NOVA SCHOOL OF SCIENCE & TECHNOLOGY
COMPUTER SCIENCE DEPARTMENT

Parallel Programming Models and Architectures

lecture 03 (2021-03-22)

Master in Computer Science and Engineering

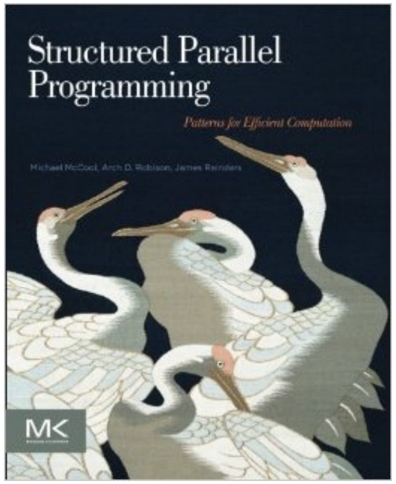
— Concurrency and Parallelism / 2020-21 —

João Lourenço <joao.lourenco@fct.unl.pt>

Outline

- Performance scalability
  - Analytical performance measures
  - Amdahl’ s law
  - Gustafson-Barsis’ law
  - Work-span and Brent’s lemma

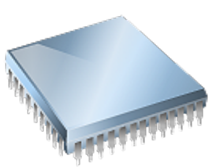
  - Bibliography:
    - **Chapter 2** of book
      McCool M., Arch M., Reinders J.;
      Structured Parallel Programming: Patterns for
      Efficient Computation;
      Morgan Kaufmann (2012);
      ISBN: 978-0-12-415993-8



What is Performance?

- In computing, performance is defined by 2 factors
  - Computational requirements (what needs to be done?) **Efficacy**
  - Computing resources (how much will it cost?) **Efficiency**
- Computational problems translate to requirements
- Computing resources interplay and tradeoff

$$\mathit{Performance} \sim \frac{1}{\mathit{Resources\ for\ solution}}$$


Hardware

Time


Energy

… and ultimately


Money

What is Parallel Performance?

- We are concerned with performance issues when using a parallel computing environment
  - Performance with respect to parallel computation
- Performance is the *raison d'être* for parallelism
  - Parallel performance versus sequential performance
  - If the "performance" is not better, parallelism is not necessary
- *Parallel processing* includes techniques and technologies necessary to compute in parallel
  - Hardware, networks, operating systems, parallel libraries, languages, compilers, algorithms, tools, …
- Parallelism must deliver performance
  - How? How well?

Performance Expectation (Loss)

- If each processor is rated at "f" MFLOPS and there are "p" processors, should we see "$f \times p$" MFLOPS performance?
  - If it takes 100 seconds on 1 processor, shouldn't it take 10 seconds on 10 processors?
- Several causes affect performance
  - Each must be understood separately
  - But they interact with each other in complex ways
    - Solution to one problem may create another
    - One problem may mask another
- Scaling (system, problem size) can change conditions
- Need to understand *performance space*

Embarrassingly Parallel Computations

- An embarrassingly parallel computation is one that can be obviously divided into completely independent parts that can be executed simultaneously
  - In an embarrassingly parallel computation there is no interaction between separate processes, except for the (initial) work distribution and (final) results collection and combination
- Embarrassingly parallel computations have potential to achieve maximal speedup on parallel platforms
  - If it takes T time sequentially, there is the potential to achieve T/P time running in parallel with P processors
  - Why is this not the (usual) case?

Scalability

- Can the program scale up to use many processors?
  - What does that mean?
- How do we measure scalability?
  - How do we evaluate scalability goodness?
    - Comparative evaluation
      - If double the number of processors, what to expect?
      - Is scalability linear?
      - Is efficiency retained as problem size increases?
  - Apply performance metrics

Performance and Scalability

- Performance evaluation
  - *Sequential* runtime (T_{seq} or T_1) is a function of
    - **problem size and architecture**
  - *Parallel* runtime (T_{par}) is a function of
    - **problem size and parallel architecture**
    - # processors used in the execution
  - Performance is affected by
    - **algorithm + architecture**
- Scalability
  - Ability of parallel algorithm to achieve performance gains proportional to the number of processors and the size of the problem

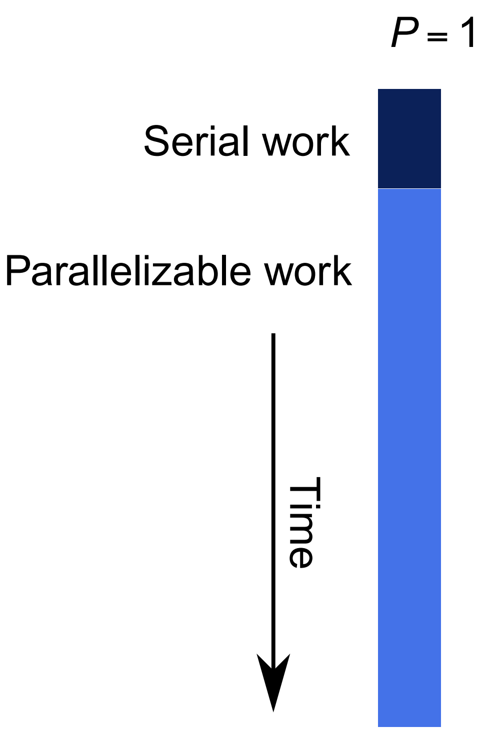
Performance Metrics and Formulas

- T_1 is the execution time on a single processor
- T_p is the execution time on a "p" processor system
- S_p is the speedup $\quad S(p) = \dfrac{T_1}{T_p}$
- E_p is the efficiency $\quad E(p) = \dfrac{S_p}{p}$
- C_p is the cost $\quad Cost(p) = p \times T_p$

- A parallel algorithm is *cost-optimal* if
  - $\sum$ Parallel time = sequential time (E_p = 100%, $C_p = T_1$)

Amdahl's Law (Fixed Size Speedup)

