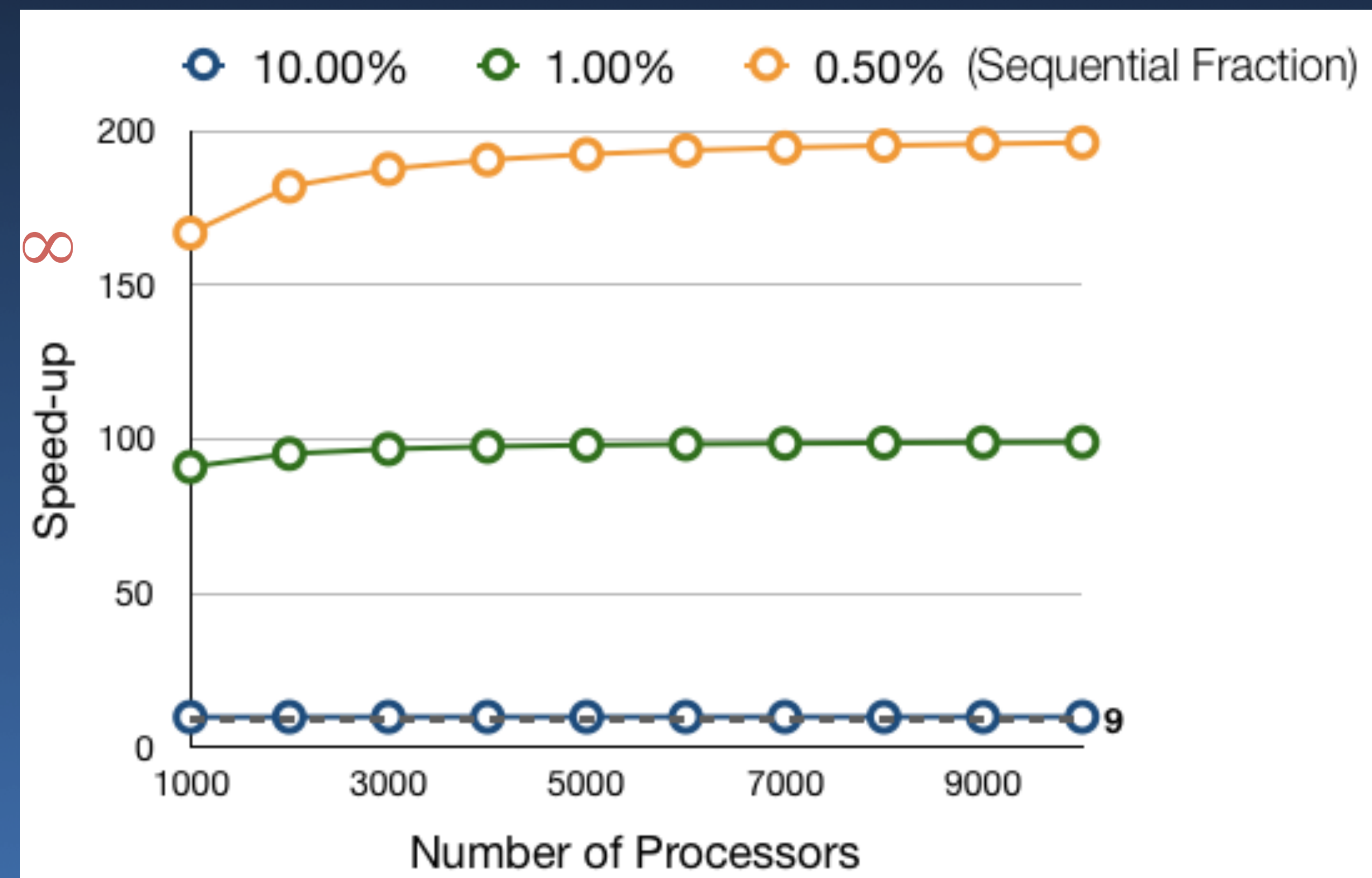
## Amdahl's Law

$$S_p = \frac{1}{f + \frac{1-f}{p}} \rightarrow 0$$

