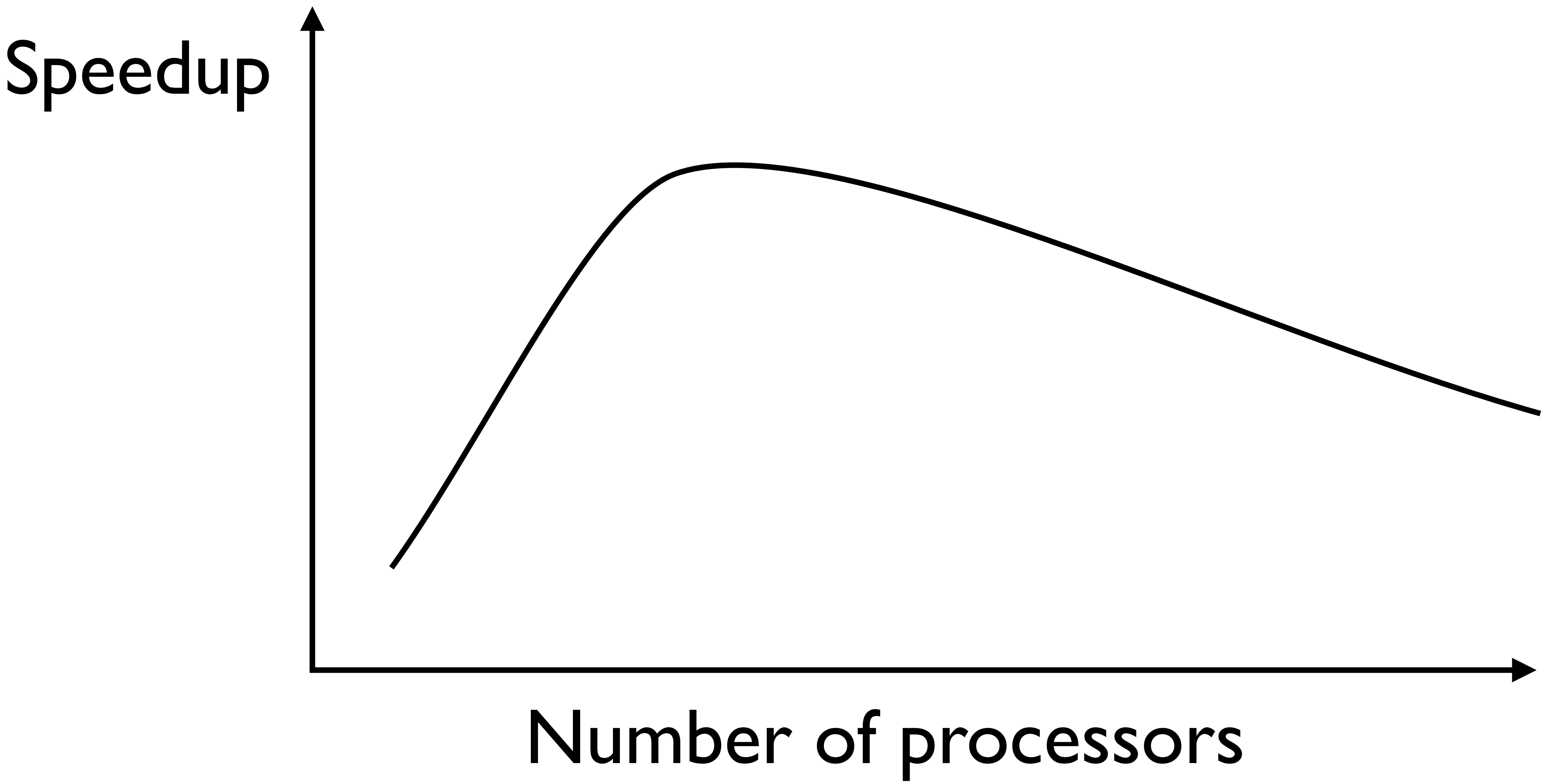
Performance analysis

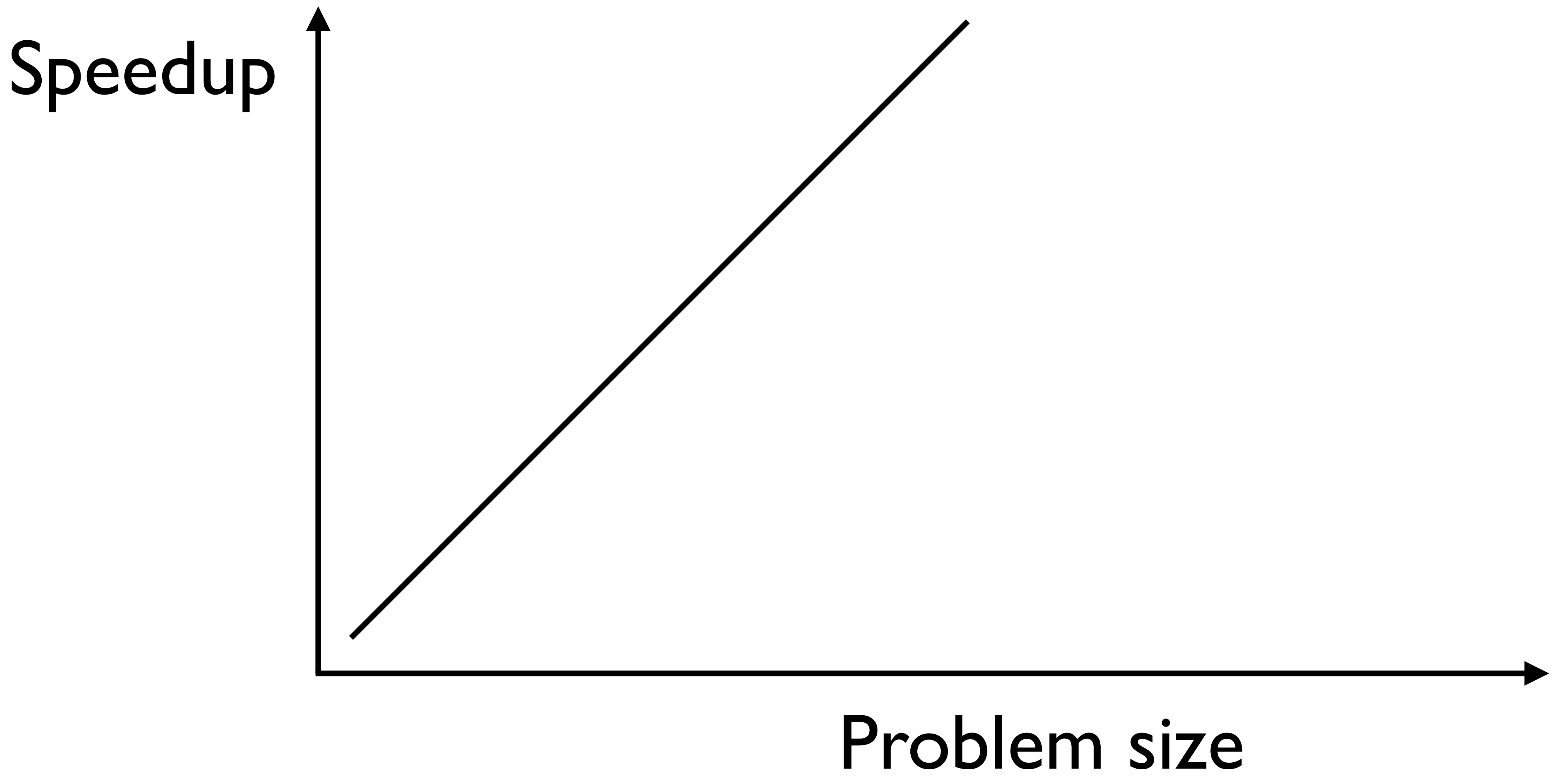
Goals are

1. to be able to understand better why your program has the performance it has, and
2. what could be preventing its performance from being better.

A typical *speedup* curve - fixed problem size



A typical *speedup* curve - problem size grows with number of processors



Speedup

$$speedup \leq \frac{T_S}{T_P(p)}$$

$$speedup \leq \frac{serial\ \ time}{parallel\ \ time}$$

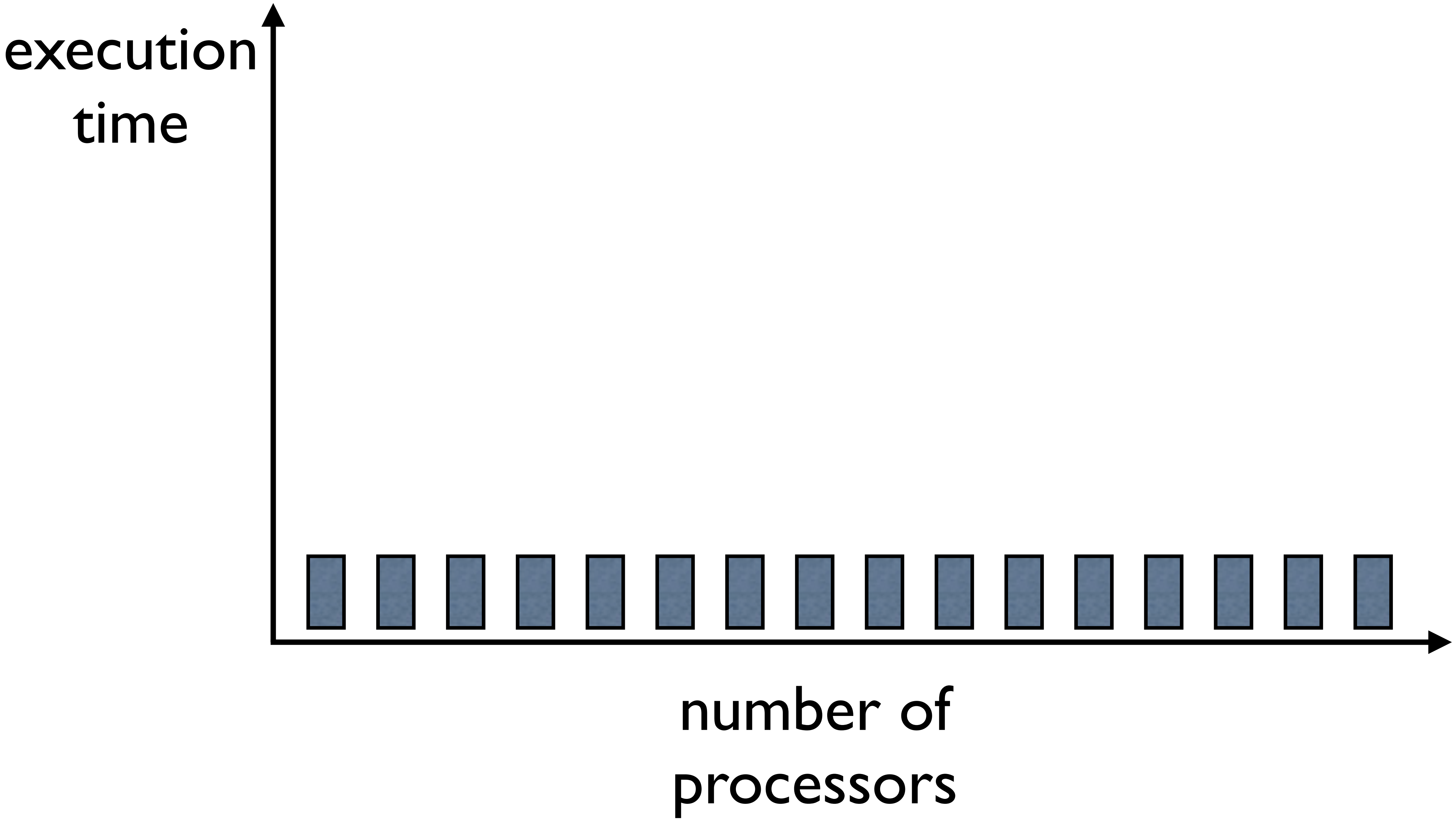
- Parallel time $T_P(p)$ is the time it takes the parallel form of the program to run on p processors
- Sequential time T_S is more problematic
  a. Can be $T_P(1)$, but this carries the overhead of extra code needed for parallelization. Even with one thread, OpenMP code will call libraries for threading. **One way to “cheat” on benchmarking.**
  b. Should be the best possible sequential implementation: tuned, good or best compiler switches, etc.

What is execution time?