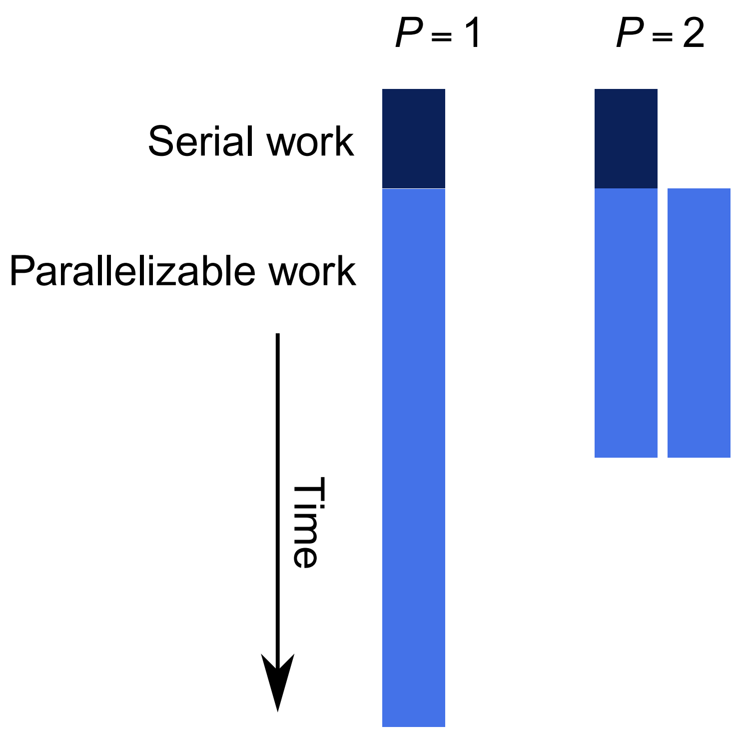
- Interested in solving the problem faster
- *Reduce execution time*



Serial Work (f) is ≈16% of execution time

Amdahl's Law (Fixed Size Speedup)

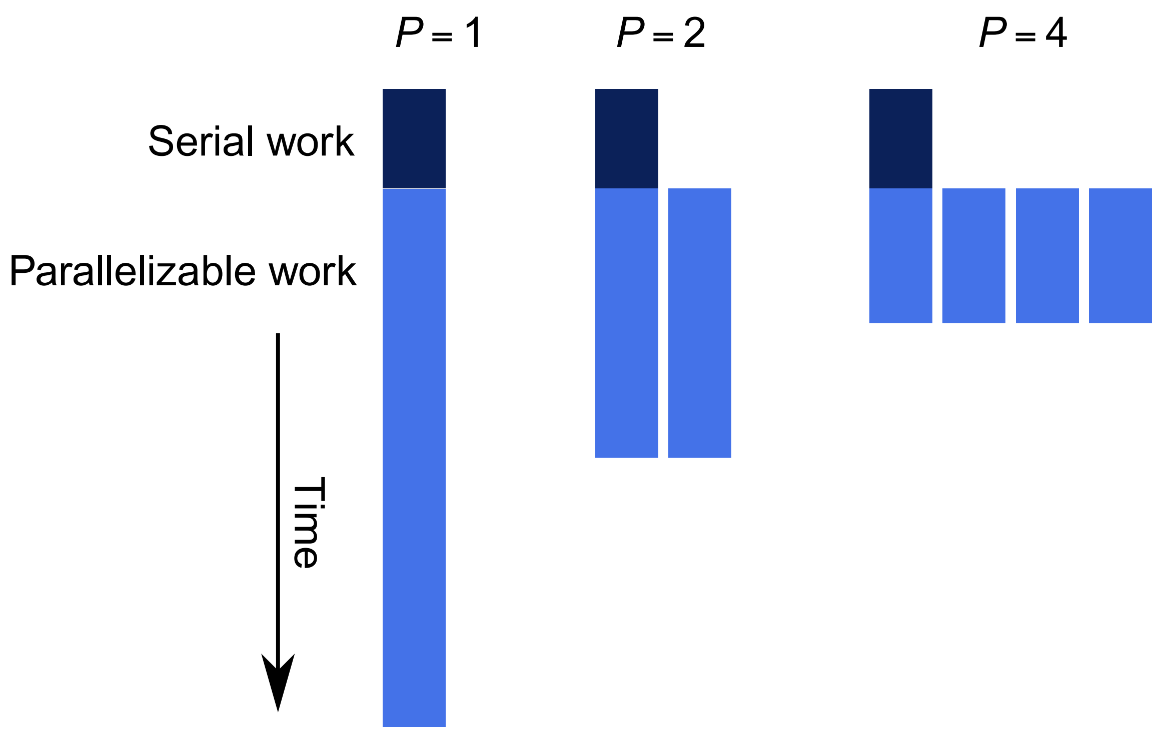
- Interested in solving the problem faster
- *Reduce execution time*



Serial Work (f) is ≈25% of execution time

Amdahl's Law (Fixed Size Speedup)

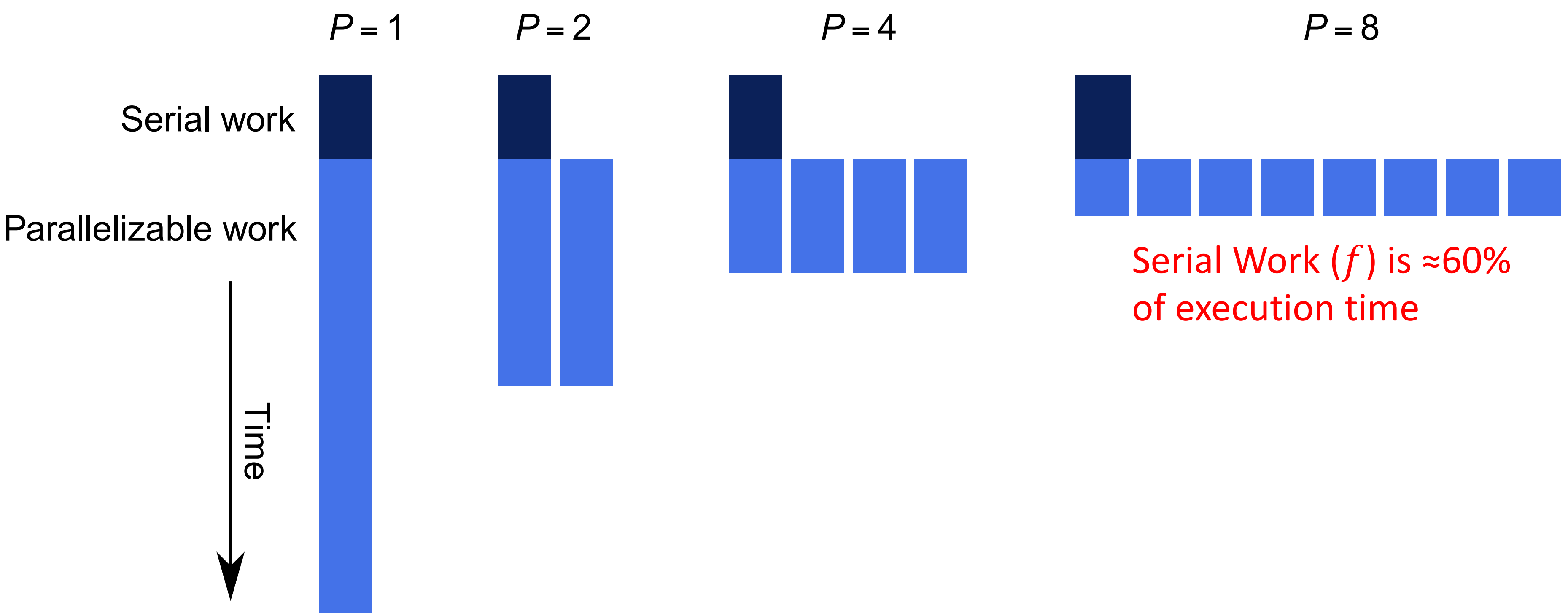
- Interested in solving the problem faster
- *Reduce execution time*