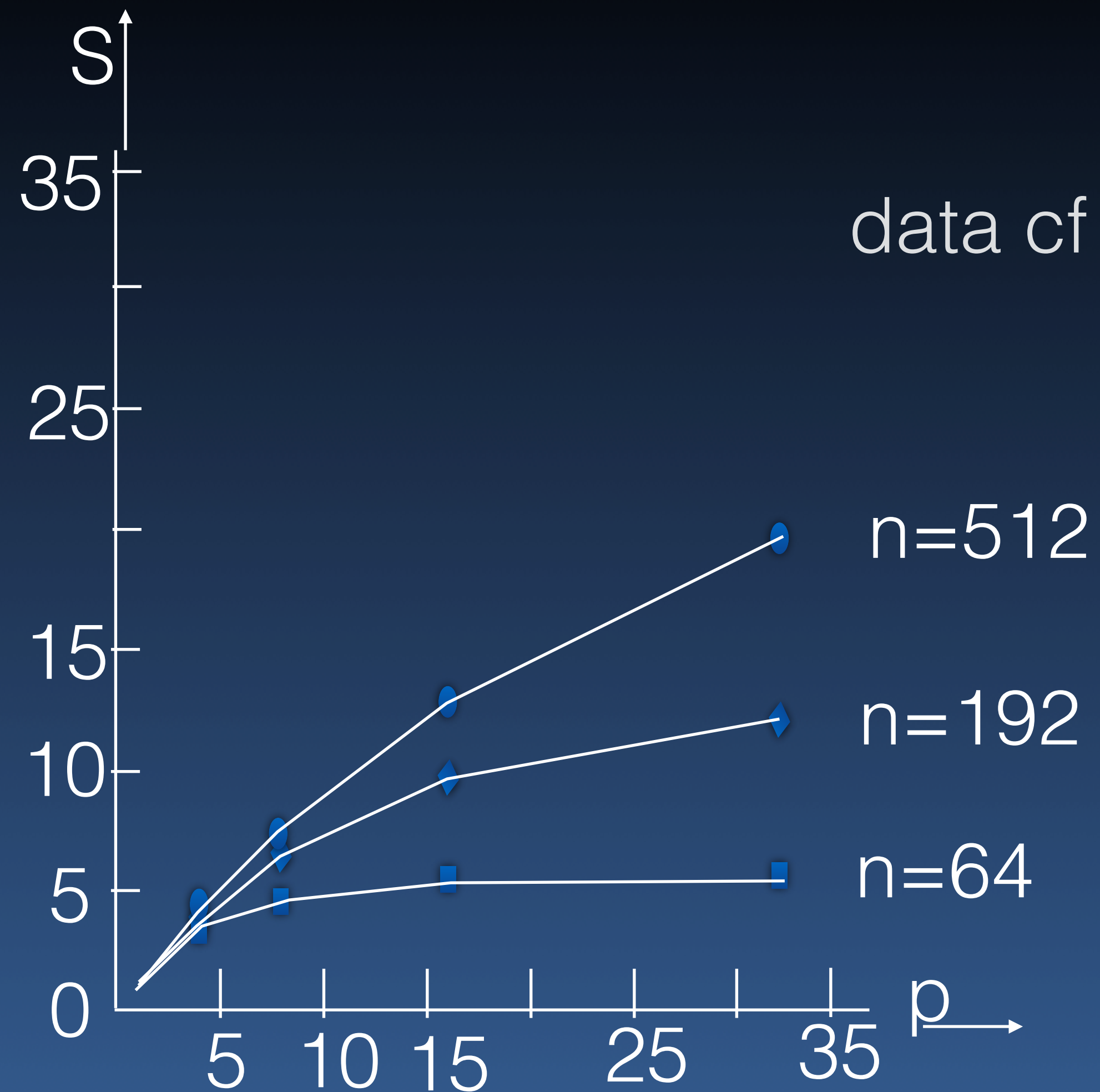
- Speed-up due to  $p$  processors
- Upper bound on speedup at  $p = \infty$