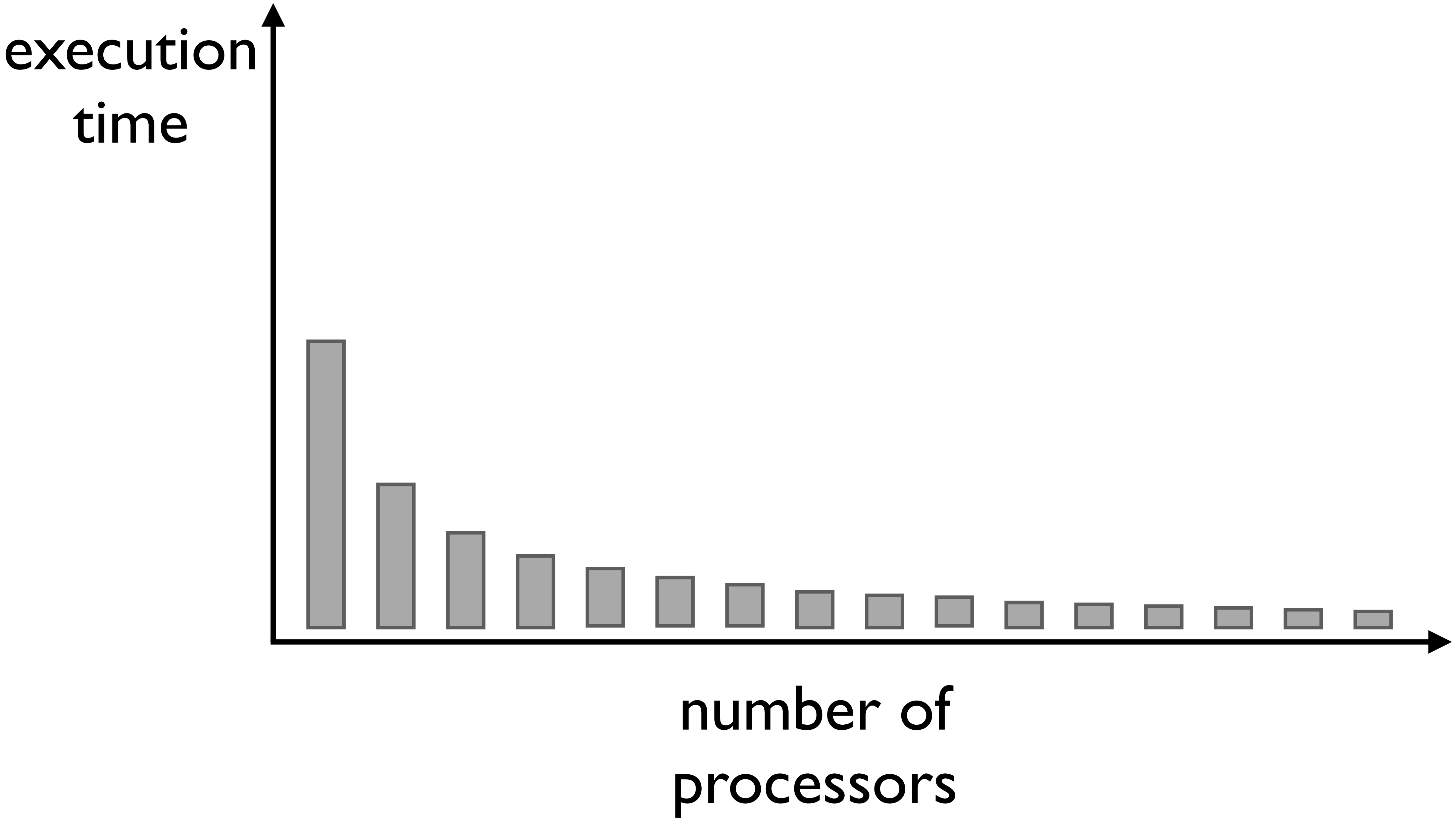
- Execution time can be modeled as the sum of:

1. Inherently sequential computation $\sigma(n)$
2. Potentially parallel computation $\phi(n)$
3. Communication time $\kappa(n,p)$

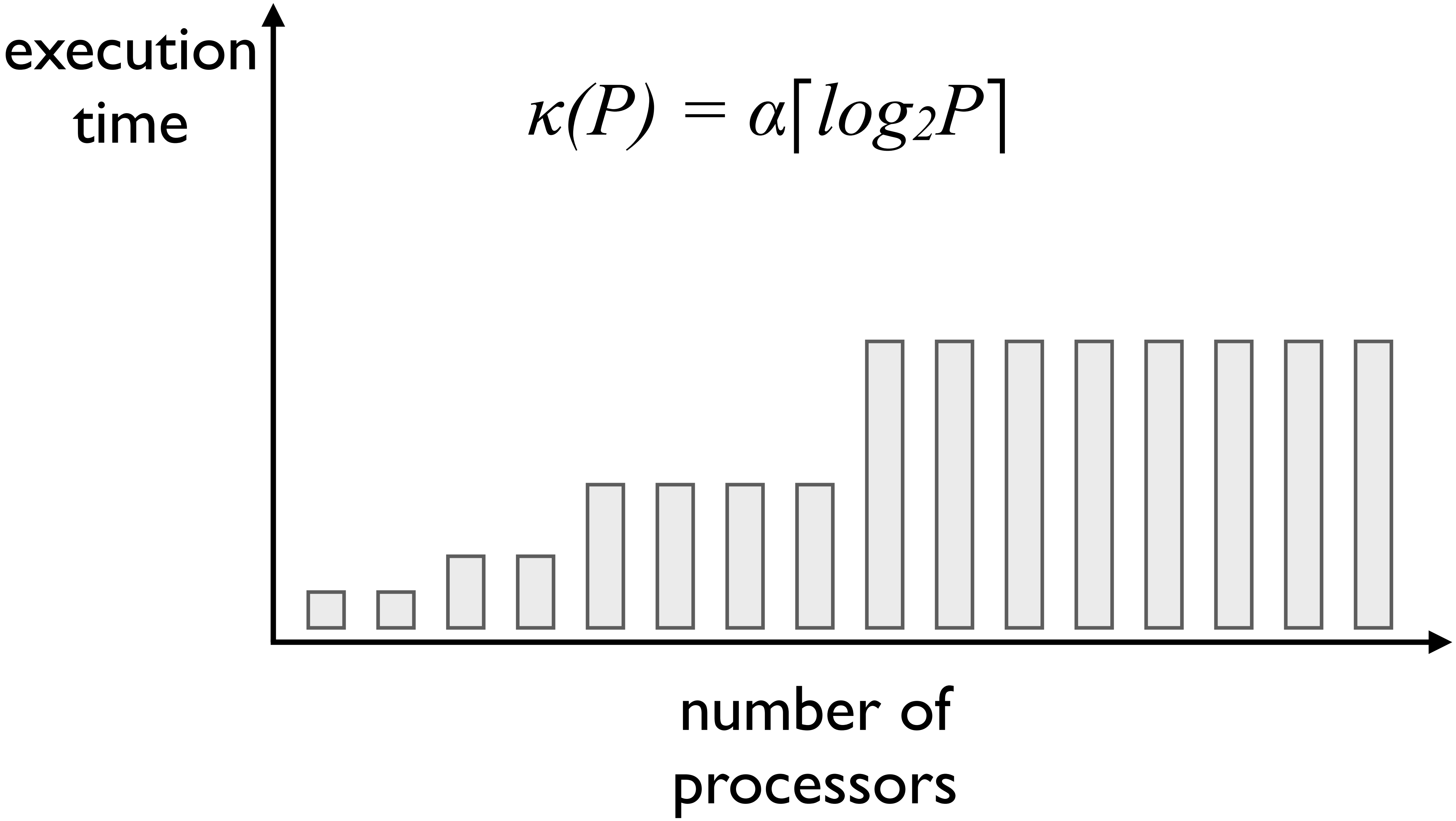
Components of execution time *Inherently Sequential Execution time*



Components of execution time
Parallel time

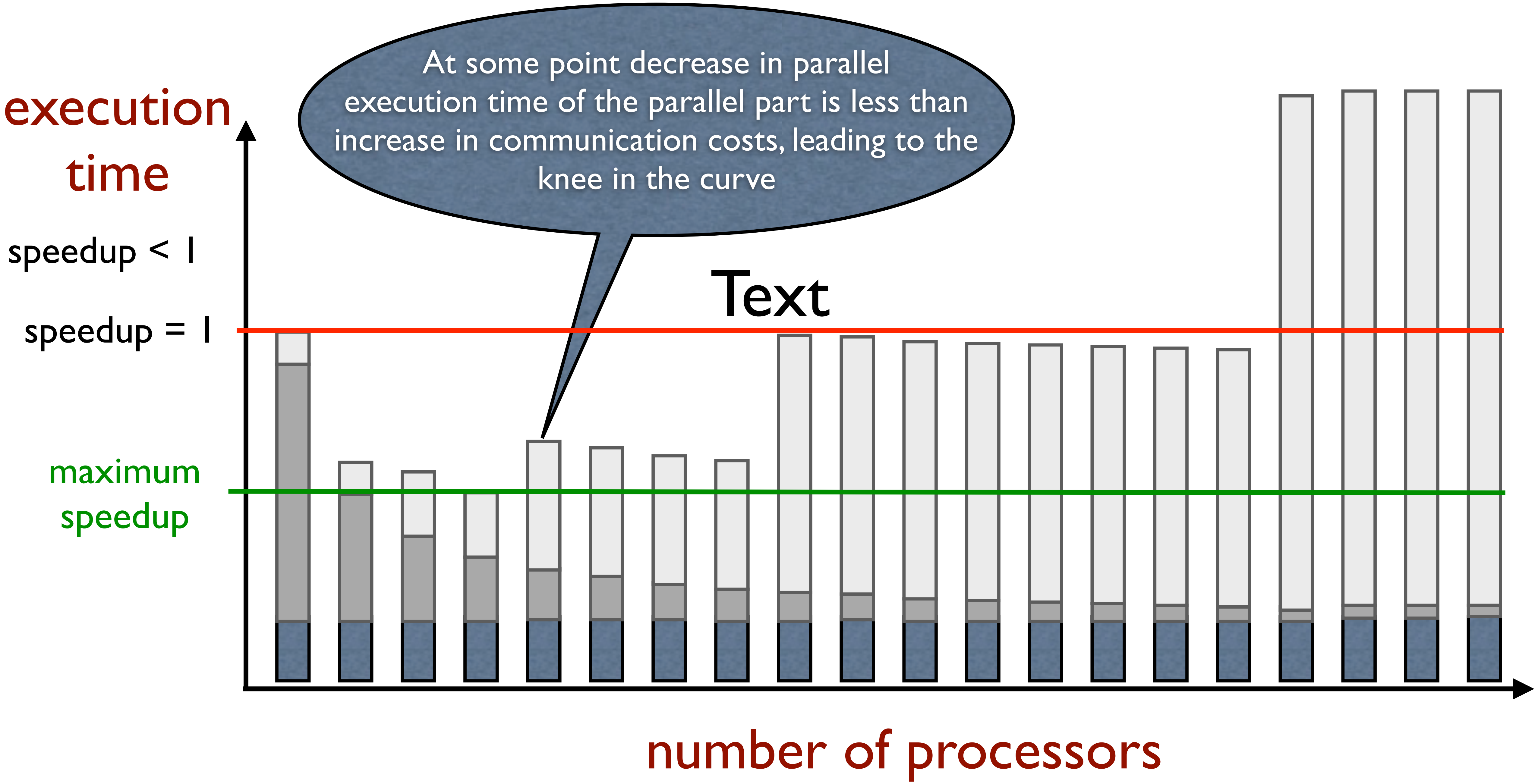


Components of execution time
Communication time and other parallel overheads



$$\kappa(P) = \alpha\lceil log_2 P\rceil$$

Components of execution time
Sequential time



At some point decrease in parallel execution time of the parallel part is less than increase in communication costs, leading to the knee in the curve

Speedup as a function of these components

T_S - sequential time

$$\psi(n,p) \leq \frac{\sigma(n)+\phi(n)}{\sigma(n)+\phi(n)/p+\kappa(n,p)}$$

$T_P(p)$- parallel time

- Sequential time is
  i. the sequential computation
  ii. the parallel computation
- Parallel time is
  i. the sequential computation
  ii. the (parallel computation) / (number of processors)
  iii. the communication cost

Efficiency

$$efficiency \le \frac{sequential\ \ execution\ \ time}{num\ \ processors \times parallel\ \ execution\ \ time}$$

$0 < \varepsilon(n,p) \le 1$

$$\epsilon(n,p) \le \frac{\sigma(n)+\phi(n)}{p(\sigma(n)+\phi(n)/p+p\kappa(n,p))}$$

all terms > 0,
$\varepsilon(n,p) > 0$

$$\epsilon(n,p) \le \frac{\sigma(n)+\phi(n)}{p\sigma(n)+\phi(n)+p\kappa(n,p)}$$

numerator ≤ denominator ≤ 1

Intuitively, *efficiency* is how effectively the machines are being used by the parallel computation