Serial Work (f) is ≈40% of execution time

Amdahl's Law (Fixed Size Speedup)

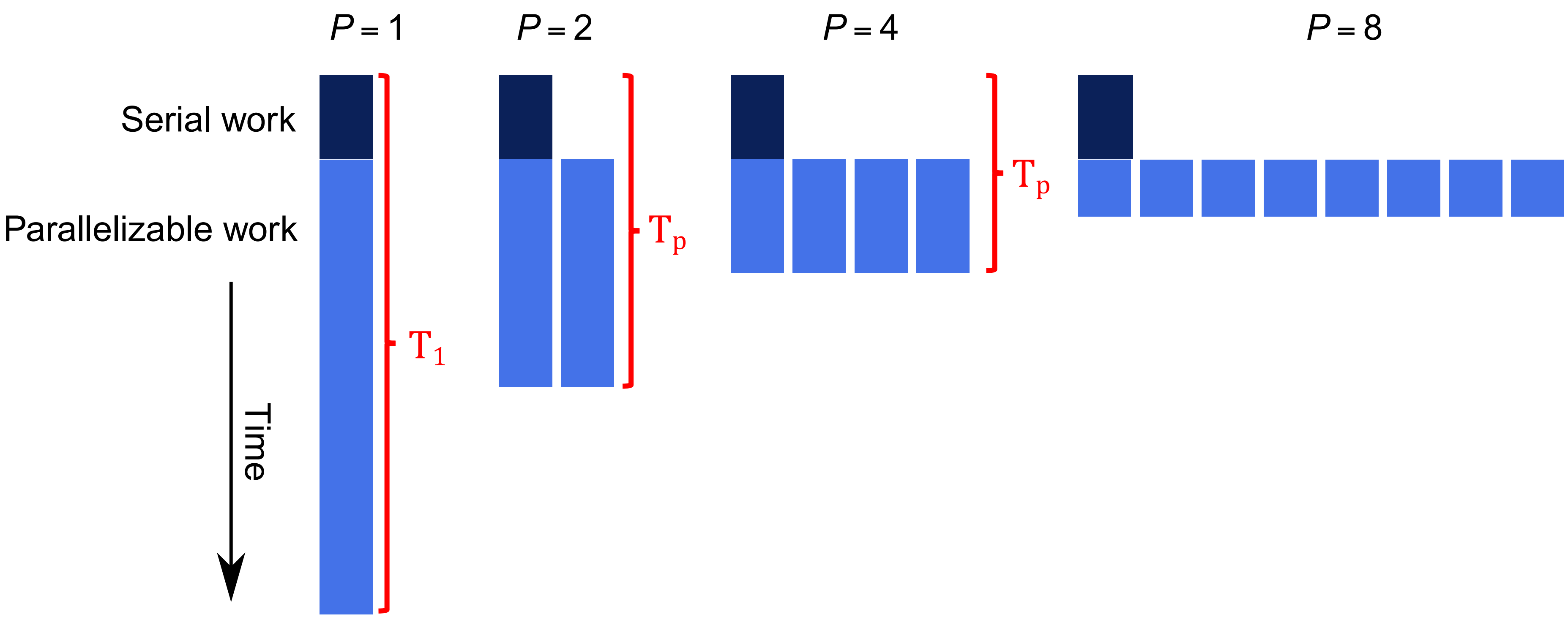
- Interested in solving the problem faster
- *Reduce execution time*



Serial Work (f) is ≈60% of execution time

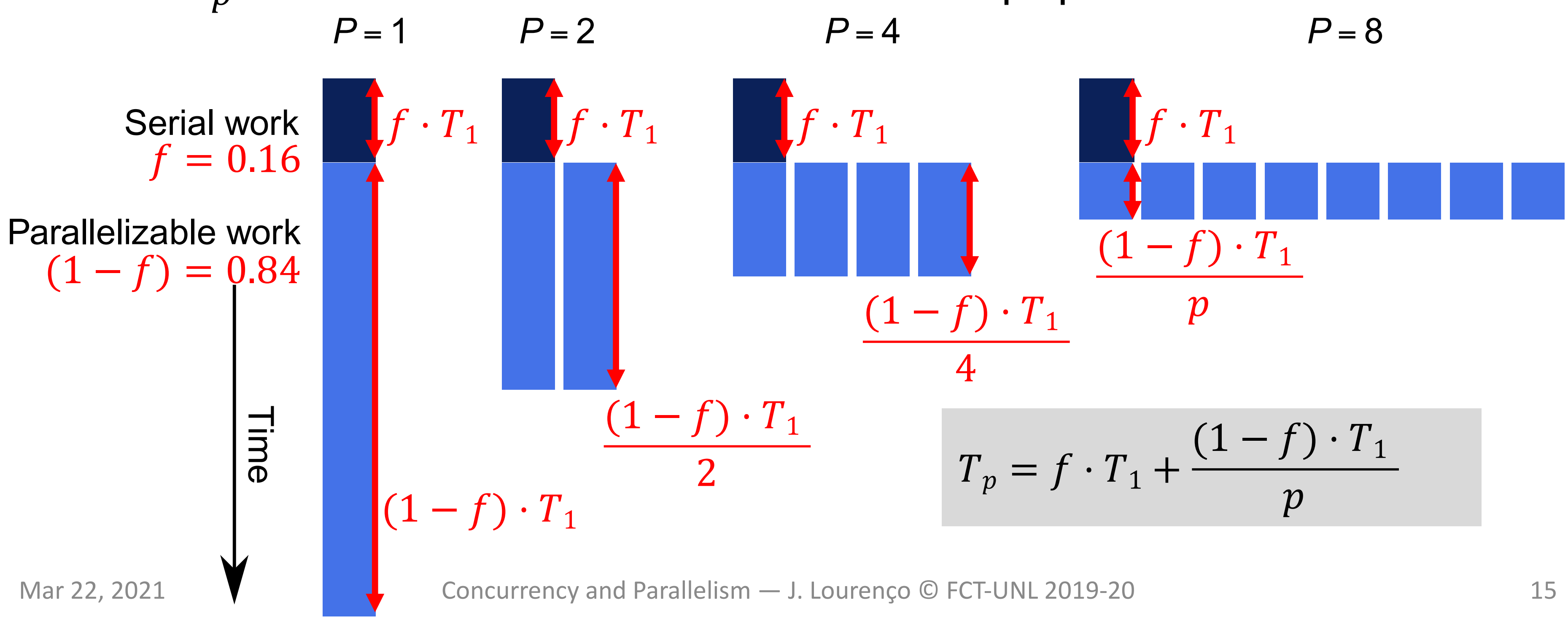
Amdahl's Law (Fixed Size Speedup)