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

$$f = 10\%, S_{\infty} \rightarrow 1 / 0.1 = 10$$



## Example Scaling



Speedup saturates and efficiency drops  
(Amdahl's law)

But the limit depends on n:  
size of the problem

Speedup versus the number of processing  
elements for adding a list of numbers

- Observation: applications seem to exceed Amdahl's speed-up
  - (i.e., assumption are too restrictive)
- As  $p$  increases, the opportunity for parallelization can also increase

Time taken for a problem on  $p$  processors =  $t_{\text{seq}} + t_{\text{par}} = t(n, p)$

## Gustaffson's Law

- Observation: applications seem to exceed Amdahl's speed-up
  - (i.e., assumption are too restrictive)
- As  $p$  increases, the opportunity for parallelization can also increase

$$\text{Time taken for a problem on } p \text{ processors} = \underbrace{t_{\text{seq}}}_{f \cdot t(n,p)} + \underbrace{t_{\text{par}}}_{(1-f) \cdot t(n,p)} = t(n,p)$$

$f$  = fraction of  
time spent in  
sequential work



## Gustaffson's Law

- Observation: applications seem to exceed Amdahl's speed-up
  - (i.e., assumption are too restrictive)
- As  $p$  increases, the opportunity for parallelization can also increase

Time taken for a problem on  $p$  processors =  $\underbrace{t_{\text{seq}}}_{f t(n,p)} + \underbrace{t_{\text{par}}}_{(1-f) t(n,p)} = t(n,p)$

$$t(n,1) = f t(n,p) + p (1-f) t(n,p)$$

$f$  = fraction of  
time spent in  
sequential work



## Gustaffson's Law

- Observation: applications seem to exceed Amdahl's speed-up
  - (i.e., assumption are too restrictive)
- As  $p$  increases, the opportunity for parallelization can also increase

Time taken for a problem on  $p$  processors =  $\underbrace{t_{\text{seq}}}_{f t(n,p)} + \underbrace{t_{\text{par}}}_{(1-f) t(n,p)} = t(n,p)$