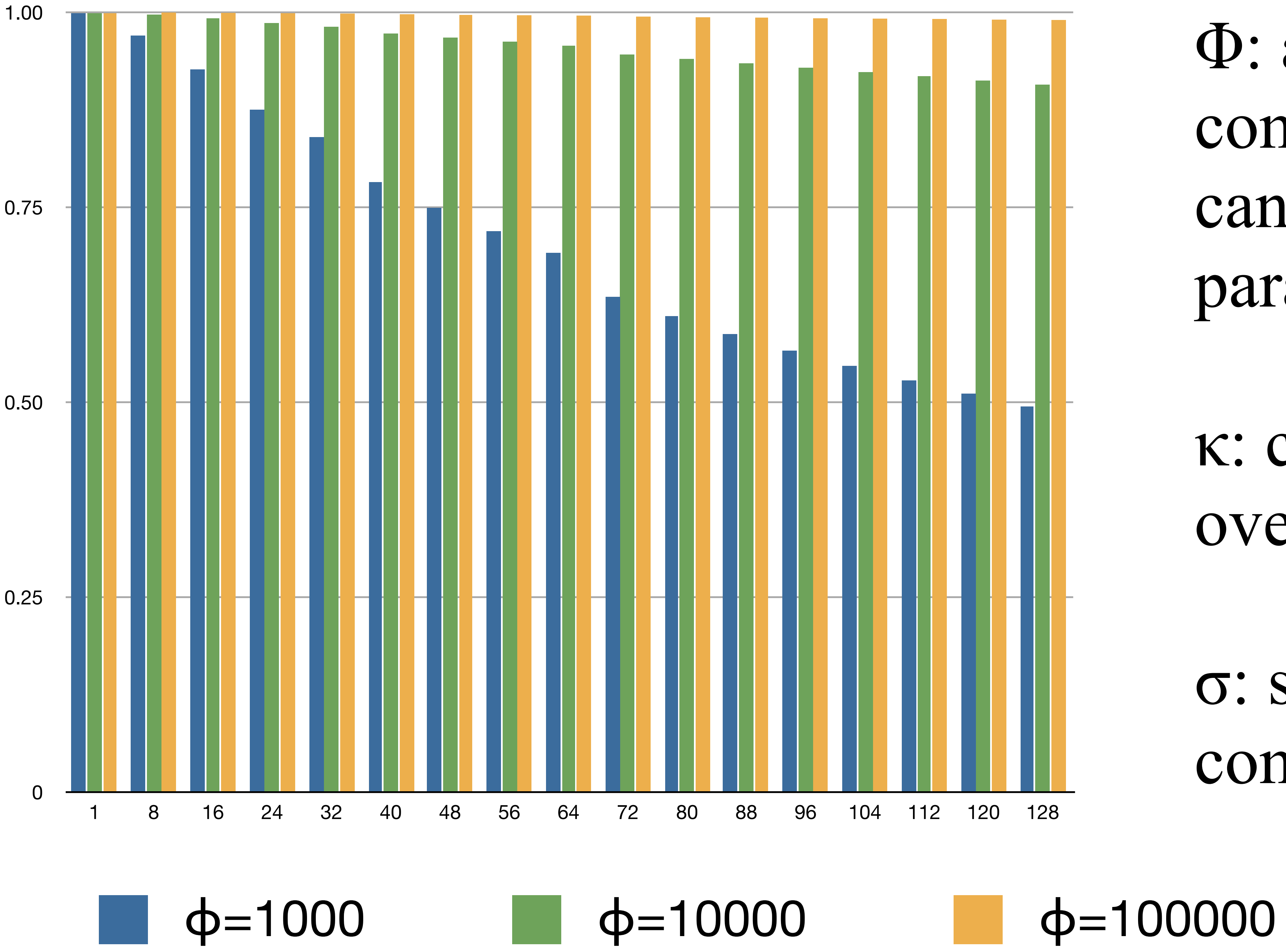
If the number of processors is doubled, for the efficiency to stay the same the parallel execution time Tp must be halved.

Easy efficiency

- Compute speedup
- divide by number of processors
- efficiency = speedup/p

Efficiency by amount of work

σ=1, κ = 1 when p = 1, κ increases by log2 P



Φ: amount of computation that can be done in parallel

κ: communication overhead

σ: sequential computation

Amdahl's Law

- Developed by Gene Amdahl
- Basic idea: the parallel performance of a program is limited by the sequential portion of the program
  - argument for fewer, faster processors
- Can be used to model performance on various sizes of machines, and to derive other useful relations.

Gene Amdahl

- Worked on IBM 704, 709, Stretch and 7030 machines
  - Stretch was first transistorized computer, fastest from 1961 until CDC 6600 in 1964, 1.2 MIPS
  - Multiprogramming, memory protection, generalized interrupts, the 8-bit byte, Instruction pipelining, prefetch and decoding introduced in this machine
- Worked on IBM System 360 -- first machine to have a uniform programming model across different performance models
- In technical disagreement with IBM, set up Amdahl Computers to build plug-compatible machines -- later acquired by Hitachi
- Amdahl's law came from discussions with Dan Slotnick (Illiac IV architect at UIUC) and others about future of parallel processing

Amdahl’s law - key insight

$$\psi(1) = \frac{T_{total\_work}}{T_{serial}+T_{parallel}}$$

$$\psi(\infty) = \frac{T_{total\_work}}{T_{serial}}$$

$\psi(p)$: speedup with p processors

With perfect utilization of parallelism on the parallel part of the job, must take at least T_{serial} time to execute. This observation forms the motivation for Amdahl’s law

As $p \Rightarrow \infty$, $T_{parallel} \Rightarrow 0$ and $\psi(\infty) \Rightarrow (T_{total\ work})/T_{serial}$. Thus, ψ is limited by the serial part of the program.

Two measures of speedup

$$\psi(n,p) \le \frac{\sigma(n)+\phi(n)}{\boxed{\sigma(n)+\phi(n)/p}+\boxed{\kappa(n,p)}}$$

Takes into account communication cost.

- $\sigma(n)$ and $\phi(n)$ are arguably fundamental properties of a program
- $\kappa(n,p)$ is a property of both the program, the hardware, and the library implementations -- arguably a less fundamental concept.
- Can formulate a meaningfully approximation to the speedup without $\kappa(n,p)$

$$\psi(n,p) \le \frac{\sigma(n)+\phi(n)}{p\sigma(n)+\phi(n)}$$

Speedup in terms of the serial fraction of a program

$$\psi(n,p) \leq \frac{\sigma(n)+\phi(n)}{\sigma(n)+\phi(n)/p}$$

Given this formulation, the fraction of the program that is serial is simply $f \leq \frac{\sigma(n)}{\sigma(n)+\phi(n)}$

Speedup can be rewritten in terms of *f*: $\psi(p) \leq \frac{1}{f+(1-f)/p}$

This gives us Amdahl's Law.

Amdahl's Law □ speedup

$$
\begin{aligned}
speedup &= \frac{1}{f + (1-f)/p} \\
&= \frac{1}{\frac{\sigma(n)}{\sigma(n)+\phi(n)} + \left(1 - \frac{\sigma(n)}{\sigma(n)+\phi(n)}\right)/p} \\
&= \frac{\sigma(n)+\phi(n)}{\sigma(n)+\phi(n)} \cdot \frac{1}{\frac{\sigma(n)}{\sigma(n)+\phi(n)} + (1 - \frac{\sigma(n)}{\sigma(n)+\phi(n)})/p} \\
&= \frac{\sigma(n)+\phi(n)}{\sigma(n) + (\sigma(n)+\phi(n)-\sigma(n))/p} \\
&= \frac{\sigma(n)+\phi(n)}{\sigma(n)+\phi(n)/p}
\end{aligned}
$$

Example of using Amdahl's Law

A program is 90% parallel. What speedup can be expected when running on four, eight and 16 processors?

$$\psi(p) \;\leq\; 3.077 = \frac{1}{0.1 + (1 - 0.1)/4}$$

$$\psi(p) \;\leq\; 4.71 = \frac{1}{0.1 + (0.9)/8}$$

$$\psi(p) \;\leq\; 6.4 = \frac{1}{0.1 + (0.9)/16}$$

What is the efficiency of this program?

$$\epsilon(p) \;\leq\; 0.769 = \frac{3.077}{4}$$

$$\epsilon(p) \;\leq\; 0.589 = \frac{4.71}{8}$$

$$\epsilon(p) \;\leq\; 0.4 = \frac{6.4}{16}$$

A *2X* increase in machine cost gives you a *1.4X* increase in performance.

And this is optimistic since communication costs are not considered.

Another Amdahl's Law example

A program is 20% inherently serial. Given 2, 16 and infinite processors, how much speedup can we get?

$$\psi(p) \ \leq \ 1.67 = \frac{1}{0.2 + (0.8)/2}$$

$$\psi(p) \ \leq \ 4 = \frac{1}{0.2 + (0.8)/16}$$

$$\psi(p) \ \leq \ 5 = \frac{1}{0.2 + (0.8)/\infty}$$

Limitation of Amdahl’s Law

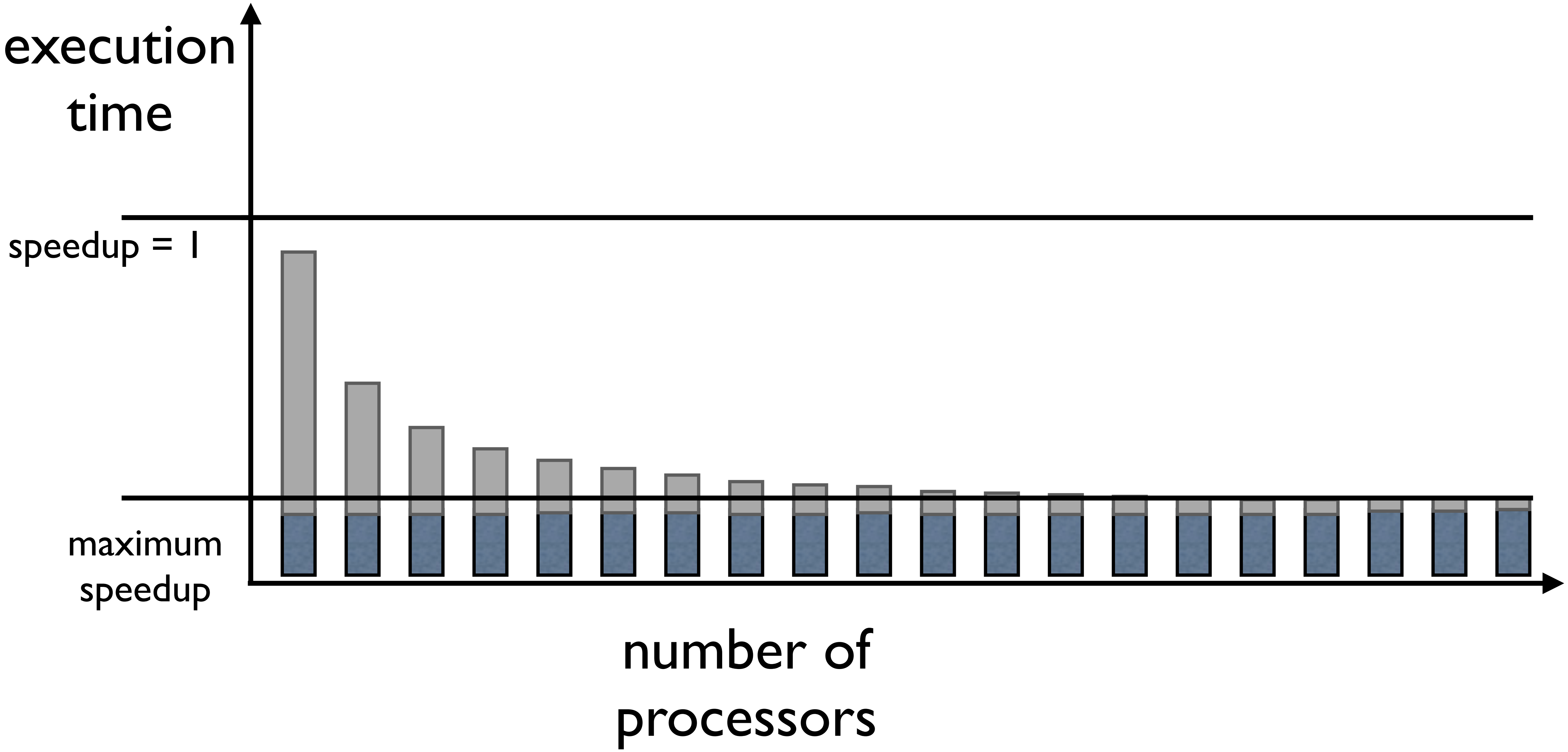
$$\psi(p) = 5 = \frac{1}{0.2+(0.8)/\infty}$$

This result is a limit, not a realistic number.