- Interested in solving the problem faster
- *Reduce execution time*



Amdahl's Law (Fixed Size Speedup)

- Let f be the fraction of a program that is sequential
  - $(1-f)$ is the fraction that can be parallelized
- Let T_1 be the execution time on 1 processor
- Let T_p be the execution time on p processors



$$T_p = f \cdot T_1 + \frac{(1-f) \cdot T_1}{p}$$

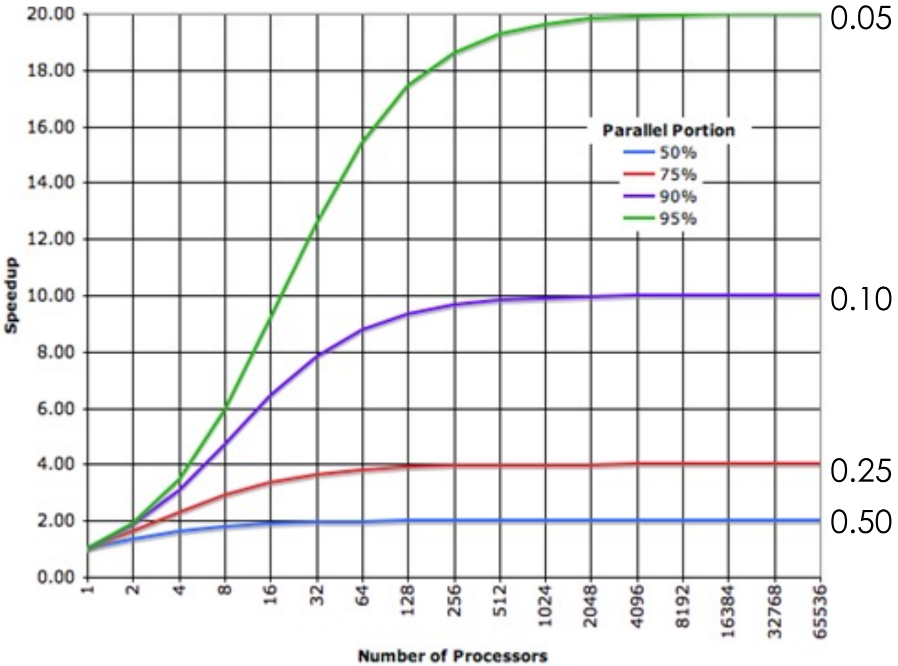
Amdahl's Law (Fixed Size Speedup)

- Let f be the fraction of a program that is sequential
  - $1 - f$ is the fraction that can be parallelized
- Let T_1 be the execution time on 1 processor
- Let T_p be the execution time on p processors
- S_p is the *speedup*

$$S_p \leq \frac{T_1}{T_p} = \frac{T_1}{f \cdot T_1 + \frac{(1-f) \cdot T_1}{p}}$$

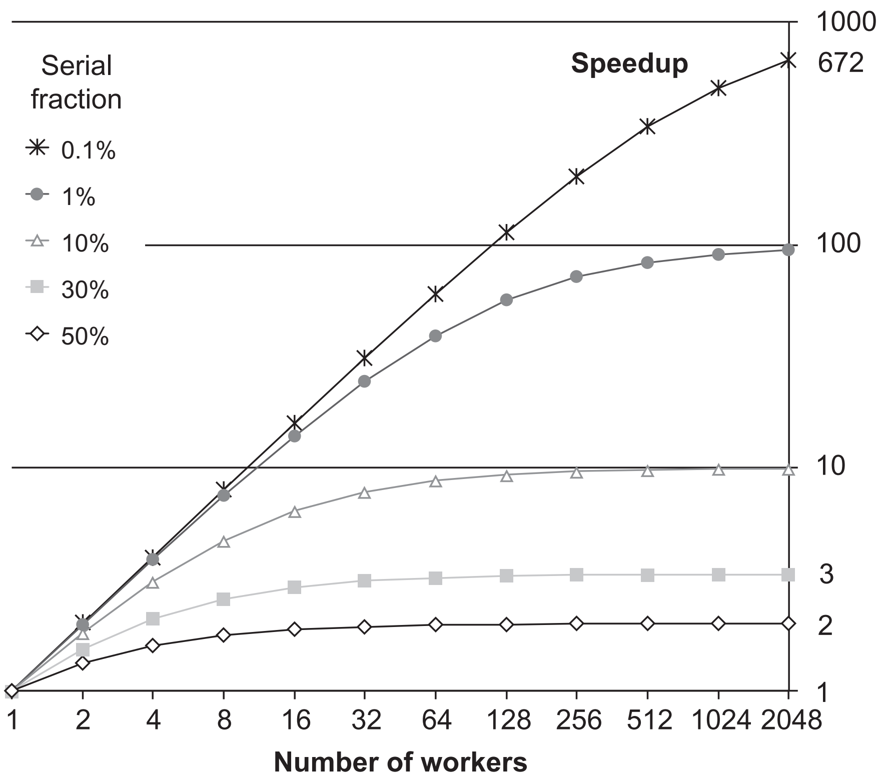
$$S_p \leq \frac{1}{f + \frac{(1-f)}{p}}$$