$$t(n,1) = f t(n,p) + p (1-f) t(n,p)$$

$$\Rightarrow S_p = \frac{t(n,1)}{t(n,p)} = f + p (1-f)$$

$f$  = fraction of  
time spent in  
sequential work

## Gustaffson's Law

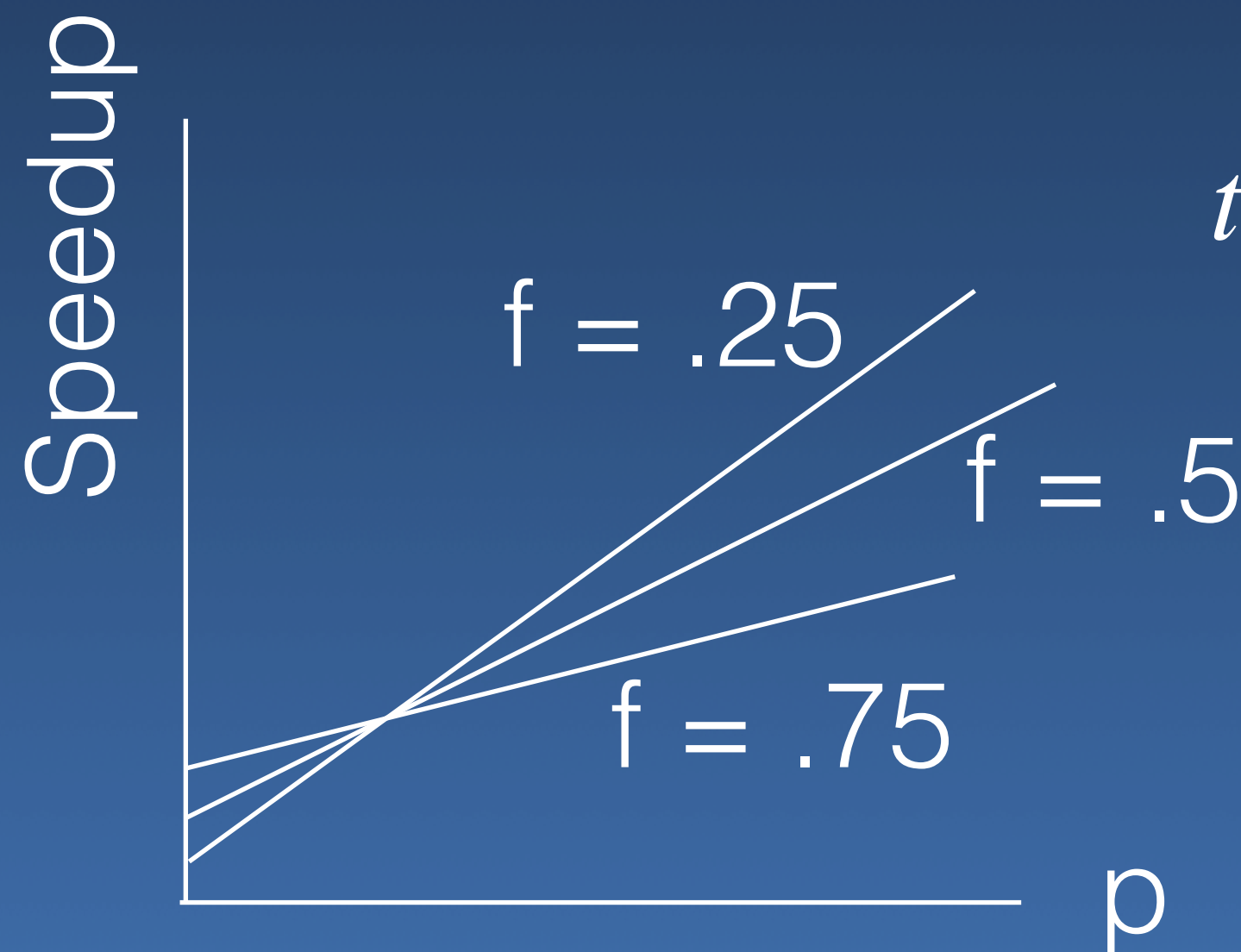
- Observation: applications seem to exceed Amdahl's speed-up
  - (i.e., assumption are too restrictive)
- As  $p$  increases, the opportunity for parallelization can also increase

Time taken for a problem on  $p$  processors =  $\underbrace{t_{\text{seq}}}_{f t(n,p)} + \underbrace{t_{\text{par}}}_{(1-f) t(n,p)} = t(n,p)$