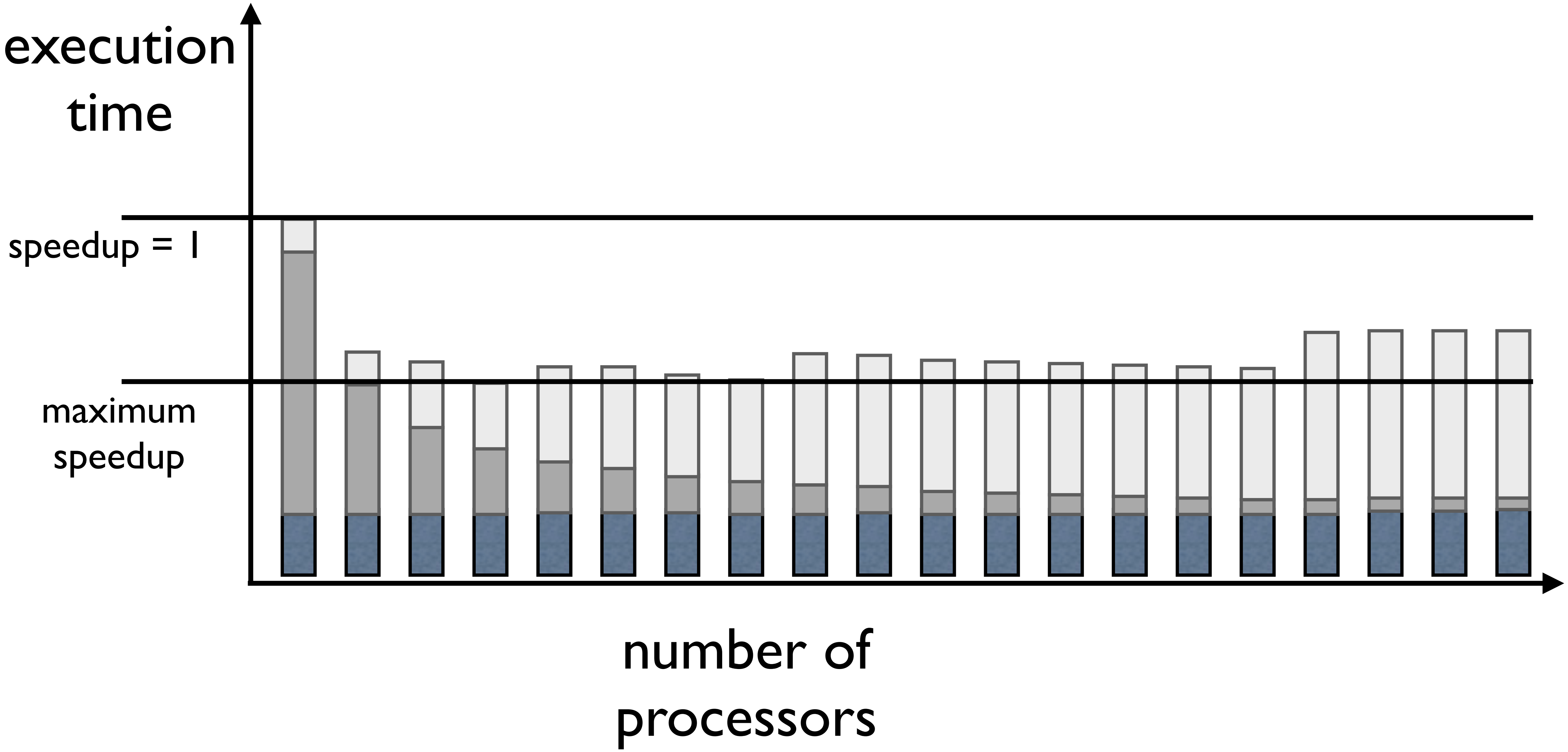
The problem is that communication costs (*κ(n,p)*) are ignored, and this is an overhead that is worse than fixed (which *f* is), but actually grows with the number of processors.

Amdahl’s Law is too optimistic *and may target the wrong problem*

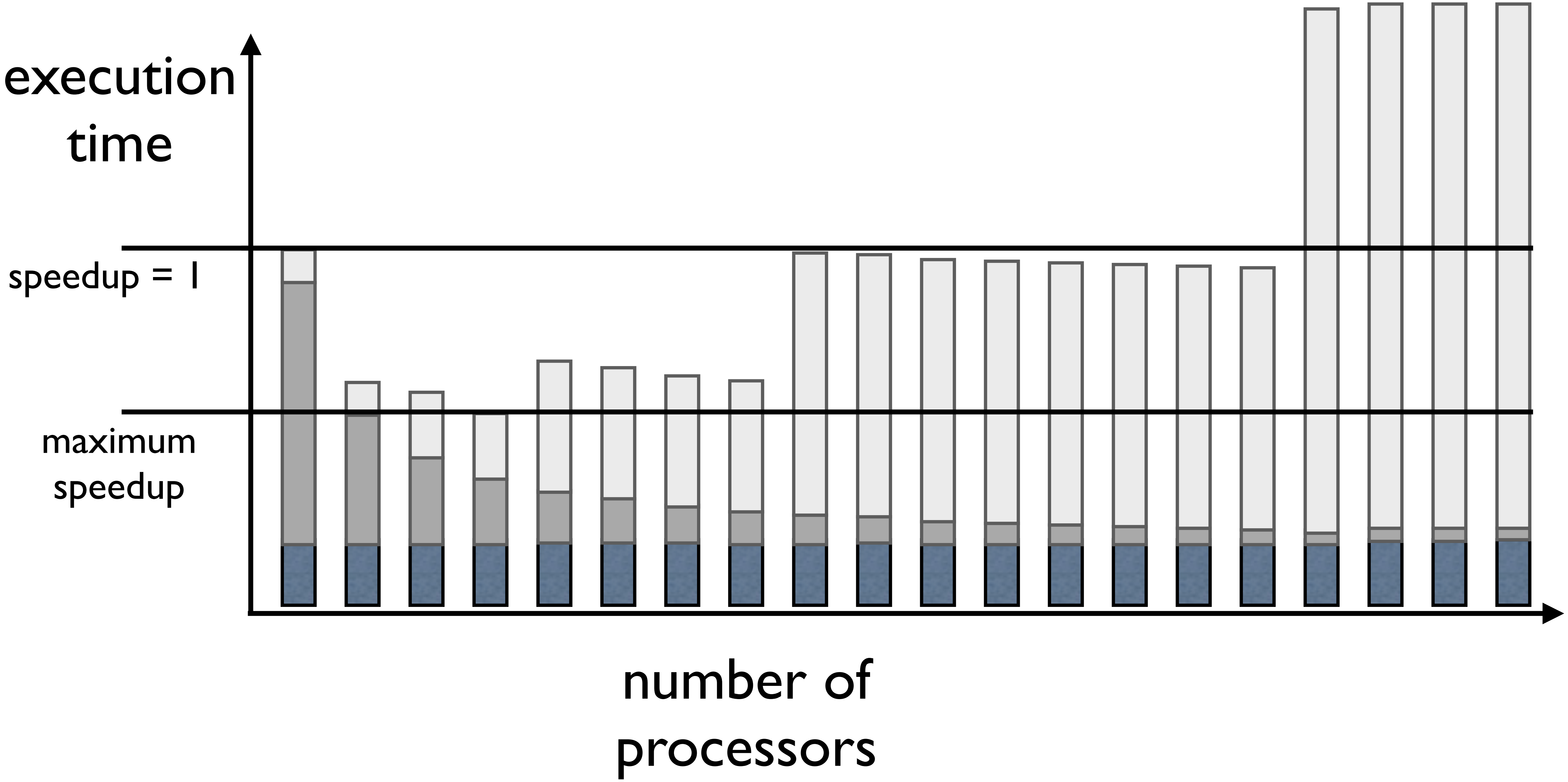
No communication overhead



$O(Log_2P)$ communication costs



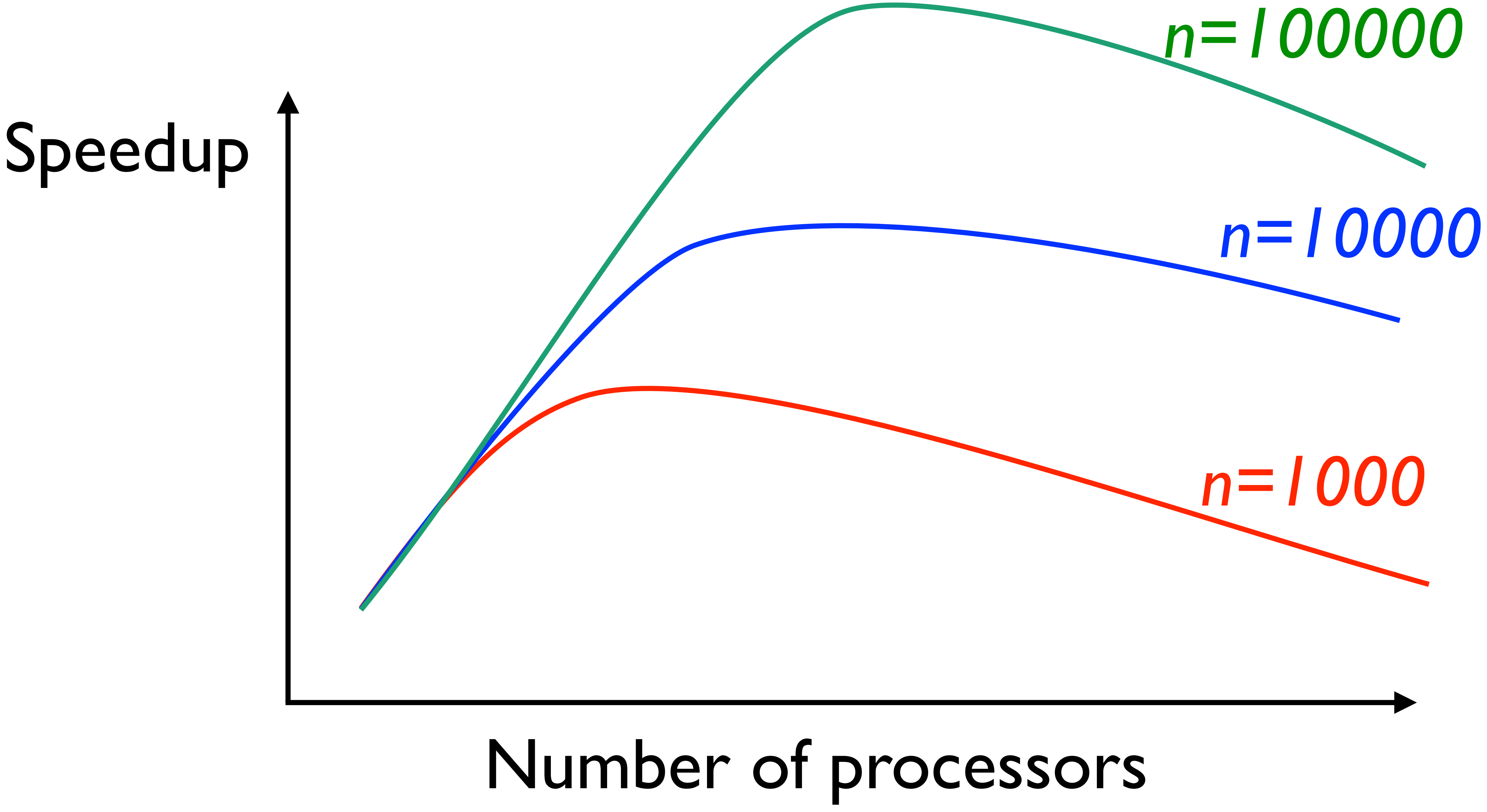
O(P) Communication Costs



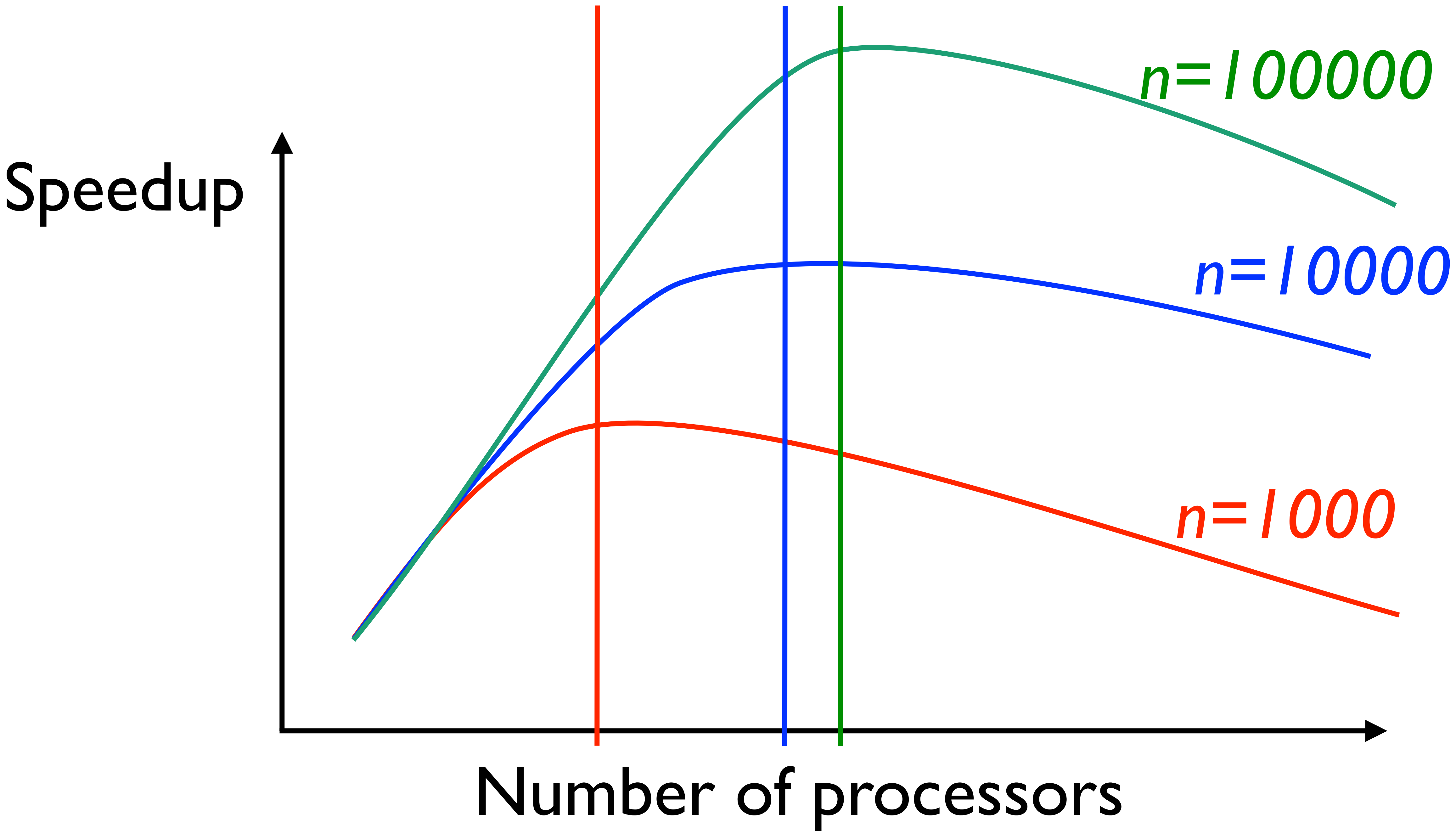
Amdahl Effect

- Complexity of $\phi(n)$ usually higher than complexity of $\kappa(n,p)$ (i.e. computational complexity usually higher than complexity of communication -- same is often true of $\sigma(n)$ as well.) $\phi(n)$ usually $O(n)$ or higher
  - $\kappa(n,p)$ often $O(1)$ or $O(log_2 P)$
- Increasing n allows $\phi(n)$ to dominate $\kappa(n,p)$
- Thus, increasing n increases the speedup Ψ for some number of processors
- Another ~~cheat~~ technique to get good results -- make n large
- Most benchmarks have standard sized inputs to preclude this

Amdahl Effect



Amdahl Effect both increases speedup and move the knee of the curve to the right



Summary

- Allows speedup to be computed for
  - fixed problem size n
  - varying number of processes
- Ignores communication costs
- Is optimistic, but gives an upper bound

Gustafson-Barsis' Law

How does speedup scale with larger problem sizes?