$$S_{p \rightsquigarrow \infty} \leq \frac{1}{f}$$



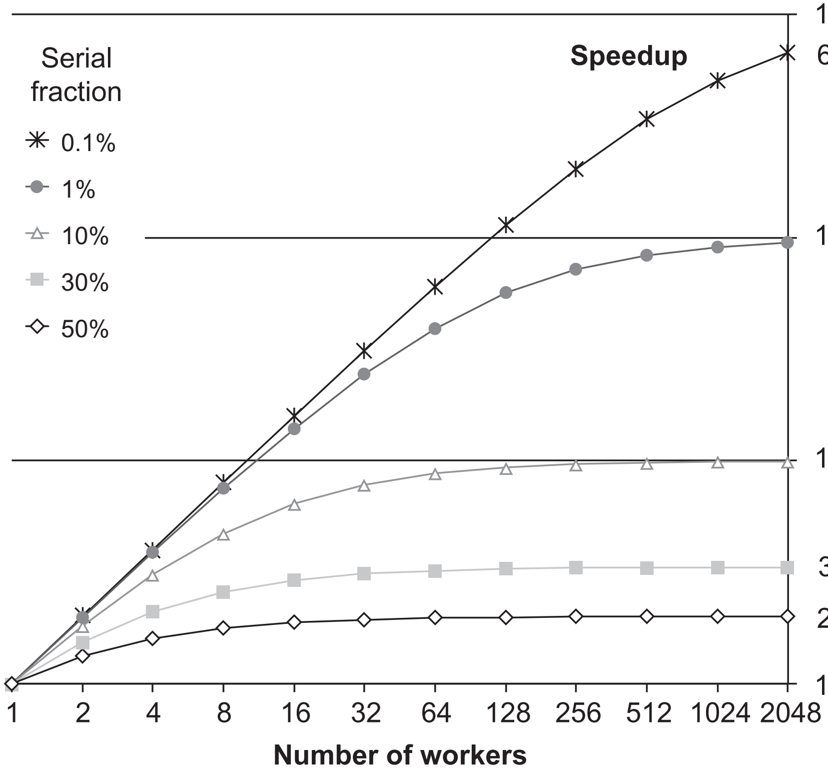
Amdahl's Law (Fixed Size Speedup)

- Amdahl's Law:
  Maximal Speedup

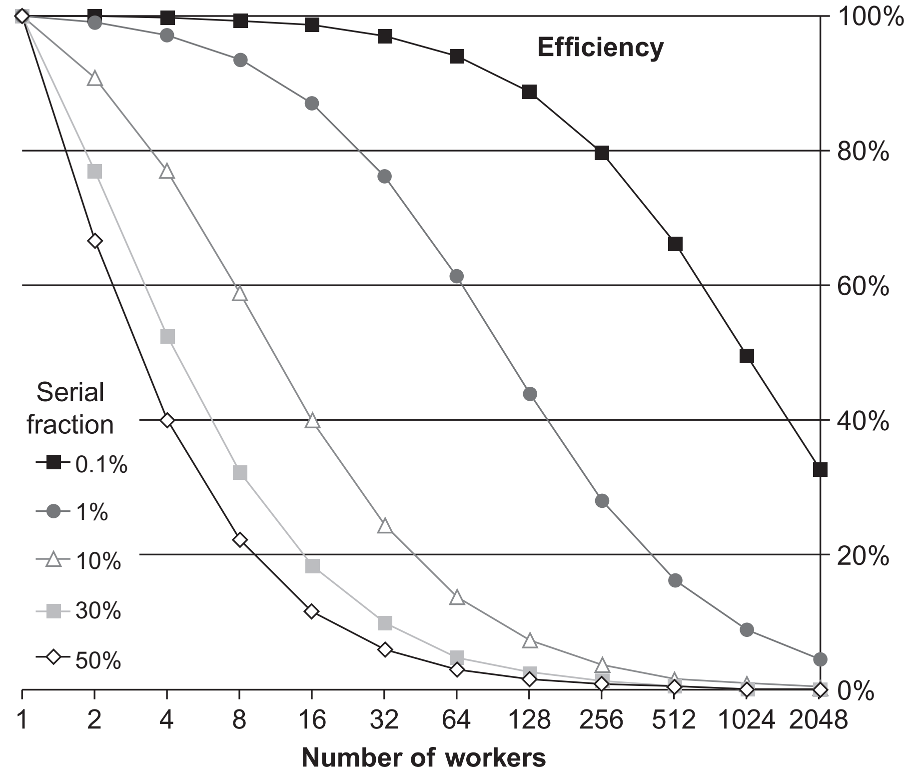


Amdahl's Law (Fixed Size Speedup)

- Amdhal's Law: Maximal Speedup



- Amdahl's Law: Efficiency → S_p/p



Amdahl's Law (Example)

- If 90% of the computation can be parallelized, what is the max. speedup achievable using 8 processors?
- Solution:

$$f = 10\% = 0.1$$

$$S(8) \leq \frac{1}{0.1 + \frac{1-0.1}{8}} \approx 4.7$$

Amdahl's Law and Scalability

- Scalability
  - Ability of parallel algorithm to achieve performance gains proportional to the number of processors and the size of the problem
- When does Amdahl's Law apply?
  - When the problem size is fixed
  - *Strong scaling* ($p \rightarrow \infty$, $S_p = S_\infty \rightarrow 1 / f$)
  - **Speedup bound is determined by the degree of sequential execution time in the computation, not # processors!!!**
  - Uhh, this is not good … Why?
  - Perfect efficiency is hard to achieve
- See original paper by Amdahl at
  - http://inst.eecs.berkeley.edu/~n252/sp07/Papers/Amdahl.pdf