$f$  = fraction of time spent in sequential work

$$t(n,1) = f t(n,p) + p (1-f) t(n,p)$$

$$\Rightarrow S_p = \frac{t(n,1)}{t(n,p)} = f + p (1-f)$$



## Gustaffson's Law

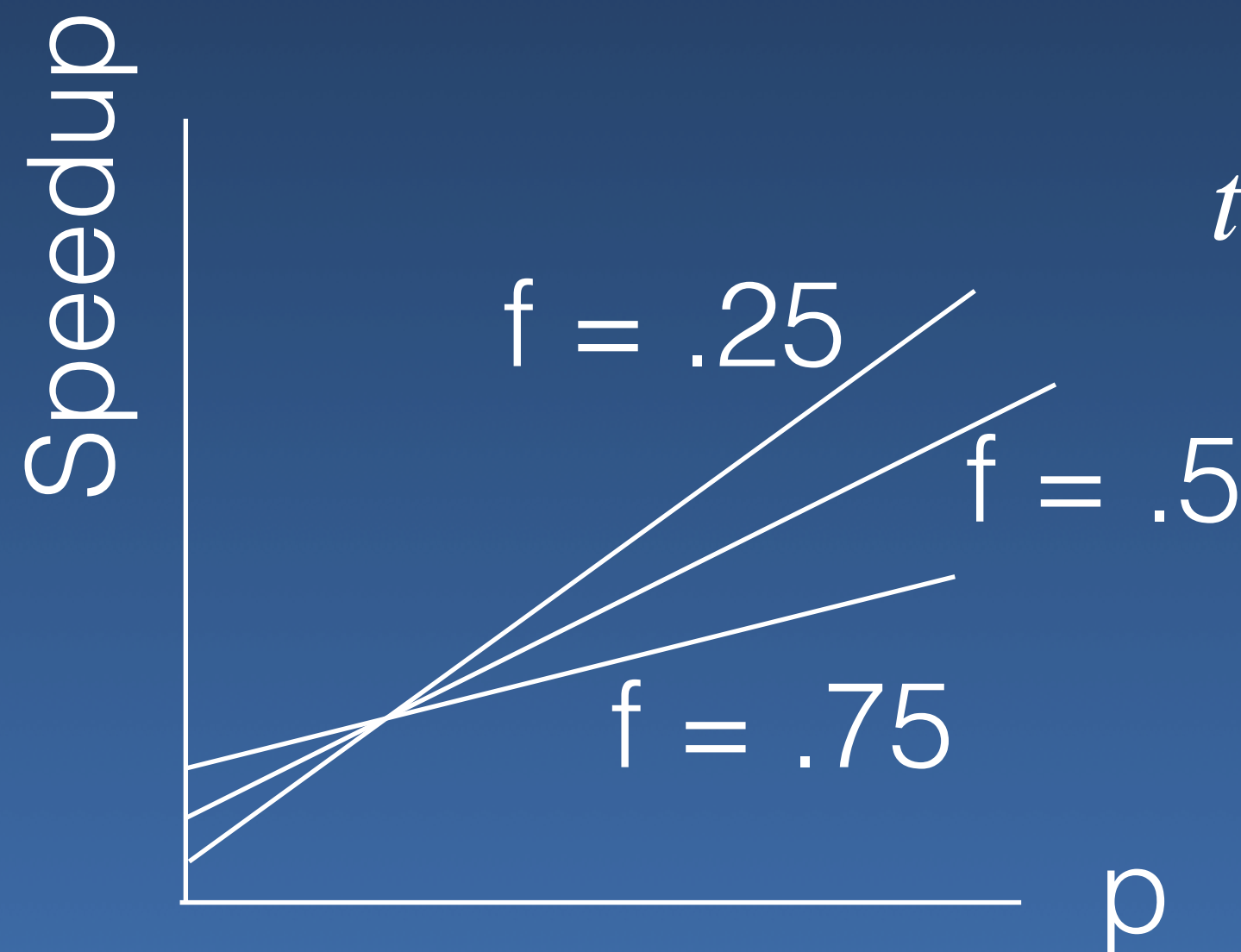
- Observation: applications seem to exceed Amdahl's speed-up
  - (i.e., assumption are too restrictive)
- As  $p$  increases, the opportunity for parallelization can also increase

Time taken for a problem on  $p$  processors =  $\underbrace{t_{\text{seq}}}_{f t(n,p)} + \underbrace{t_{\text{par}}}_{(1-f) t(n,p)} = t(n,p)$

$f$  = fraction of time spent in sequential work

$$t(n,1) = f t(n,p) + p (1-f) t(n,p)$$

$$\Rightarrow S_p = \frac{t(n,1)}{t(n,p)} = f + p (1-f) \neq \frac{1}{f + \frac{1-f}{p}} \quad (\text{Amdahl's law})$$



## Karp Flatt metric

$$S_p = \frac{1}{f + \frac{1-f}{p}}$$

$f$   
(Amdahl's law)