Given a fixed amount of time, how much bigger of a problem can we solve by adding more processors?

Large problem sizes often correspond to better resolution and precision on the problem being solved.

Basic terms

Speedup is $\psi(n,p) \leq \frac{\sigma(n)+\phi(n)}{\sigma(n)+\phi(n)/p+\kappa(n,p)}$

Because $\kappa(n,p) > 0$ $\psi(n,p) \leq \frac{\sigma(n)+\phi(n)}{\sigma(n)+\phi(n)/p}$

Let s be the fraction of time in a *parallel execution* of the program that is spent performing sequential operations.

Then, $(1\text{-}s)$ is the fraction of time spent in a *parallel execution* of the program performing parallel operations.

Note that Amdahl's Law looks at the sequential and parallel parts of the program for a given problem size, and the value of f is the fraction in a sequential execution that is inherently sequential

Or stated differently . . .

Speedup in terms of the serial fraction of a program

$$\psi(n,p) \leq \frac{\sigma(n)+\phi(n)}{\sigma(n)+\phi(n)/p}$$

Given this formulation, the fraction of sequential execution that is serial is simply $f \leq \frac{\sigma(n)}{\sigma(n)+\phi(n)}$

Speedup can be rewritten in terms of *f*: $\psi(p) \leq \frac{1}{f+(1-f)/p}$

This gives us Amdahl's Law.

Note number of processors not mentioned for definition of *f* because *f* is for time in a sequential run

Some definitions

The sequential part of a ***parallel*** computation:

$$s = \frac{\sigma(n)}{\sigma(n)+\phi(n)/p}$$

The parallel part of a parallel computation:

$$(1 - s) \ = \ \frac{\phi(n)/p}{\sigma(n) + \phi(n)/p}$$

And the speedup

$$\psi(n,p) \ \leq \ \frac{\sigma(n) + \phi(n)}{\sigma(n) + \phi(n)/p}$$

G-B Law speedup

$$\text{speedup} = s + (1\text{-}s)p$$
$$= p + (1\text{-}s)p$$

The serial portion in G-B is a fraction of the *parallel* execution time of the program. *To use G-B Law we assume work scales to maintain value of* s

Deriving G-B Law

First, we show that the formula circled in blue leads to our speedup formula.

substitute for (s + (1 - s)p)

$$\psi(n,p) \le \frac{(\sigma(n)+\frac{\phi(n)}{p})(s+(1-s)p)}{\sigma(n)+\frac{\phi(n)}{p}}$$

$$\le \frac{\left(\sigma(n)+\frac{\phi(n)}{p}\right)\left(\frac{\sigma(n)}{\sigma(n)+\frac{\phi(n)}{p}}+\left(1-\frac{\sigma(n)}{\sigma(n)+\frac{\phi(n)}{p}}\right)p\right)}{\sigma(n)+\frac{\phi(n)}{p}}$$

$$\le \frac{\frac{\sigma(n)\left(\sigma(n)+\frac{\phi(n)}{p}\right)}{\sigma(n)+\frac{\phi(n)}{p}}+\left(\sigma(n)+\frac{\phi(n)}{p}-\frac{\sigma(n)\left(\sigma(n)+\frac{\phi(n)}{p}\right)}{\sigma(n)+\frac{\phi(n)}{p}}\right)}{\sigma(n)+\frac{\phi(n)}{p}}$$

Multiply through

$$\le \frac{\sigma(n)+p\sigma(n)+\phi(n)-p\sigma(n)}{\sigma(n)+\frac{\phi(n)}{p}}$$

$$\le \frac{\sigma(n)+\phi(n)}{\sigma(n)+\frac{\phi(n)}{p}}$$

simplify, simply

Deriving G-B Law

Second, we show that the formula circled in blue (that we just showed is equivalent to speedup) leads to the G-B Law formula.

$$\psi(n,p) \quad \leq \quad \frac{\left(\sigma(n) + \frac{\phi(n)}{p}\right)(s + (1-s)p)}{\sigma(n) + \frac{\phi(n)}{p}}$$

$$\psi(n,p) \quad \leq \quad s + (1-s)p$$

$$\psi(n,p) \quad \leq \quad p + (1-p)s$$

$$\begin{aligned} s + (1-s)p &= s + p - pS \\ &= p + (1-p)s \end{aligned}$$

An example

An application executing on 64 processors requires 220 seconds to run. It is experimentally determined through benchmarking that 5% of the time is spent in the serial code on a single processor. What is the scaled speedup of the application?

$s = 0.05$, thus on 64 processors
$\Psi = 64 + (1-64)(0.05) = 64 - 3.15 = 60.85$

An example

Another way of looking at this: given P processors, P amount of useful work can be done, except on P-1 processors there is time wasted due to the sequential part that must be subtracted out from the useful work.

$s = 0.05$, thus on 64 processors

$\Psi = 64 + (1-64)(0.05) = 64 - 3.15 = 60.85$

Another example

You have money to buy a 16K (*16,384*) core distributed memory system, but you only want to spend the money if you can get decent performance on your application. Allowing the problem to scale with increasing numbers of processors, what must s be to get a scaled speedup of *15,000* on the machine, i.e. what percentage of the application's *parallel* execution time can be devoted to inherently serial computation?

$$\begin{aligned} 15{,}000 &= 16{,}384 - 16{,}383s \\ \Rightarrow\ s &= 1{,}384 / 16{,}383 \\ \Rightarrow\ s &= \mathbf{0.084} \end{aligned}$$

Comparison with Amdahl's Law result

$$\psi(n,p) \le p + (1 - p)s$$

$$15{,}000 = 16{,}384 - 16{,}383s$$

$$\Rightarrow s = 1{,}384 / 16{,}383$$

$$\Rightarrow s = \mathbf{0.084}$$

G-B almost 1% can be sequential

$$\psi(p) \le \frac{1}{f + (1-f)/p}$$

$$15,000 \le \frac{1}{f + (1-f)/16,384}$$

$$15,000f(p-1) = p - 15,000$$

$$f = \frac{p - 15,000)}{15,000(p-1)}$$

$$f = 0.0000056$$

!

Amdahl's law (56 millionths)

Comparison with Amdahl's Law result

$$\psi(n,p) \le p + (1 - p)s$$

$$15{,}000 = 16{,}384 - 16{,}383s$$

$$\Rightarrow s = 1{,}384 / 16{,}383$$

$$\Rightarrow s = \mathbf{0.084}$$

$$\psi(p) \le \frac{1}{f + (1 - f)/p}$$

$$15,000 \le \frac{1}{f + (1 - f)/16,384}$$

$$15,000 f(p - 1) = p - 15,000$$

$$f = \frac{p - 15,000)}{15,000(p - 1)}$$

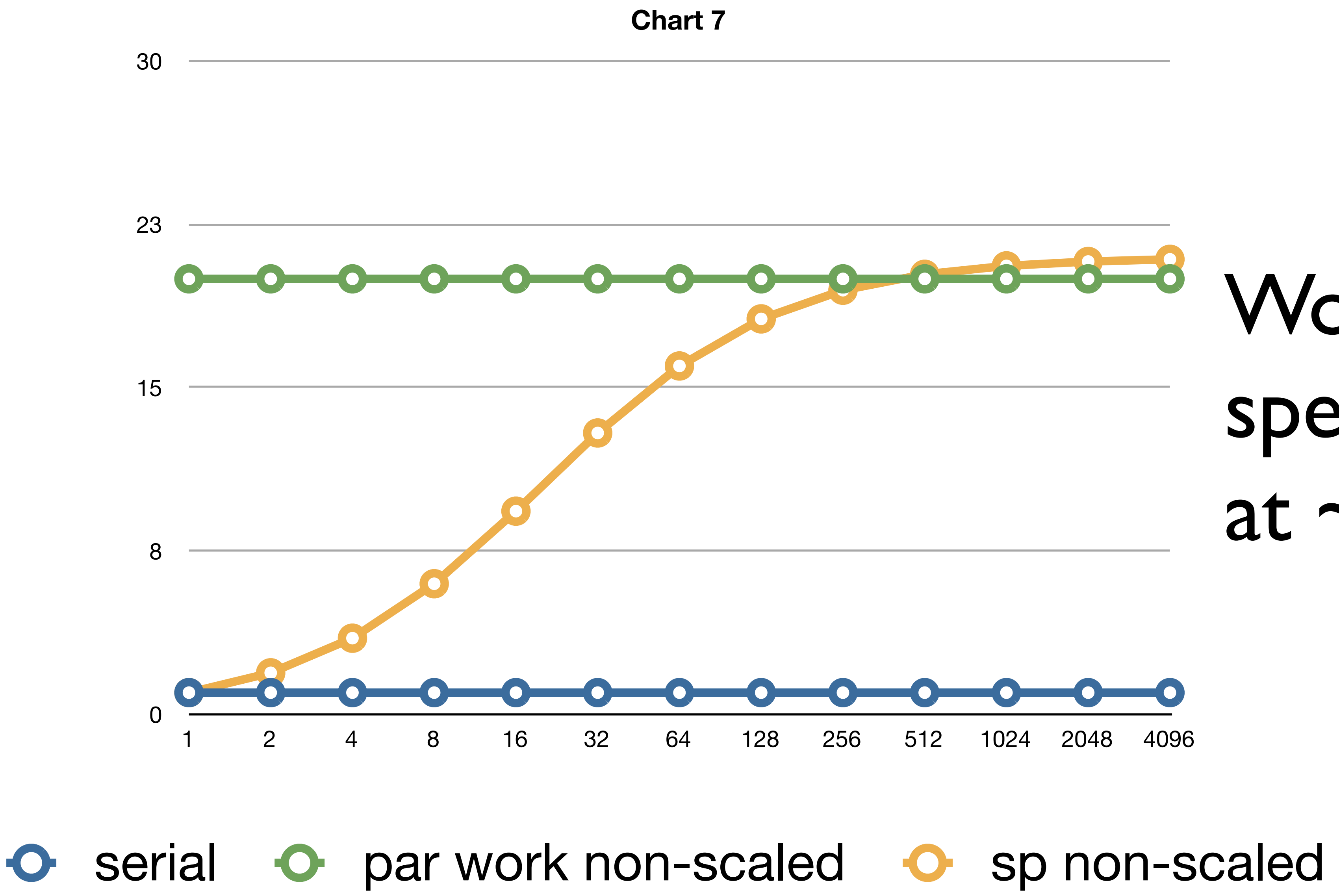
$$f = 0.0000056$$

But then Amdahl's law doesn't allow the problem size to scale.

!