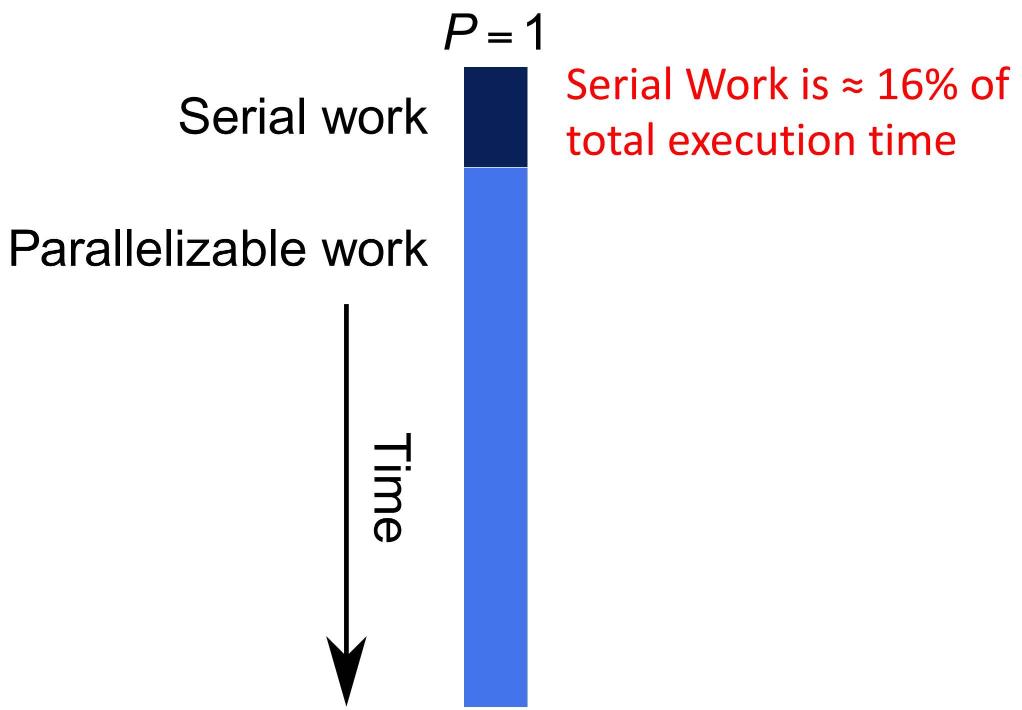
Gustafson-Barsis' Law (Scaled Speedup)

…speedup should be measured by scaling the problem to the number of processors, not by fixing the problem size.

— John Gustafson

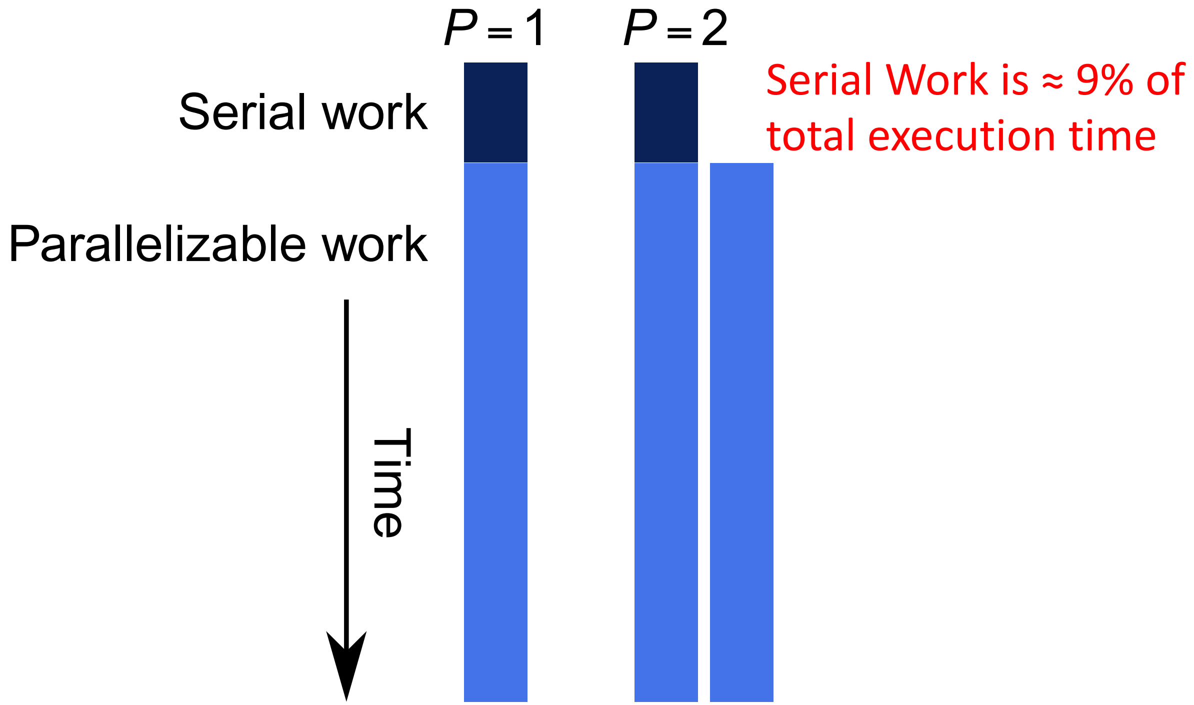
Gustafson-Barsis' Law (Scaled Speedup)

- Often interested in larger problems when scaling
  - How big of a problem can be run (HPC Linpack)
  - Constrain problem size by parallel time



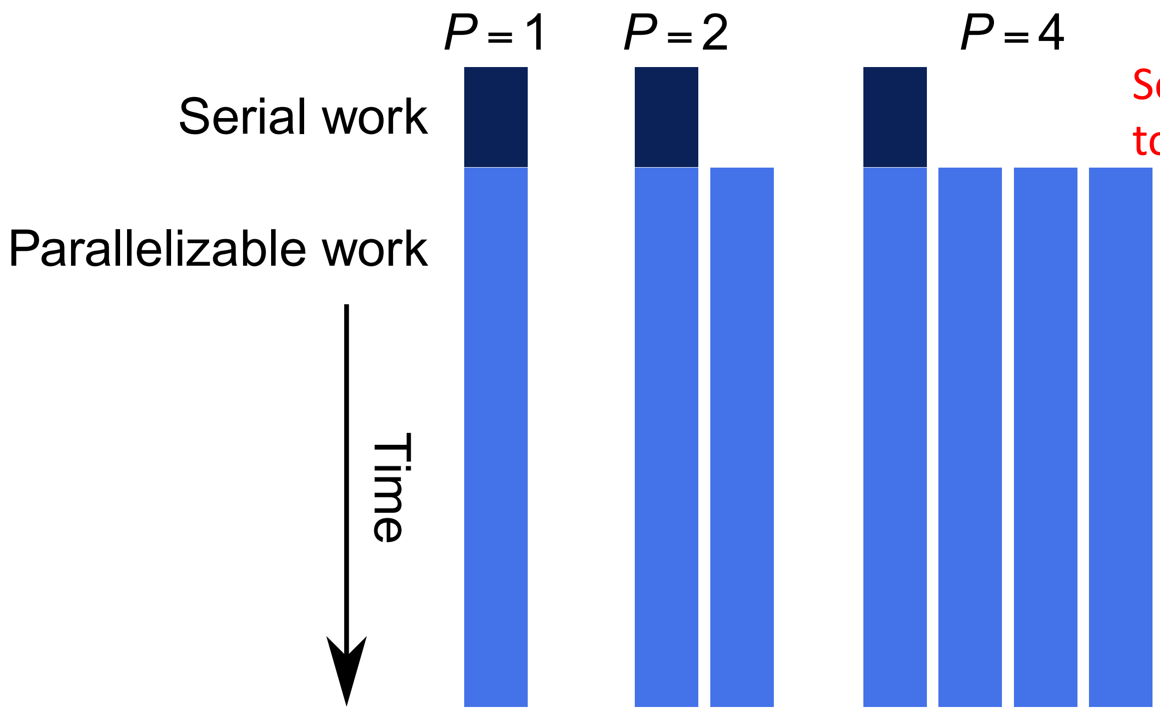
Gustafson-Barsis' Law (Scaled Speedup)