- Estimate the fractional part



## Karp Flatt metric

$f$   
(Amdahl's law)

$$S_p = \frac{1}{f + \frac{1-f}{p}}$$

$$\Rightarrow S_p = \frac{p}{pf + 1 - f}$$

- Estimate the fractional part

$$\Rightarrow \frac{p}{S_p} - 1 = (p - 1)f$$

$$\Rightarrow f = \frac{\frac{1}{S_p} - \frac{1}{p}}{1 - \frac{1}{p}}$$



## Karp Flatt metric

$f$   
(Amdahl's law)

$$S_p = \frac{1}{f + \frac{1-f}{p}}$$

$$\Rightarrow S_p = \frac{p}{pf + 1 - f}$$

- Estimate the fractional part

$$\Rightarrow \frac{p}{S_p} - 1 = (p - 1)f$$

$$\Rightarrow f = \frac{\frac{1}{S_p} - \frac{1}{p}}{1 - \frac{1}{p}}$$

Measure speedup,  
estimate the sequential fraction

## *Isoefficiency:* Measure of Scalability

- Rate at which *problem size* must grow (as a function of  $p$ ) to maintain constant *efficiency*

## *Isoefficiency:* Measure of Scalability

- Rate at which *problem size* must grow (as a function of  $p$ ) to maintain constant *efficiency*
  - If  $n$  need not grow with  $p \Rightarrow$  Strongly scalable
    - ▶ Weakly scalable, otherwise
    - ▶ Lower rate of growth  $\Rightarrow$  More Scalable


## *Isoefficiency:* Measure of Scalability

- Rate at which *problem size* must grow (as a function of  $p$ ) to maintain constant *efficiency*

→ If  $n$  need not grow with  $p \Rightarrow$  Strongly scalable

▶ Weakly scalable, otherwise

▶ Lower rate of growth  $\Rightarrow$  More Scalable


$$\mathcal{E}(n, p) = \frac{t(n, 1)}{p \ t(n, p)}$$




## *Isoefficiency:* Measure of Scalability

- Rate at which *problem size* must grow (as a function of  $p$ ) to maintain constant *efficiency*

→ If  $n$  need not grow with  $p \Rightarrow$  Strongly scalable

▶ Weakly scalable, otherwise

▶ Lower rate of growth  $\Rightarrow$  More Scalable


$$\mathcal{E}(n, p) = \frac{t(n, 1)}{p \ t(n, p)} = k$$




## *Isoefficiency:* Measure of Scalability

- Rate at which *problem size* must grow (as a function of  $p$ ) to maintain constant *efficiency*

→ If  $n$  need not grow with  $p \Rightarrow$  Strongly scalable

▶ Weakly scalable, otherwise

▶ Lower rate of growth  $\Rightarrow$  More Scalable


$$\mathcal{E}(n, p) = \frac{t(n, 1)}{p \cdot t(n, p)} = k$$


## *Isoefficiency:* Measure of Scalability

- Rate at which *problem size* must grow (as a function of  $p$ ) to maintain constant *efficiency*

→ If  $n$  need not grow with  $p \Rightarrow$  Strongly scalable

▶ Weakly scalable, otherwise

▶ Lower rate of growth  $\Rightarrow$  More Scalable


$$\mathcal{E}(n, p) = \frac{t(n, 1)}{p \cdot t(n, p)} = k$$


## *Isoefficiency:* Measure of Scalability

- Rate at which *problem size* must grow (as a function of  $p$ ) to maintain constant *efficiency*

→ If  $n$  need not grow with  $p \Rightarrow$  Strongly scalable

▶ Weakly scalable, otherwise

▶ Lower rate of growth  $\Rightarrow$  More Scalable


$$\mathcal{E}(n, p) = \frac{t(n, 1)}{p \cdot t(n, p)} = k$$


## *Isoefficiency:* Measure of Scalability

- Rate at which *problem size* must grow (as a function of  $p$ ) to maintain constant *efficiency*

→ If  $n$  need not grow with  $p \Rightarrow$  Strongly scalable

▶ Weakly scalable, otherwise

▶ Lower rate of growth  $\Rightarrow$  More Scalable


$$\mathcal{E}(n, p) = \frac{t(n, 1) \uparrow}{p \downarrow t(n, p)} = k$$



## Isoefficiency: Measure of Scalability

- Rate at which *problem size* must grow (as a function of  $p$ ) to maintain constant *efficiency*