Non-scaled performance

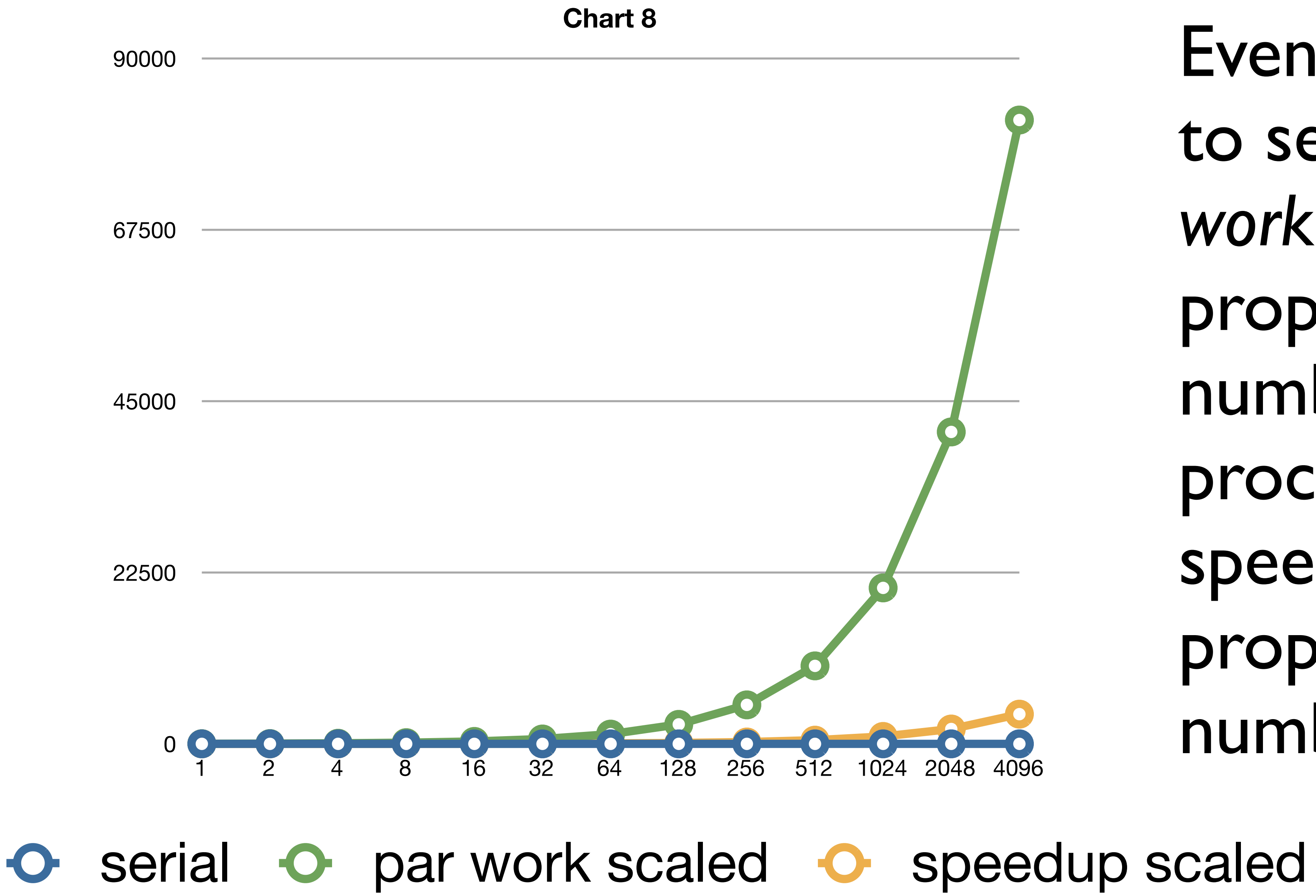
$$\sigma(1) = \sigma(p); \phi(1) = \phi(p)$$



Work is constant, speedup levels off at ~256 processors

performance

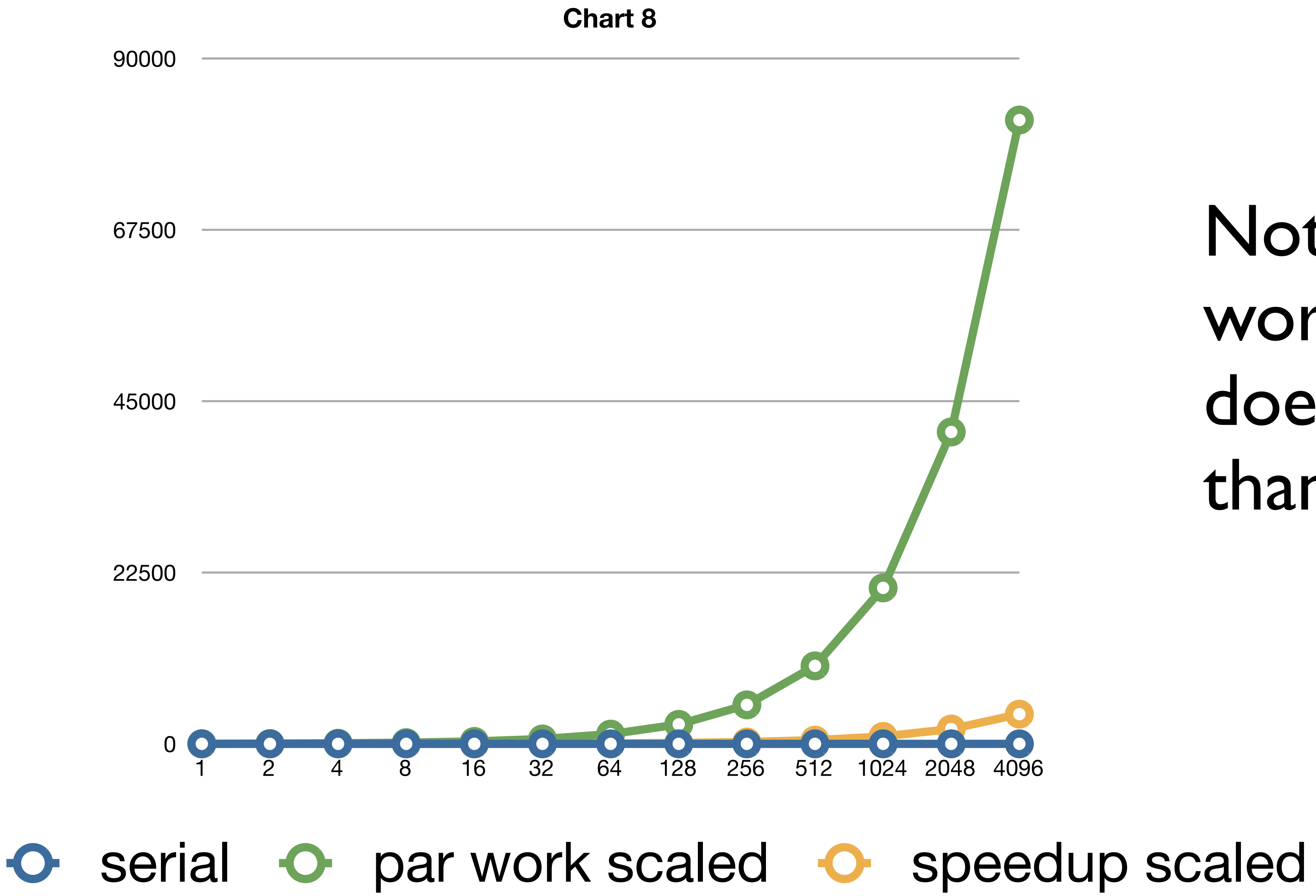
$$\sigma(1) = \sigma(p);\ p \cdot \phi(1) = \phi(p)$$



Even though it is hard to see, as the *parallel work* increases proportionally to the number of processors, the speedup scales proportionally to the number of processors

performance

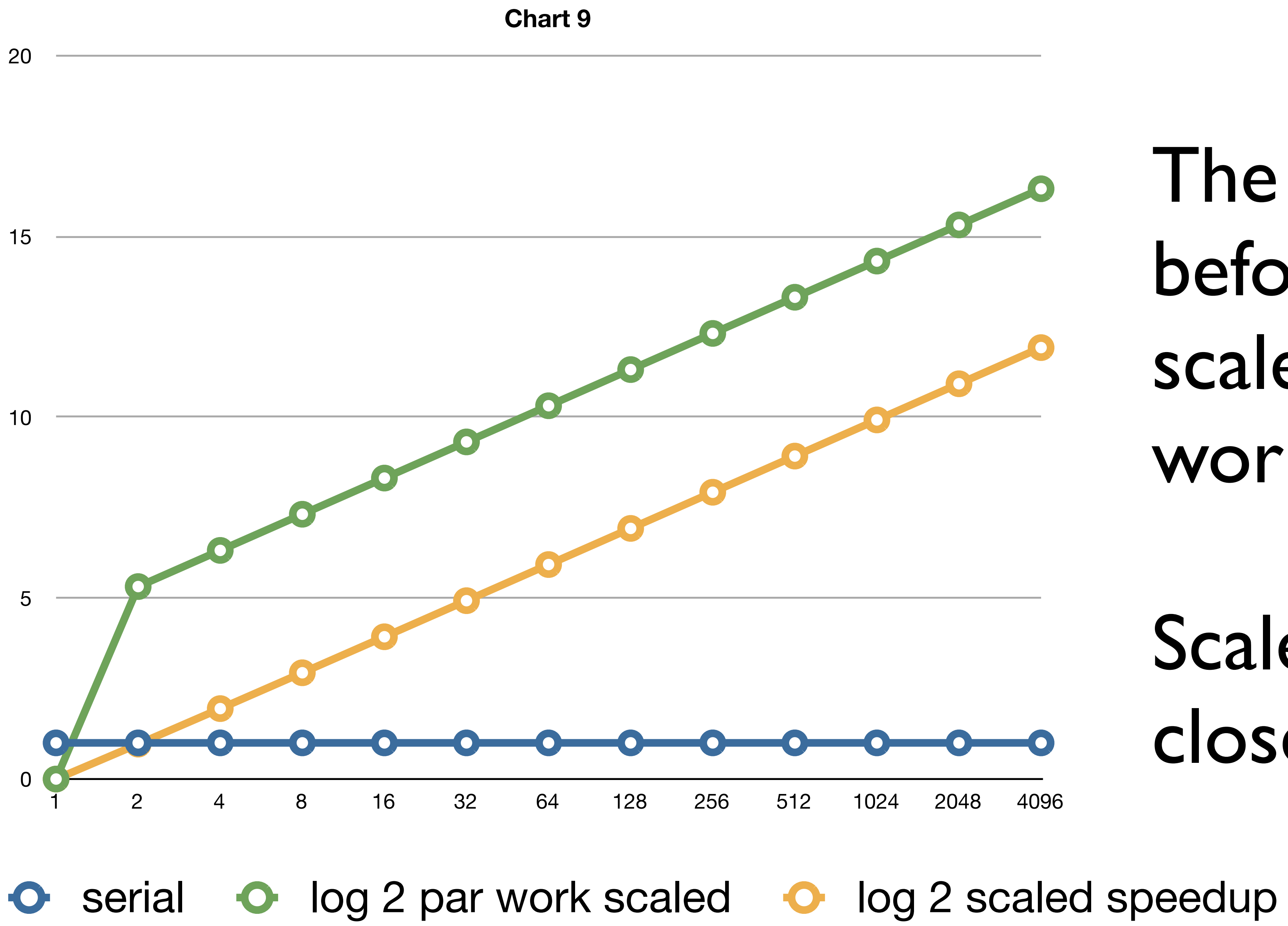
$$\sigma(1) = \sigma(p);\ p\cdot\phi(1) = \phi(p)$$



Note that the parallel work may (and usually does) increase faster than the problem size

Scaled speedups, log scales

$$\sigma(1) = \sigma(p); p \cdot \phi(1) = \phi(1)$$

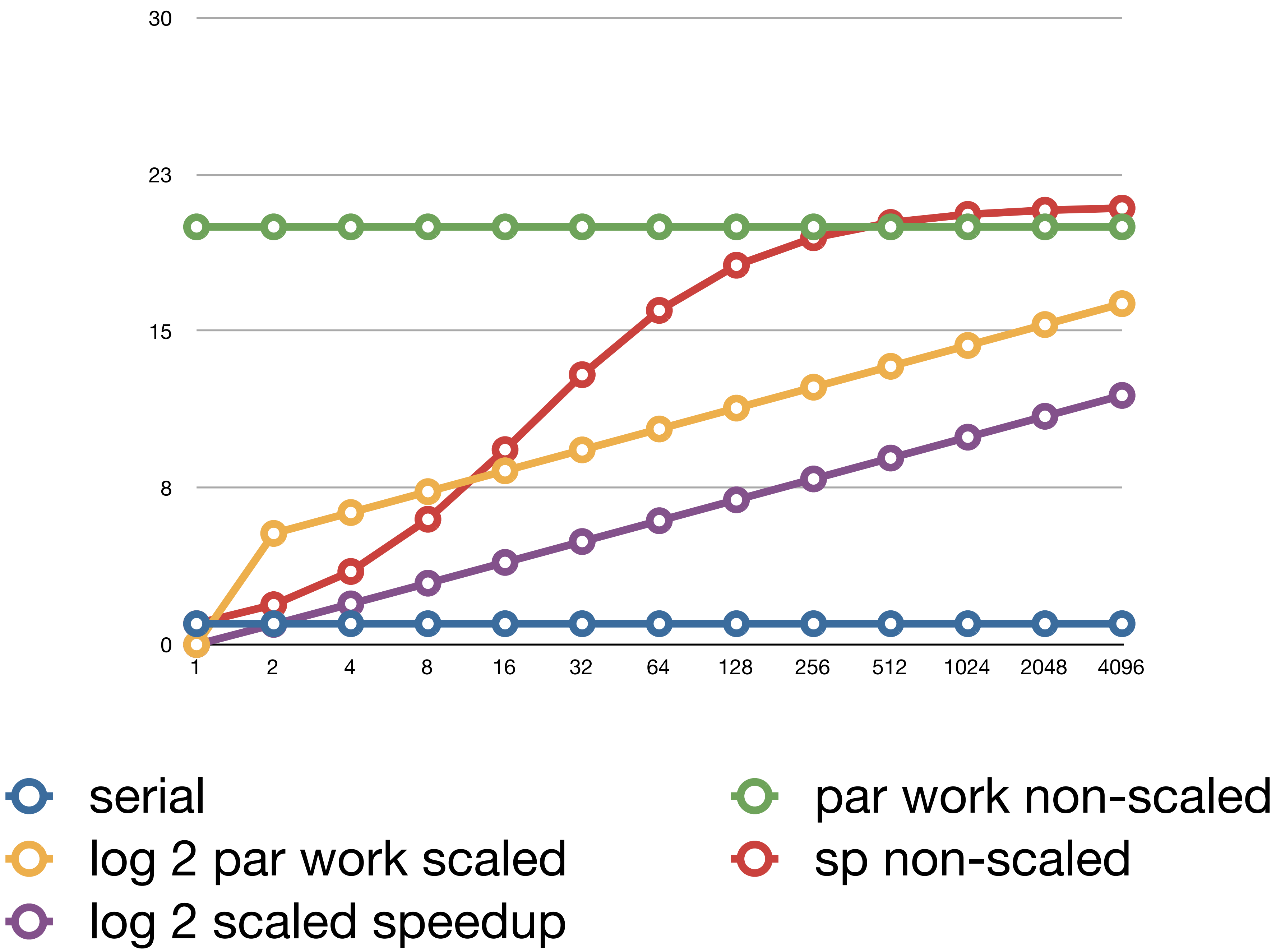


The same chart as before, except log scales for parallel work and speedup.