- Often interested in larger problems when scaling
  - How big of a problem can be run (HPC Linpack)
  - Constrain problem size by parallel time



Gustafson-Barsis' Law (Scaled Speedup)

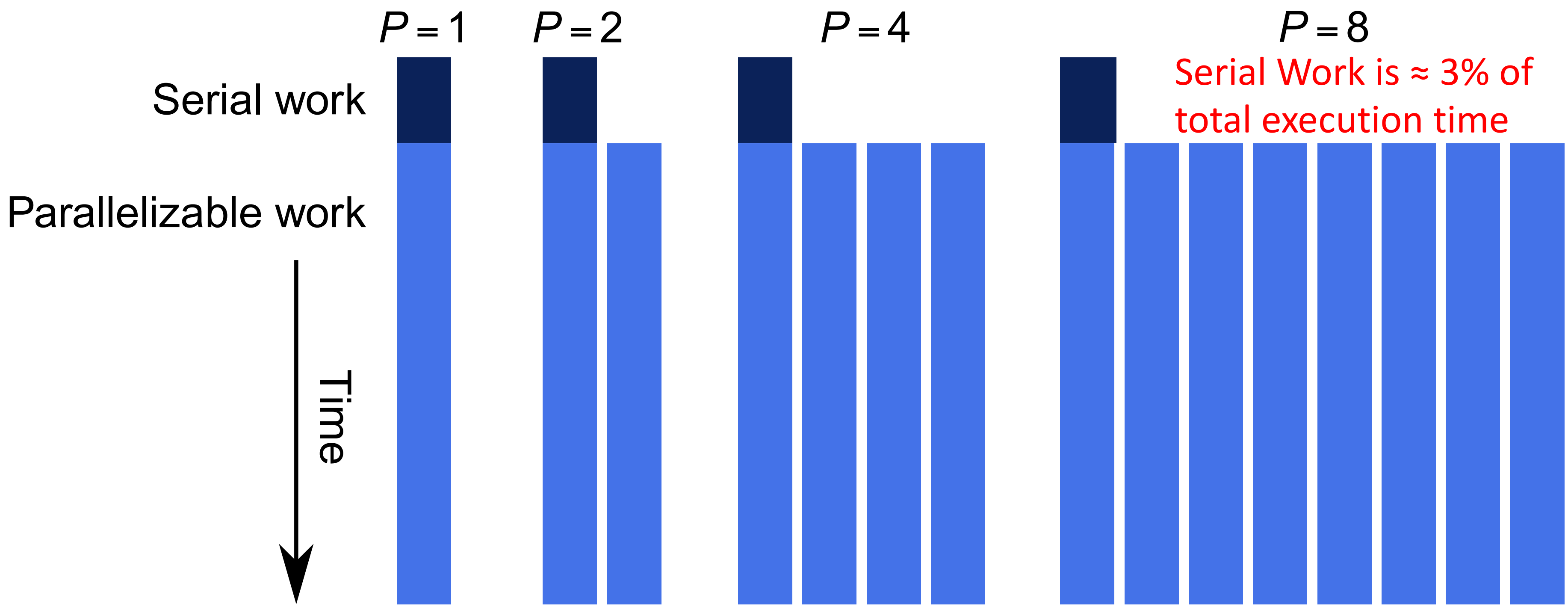
- Often interested in larger problems when scaling
  - How big of a problem can be run (HPC Linpack)
  - Constrain problem size by parallel time



Serial Work is ≈ 5% of total execution time

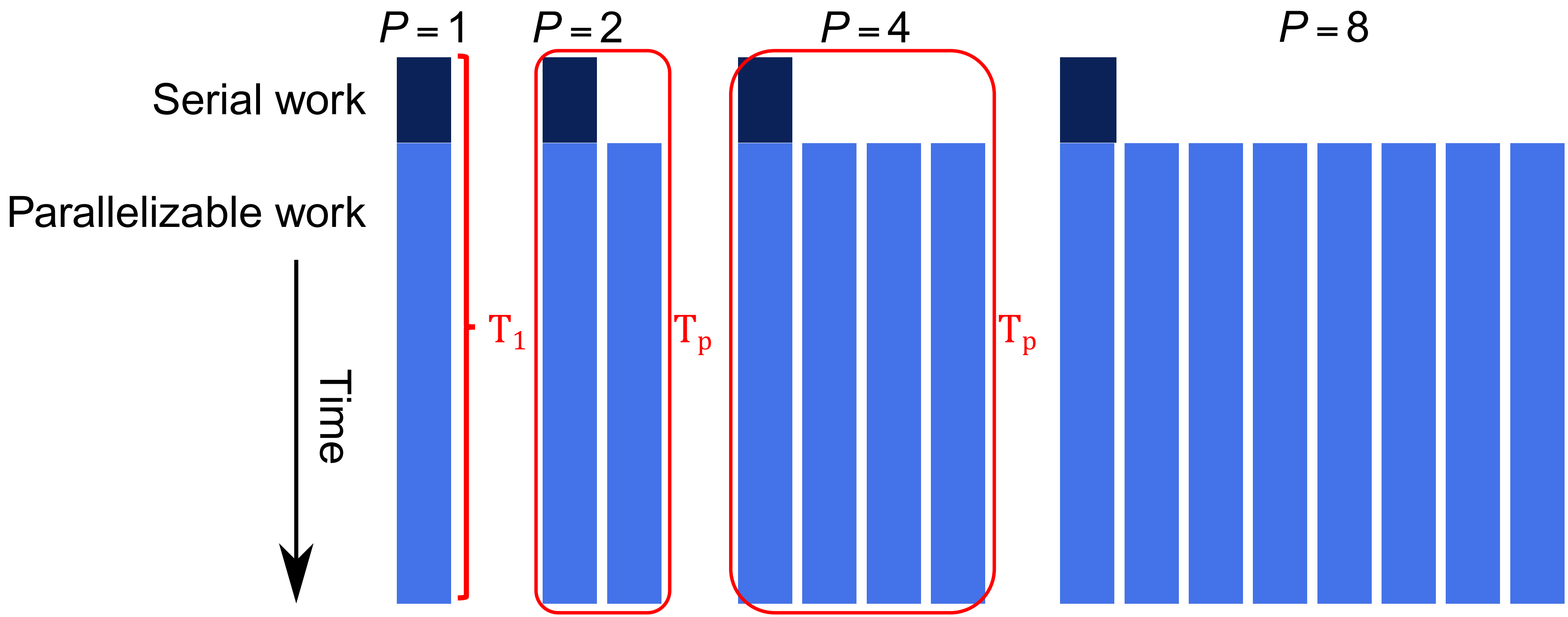
Gustafson-Barsis' Law (Scaled Speedup)