→ If  $n$  need not grow with  $p \Rightarrow$  Strongly scalable

▶ Weakly scalable, otherwise

▶ Lower rate of growth  $\Rightarrow$  More Scalable

$$t(n,1) = \mathcal{F}(n) \text{ “problem size”}$$

$$\mathcal{E}(n,p) = \frac{t(n,1) \uparrow}{p \downarrow t(n,p)} = k$$

## Isoefficiency: Measure of Scalability

- Rate at which *problem size* must grow (as a function of  $p$ ) to maintain constant *efficiency*

→ If  $n$  need not grow with  $p \Rightarrow$  Strongly scalable

▶ Weakly scalable, otherwise

▶ Lower rate of growth  $\Rightarrow$  More Scalable

$$t(n,1) = \mathcal{F}(n) \text{ “problem size”}$$

$$\mathcal{E}(n,p) = \frac{t(n,1) \uparrow}{p \downarrow t(n,p)} = k$$

Measure of problem size:  $n(p) = \mathcal{F}(p) \text{ s.t. } \mathcal{E} = k$

## Isoefficiency: Measure of Scalability

- Rate at which *problem size* must grow (as a function of  $p$ ) to maintain constant *efficiency*

→ If  $n$  need not grow with  $p \Rightarrow$  Strongly scalable

▶ Weakly scalable, otherwise

▶ Lower rate of growth  $\Rightarrow$  More Scalable

$$t(n,1) = \mathcal{F}(n) \text{ "problem size"}$$

$$\mathcal{E}(n,p) = \frac{t(n,1)}{p \cdot t(n,p)} = k$$

Example:

$$t(n,p) = \Theta\left(\frac{n}{p} + \log p\right)$$

$$t(n,1) = \Theta(n)$$

$$pt(n,p) = \Theta(n + p \log p) = \Theta(n)$$

Measure of problem size:  $n(p) = \mathcal{F}(p) \text{ s.t. } \mathcal{E} = k$



## Isoefficiency: Measure of Scalability

- Rate at which *problem size* must grow (as a function of  $p$ ) to maintain constant *efficiency*

→ If  $n$  need not grow with  $p \Rightarrow$  Strongly scalable

▶ Weakly scalable, otherwise

▶ Lower rate of growth  $\Rightarrow$  More Scalable

$$t(n,1) = \mathcal{F}(n) \text{ "problem size"}$$

$$\mathcal{E}(n,p) = \frac{t(n,1)}{p t(n,p)} = k$$

Example:

$$t(n,p) = \Theta\left(\frac{n}{p} + \log p\right)$$

Measure of problem size:  $n(p) = \mathcal{F}(p) \text{ s.t. } \mathcal{E} = k$

$$t(n,1) = \Theta(n)$$

$$\mathcal{F}(p) = \Omega(p \log p)$$

$$pt(n,p) = \Theta(n + p \log p) = \Theta(n)$$



## Isoefficiency: Measure of Scalability

- Rate at which *problem size* must grow (as a function of  $p$ ) to maintain constant *efficiency*

→ If  $n$  need not grow with  $p \Rightarrow$  Strongly scalable

▶ Weakly scalable, otherwise

▶ Lower rate of growth  $\Rightarrow$  More Scalable

$$t(n,1) = \mathcal{F}(n) \text{ "problem size"}$$

$$\mathcal{E}(n,p) = \frac{t(n,1) \uparrow}{p \downarrow t(n,p)} = k$$

Example:

$$t(n,p) = \Theta\left(\frac{n}{p} + \log p\right)$$

Measure of problem size:  $n(p) = \mathcal{J}(p) \text{ s.t. } \mathcal{E} = k$

$$t(n,1) = \Theta(n)$$

$$\mathcal{J}(p) = \Omega(p \log p)$$

$$pt(n,p) = \Theta(n + p \log p) = \Theta(n)$$

Isoefficiency Function  $\mathcal{J}(p) = \Omega(\bar{o}(p))$

- Time, speed-up, Efficiency, Cost, Work
  - Parametrization with input size ( $n$ ) and processor count ( $p$ ).
- Scalability
  - Limits on scaling
    - ▶ Amdahl's law, Gustaffson's law, Karp-Flatt metric
  - Iso-efficiency