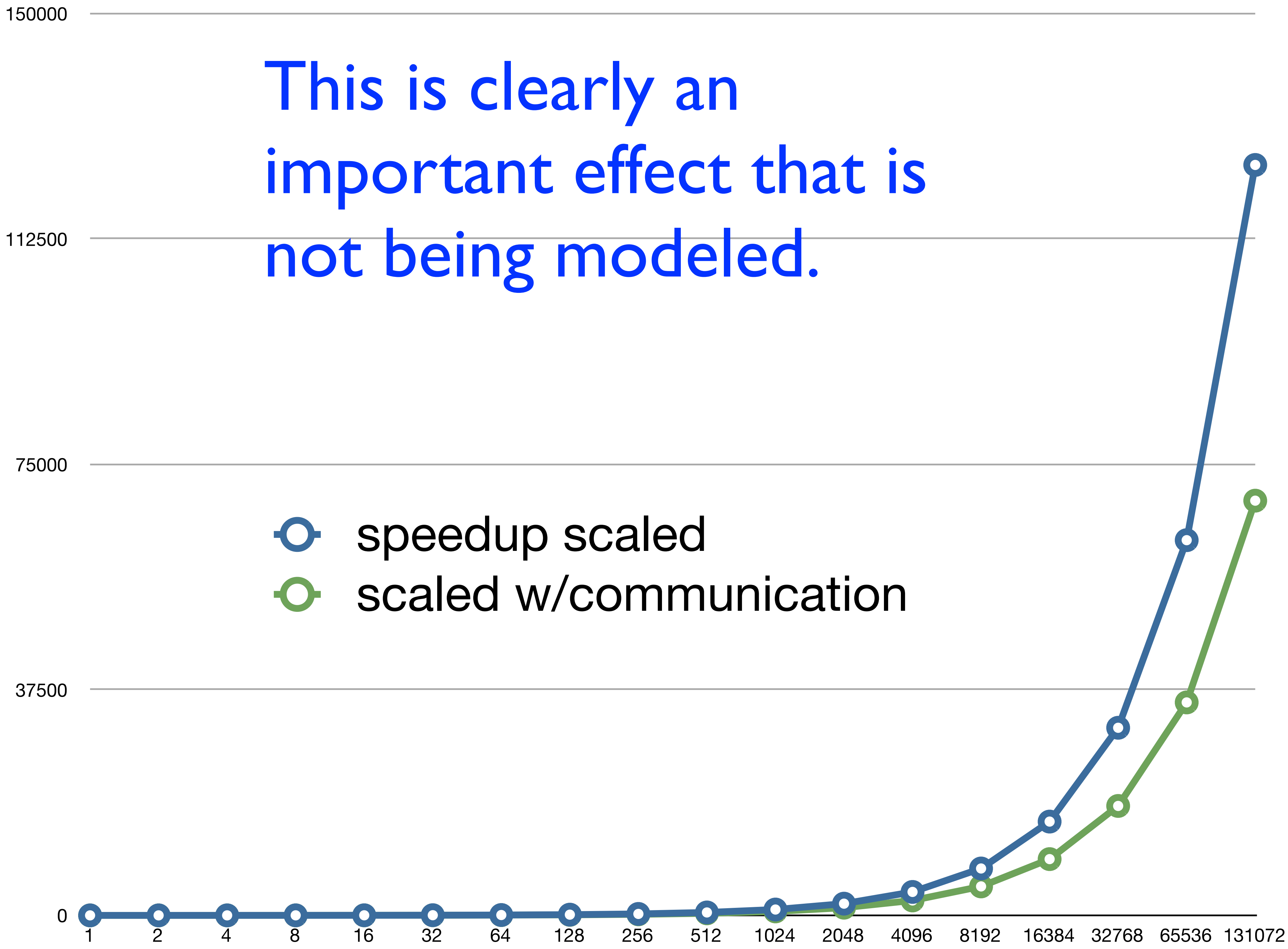
Scaled speedup close to ideal

Scaled and non-scaled speedups, log$_2$ axis for scaled

$$\sigma(1) = \sigma(p);\ p\cdot\phi(1) = \phi(p)$$



The effect of un-modeled communication



This is clearly an important effect that is not being modeled.

The Karp-Flatt Metric

- Takes into account communication costs
- $T(n,p) = \sigma(n) + \phi(n)/p + \kappa(n,p)$
- Serial time $T(n,1) = \sigma(n) + \phi(n)$
- The *experimentally determined serial fraction* e of the parallel computation is

  $e = (\sigma(n) + \kappa(n,p))/T(n,1)$

$$e = (\sigma(n) + \kappa(n,p))/T(n,1)$$

Essentially a measure of total work

- e is the fraction of the one processor execution time that is serial on all p processors
- Communication cost mandates measuring at a given processor count
  - This is because communication cost is a function of both theoretical limits and implementation.

Deriving the K-F Metric

The *experimentally determined serial fraction* e of the parallel computation is

$$e = (\sigma(n) + \kappa(n,p))/T(n,1)$$

$$e \cdot T(n,1) = \sigma(n) + \kappa(n,p)$$

The parallel execution time

$$T(n,p) = \sigma(n) + \phi(n)/p + \kappa(n,p)$$

can now be rewritten as

$$T(n,p) = T(n,1)\cdot e + T(n,1)(1 - e)/p$$

fraction of time that is parallel * total time is parallel time - a good approximation of $\phi(n)$

Let ψ represent $\psi(n,p)$, and

$$\psi = T(n,1)/T(n,p)$$

then

$$T(n,1) = T(n, p)\psi.$$

Therefore

$$T(n,p) = T(n,p)\psi e + T(n,p)\psi(1-e)/p$$

Deriving the K-F Metric

The *experimentally determined serial fraction* e of the parallel computation is

$$e = (\sigma(n) + \kappa(n,p))/T(n,1)$$

$$e \cdot T(n,1) = \sigma(n) + \kappa(n,p)$$

The parallel execution time

$$T(n,p) = \sigma(n) + \phi(n)/p + \kappa(n,p)$$

can now be rewritten as

$$T(n,p) = T(n,1) \cdot e + T(n,1)(1 - e)/p$$

Let ψ represent $\psi(n,p)$, and

$$\psi = T(n,1)/T(n,p)$$

The standard formula

then

Divide

$$T(n,1) = T(n, p)\psi.$$

Therefore

$$T(n,p) = T(n,p)\psi e + T(n,p)\psi(1-e)/p$$

Deriving the K-F Metric

The *experimentally determined serial fraction* e of the parallel computation is

$$e = (\sigma(n) + \kappa(n,p))/T(n,1)$$

$$e \cdot T(n,1) = \sigma(n) + \kappa(n,p)$$

The parallel execution time

$$T(n,p) = \sigma(n) + \phi(n)/p + \kappa(n,p)$$

can now be rewritten as

$$T(n,p) = T(n,1) \cdot e + T(n,1)(1 - e)/p$$

Let ψ represent $\psi(n,p)$, and

$$\psi = T(n,1)/T(n,p)$$

then

$$T(n,1) = T(n, p)\psi.$$

Therefore

$$T(n,p) = T(n,p)\psi e + T(n,p)\psi(1-e)/p$$

Total execution time

Experimentally determined serial fraction

Total time * serial fraction is the serial time

Deriving the K-F Metric

The *experimentally determined serial fraction* e of the parallel computation is

$$e = (\sigma(n) + \kappa(n,p))/T(n,1)$$

$$e \cdot T(n,1) = \sigma(n) + \kappa(n,p)$$

Total execution time

The parallel execution time

$$T(n,p) = \sigma(n) + \phi(n)/p + \kappa(n,p)$$

can now be rewritten as

$$T(n,p) = T(n,1) \cdot e + T(n,1)(1 - e)/p$$

Let ψ represent $\psi(n,p)$, and

$$\psi = T(n,1)/T(n,p)$$

then

$$T(n,1) = T(n, p)\psi.$$

Therefore

$$T(n,p) = T(n,p)\psi e + T(n,p)\psi(1-e)/p$$

fraction of time that is **sequential**

(Total time * parallel part)/p is the parallel time

Karp-Flatt Metric

$T(n,p) = T(n,p)\psi e + T(n,p)\psi(1\text{-}e)/p \Rightarrow$

$1 = \psi e + \psi(1\text{-}e)/p \Rightarrow$

$1/\psi = e + (1\text{-}e)/p \Rightarrow$

$1/\psi = e + 1/p - e/p \Rightarrow$

$1/\psi = e(1\text{-}1/p) + 1/p \Rightarrow e = \frac{1/\psi - 1/p}{1 - 1/p}$

Remember $e = (\sigma(n) + \kappa(n,p))/T(n,1)$. Then, if e gets bigger, either (a) sequential part ($\sigma(n)$ is larger, (b) total work on 1 processor gets larger ($T(n,1)$) in algorithm is larger, or (c) $\kappa(n,p)$ is larger. (a) and (b) don’t happen, so it must be (c).

What is it good for?

- Takes into account the parallel overhead ($\kappa(n,p)$) ignored by Amdahl's Law and Gustafson-Barsis.
- Helps us to detect other sources of inefficiency ignored in these (sometimes too simple) models of execution time
  - $\phi(n)/p$ may not be accurate because of load balance issues or work not dividing evenly into $c{\cdot}p$ chunks.
  - other interactions with the system may be causing problems
- Can determine if the efficiency drop with increasing p for a fixed size problem is
  a. because of limited parallelism
  b. because of increases in algorithmic or architectural overhead

Example

Benchmarking a program on 1, 2, ..., 8 processors produces the following speedups:

p	2	3	4	5	6	7	8
ψ	1.82	2.50	3.08	3.57	4.00	4.38	4.71

Why is the speedup only 4.71 on 8 processors?

p	2	3	4	5	6	7	8
ψ	1.82	2.50	3.08	3.57	4.00	4.38	4.71
e	0.10	0.10	0.10	0.10	0.10	0.10	0.10

$$e = (1/3.57 - 1/5)/(1-1/5) = (0.08)/.8 = 0.\ 1$$

Example 2

Benchmarking a program on 1, 2, ..., 8 processors produces the following speedups:

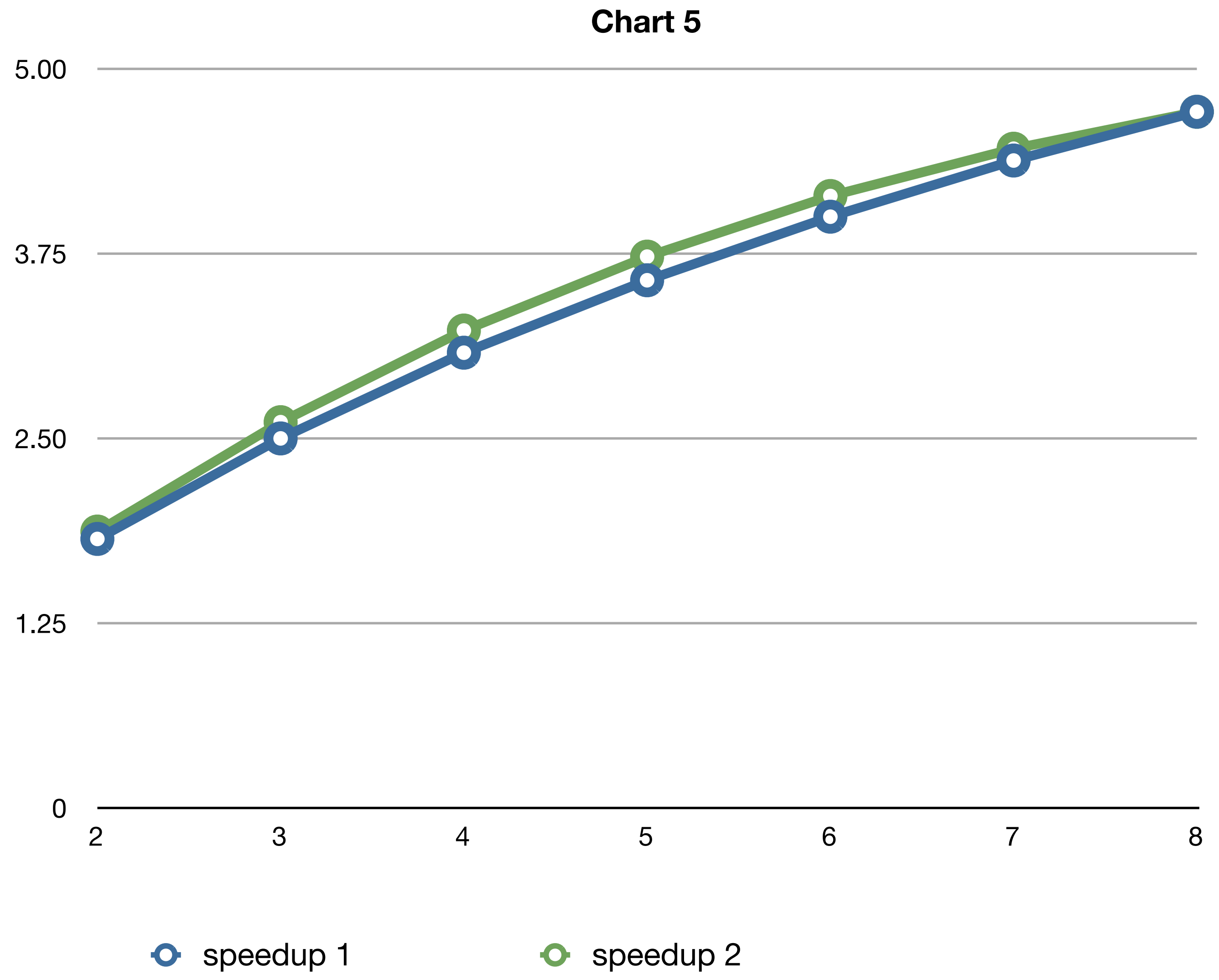
p	2	3	4	5	6	7	8
ψ	1.87	2.61	3.23	3.73	4.14	4.46	4.71

Why is the speedup only 4.71 on 8 processors?

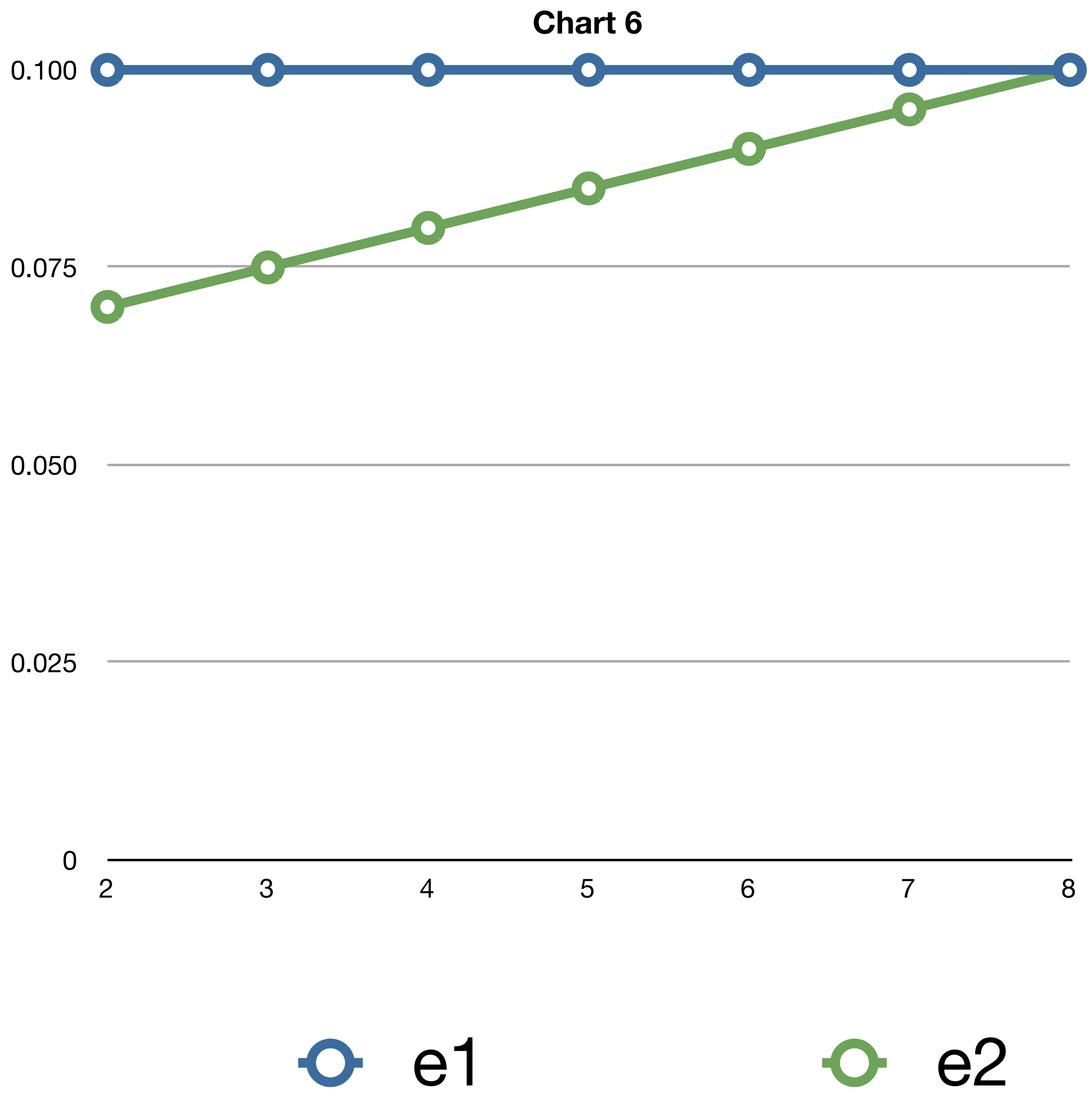
p	2	3	4	5	6	7	8
ψ	1.87	2.61	3.23	3.73	4.14	4.46	4.71
e	0.07	0.075	0.080	0.085	0.090	0.095	0.1

e is increasing: speedup problem is increasing serial overhead (process startup, communication, algorithmic issues, the architecture of the parallel system, etc.

Which has the efficiency problem?



Very easy to see using e



Isoefficiency Metric Overview

- Parallel system: parallel program executing on a parallel computer
- Scalability of a parallel system: measure of its ability to increase performance as number of processors increases
- A scalable system maintains efficiency as processors are added
- Isoefficiency: way to measure scalability

Isoefficiency Derivation Steps

- Begin with speedup formula
- Compute total amount of overhead
- Assume efficiency remains constant
- Determine relation between sequential execution time and overhead

Deriving Isoefficiency Relation

Determine overhead

$$T_o(n,p) = (p-1)\sigma(n) + p\kappa(n,p)$$

Substitute overhead into speedup equation

$$\psi(n,p) \le \frac{p(\sigma(n)+\varphi(n))}{\sigma(n)+\varphi(n)+T_0(n,p)}$$

Substitute T(n,1) = σ(n) + ϕ(n).

Assume efficiency is constant.

$$T(n,1) \ge CT_0(n,p)$$ Isoefficiency Relation

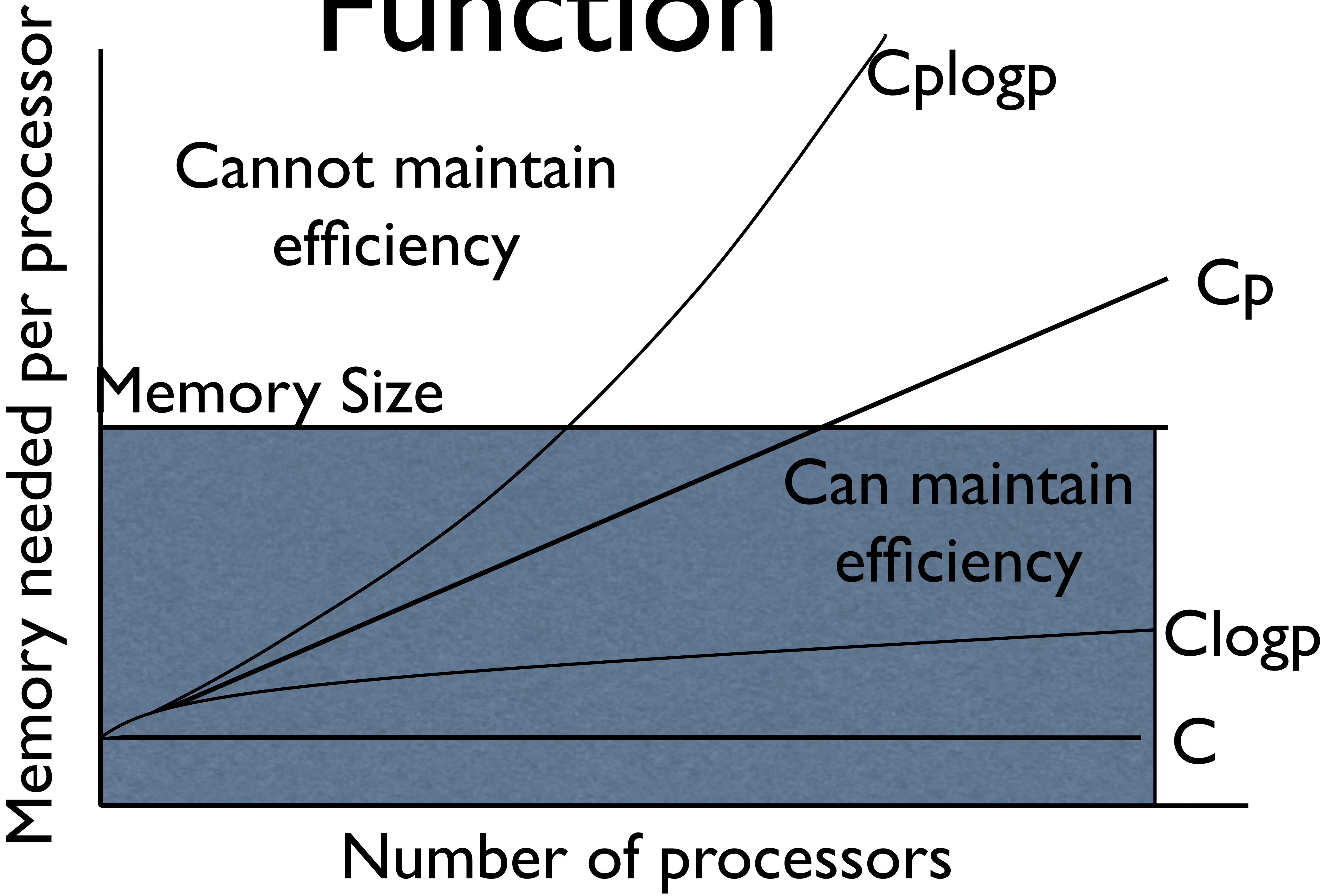
Scalability Function

- Suppose isoefficiency relation is $n \geq f(p)$
- Let $M(n)$ denote memory required for problem of size n
- $M(f(p))/p$ shows how memory usage per processor must increase to maintain same efficiency
- We call $M(f(p))/p$ the scalability function

Meaning of Scalability Function

- To maintain efficiency when increasing p, we must increase n
- Maximum problem size limited by available memory, which is linear in p
- Scalability function shows how memory usage per processor must grow to maintain efficiency
- Scalability function a constant means parallel system is perfectly scalable

Interpreting Scalability Function



Example 1: Reduction

- Sequential algorithm complexity
  $T(n,1) = \Theta(n)$
- Parallel algorithm
  - Computational complexity = $\Theta(n/p)$
  - Communication complexity = $\Theta(\log p)$
- Parallel overhead
  $T_0(n,p) = \Theta(p \log p)$

Reduction (continued)

- Isoefficiency relation: $n \geq C\, p \log p$
- We ask: To maintain same level of efficiency, how must n increase when p increases?
- $M(n) = n$

$$\boxed{M(Cp\log p)/p = Cp\log p/p = C\log p}$$

- The system has good scalability

Example 2: Floyd's Algorithm

- Sequential time complexity: $\Theta(n3)$
- Parallel computation time: $\Theta(n3/p)$
- Parallel communication time: $\Theta(n2\log p)$
- Parallel overhead: $T0(n,p) = \Theta(pn2\log p)$

Floyd’s Algorithm (continued)

- Isoefficiency relation
  n3 ≥ C(p n2 log p) ⇒ n ≥ C p log p
- M(n) = n^2

$$M(Cp\log p)/p = C^2 p^2 \log^2 p/p = C^2 p \log^2 p$$

Example 3: Finite Difference

- Sequential time complexity per iteration: Θ(n2)
- Parallel communication complexity per iteration: Θ(n/√p)
- Parallel overhead: Θ(n √p)

Finite Difference (continued)

- Isoefficiency relation
  n2 ≥ Cn√p ⇒ n ≥ C√ p
- M(n) = n2

$$M(C\sqrt{p})/\,p = C^2 p/\,p = C^2$$

- This algorithm is perfectly scalable

Summary (1/3)

- Performance terms
  - Speedup
  - Efficiency
- Model of speedup
  - Serial component
  - Parallel component
  - Communication component

Summary (2/3)

- What prevents linear speedup?
  - Serial operations
  - Communication operations
  - Process start-up
  - Imbalanced workloads
  - Architectural limitations

Summary (3/3)

- Analyzing parallel performance
    - Amdahl's Law
    - Gustafson-Barsis' Law
    - Karp-Flatt metric
    - Isoefficiency metric