- Often interested in larger problems when scaling
  - How big of a problem can be run (HPC Linpack)
  - Constrain problem size by parallel time



Gustafson-Barsis' Law (Scaled Speedup)

- Often interested in larger problems when scaling
  - How big of a problem can be run (HPC Linpack)
  - Constrain problem size by parallel time



Gustafson-Barsis' Law (Scaled Speedup)

- Execution time of a parallel program: $T_1 = a + b$
  - a => part not parallelizable
  - b => part parallelizable
- Because we are scaling the problem (data being processed), with "P" processors we have:
$$T_P = a + P \cdot b$$
- The wall clock execution time is always the same, so scaled speedup is calculated on the volume of data processed (which is proportional to the total/accumulated execution time):
$$S_p \leq T_p \, / \, T_1 = (a + P \cdot b) \, / \, (a + b)$$

Gustafson-Barsis' Law (Scaled Speedup)

- Scaled speedup $S_p \leq T_p \,/\, T_1 = (a + P \cdot b) \,/\, (a + b)$
- Let $\alpha = a \,/\, (a + b)$ be the sequential fraction of the parallel execution time
- Then the scaled speedup is

$$S_p \leq \alpha + P \cdot (1 - \alpha) = P - \alpha \cdot (P - 1)$$

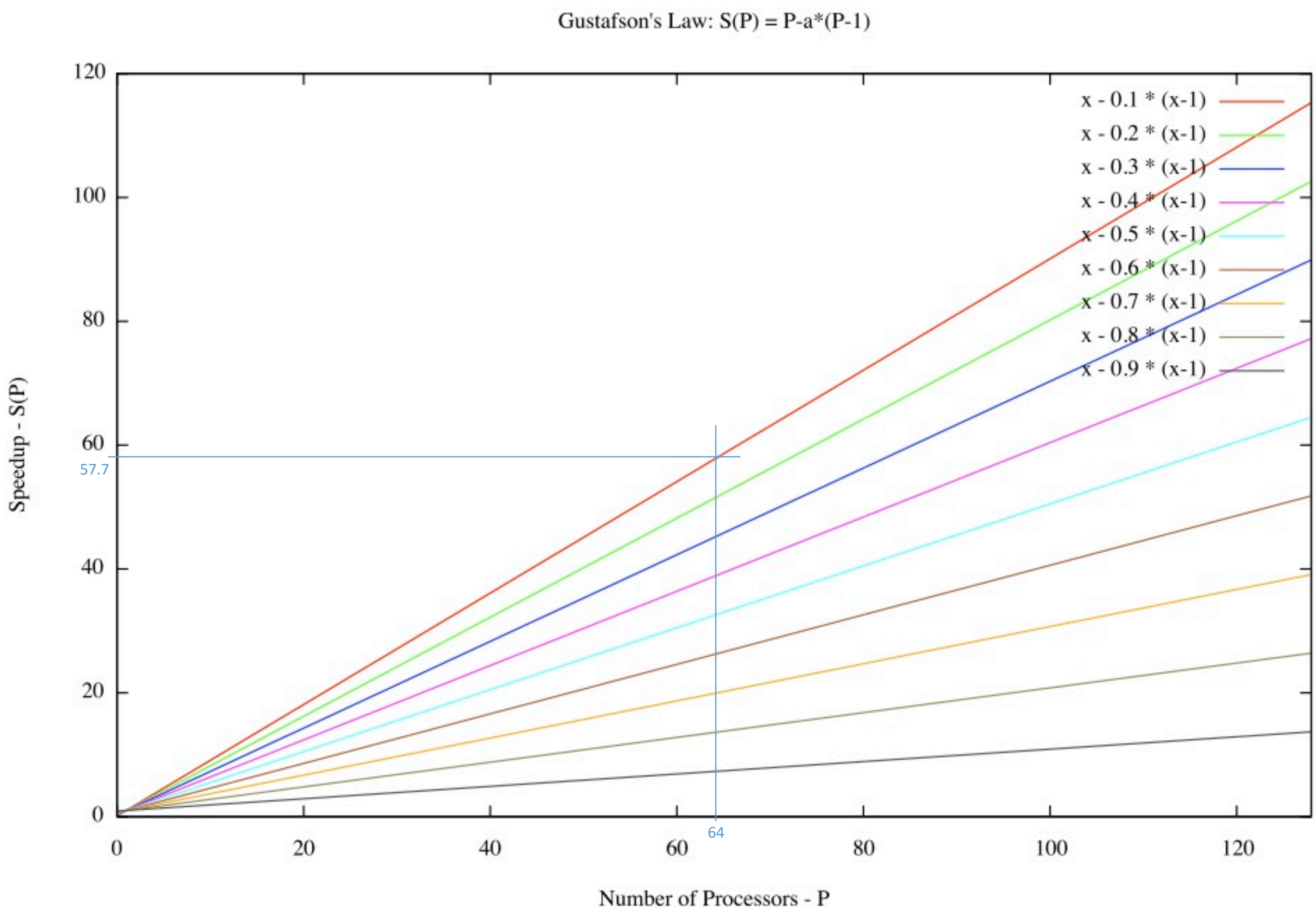
- If $\alpha \rightarrow 0$ then $S_p \rightarrow P$

Gustafson-Barsis' Law (Example)

- An application executing on 64 processors spends 5% of the total time on non-parallelizable computations. What is the scaled speedup?
- Solution:

$$
\begin{aligned}
S(64) &\leq P - \alpha \cdot (P - 1) \\
&\leq 64 - 0.05\,(64 - 1) \\
&\leq 60.85
\end{aligned}
$$

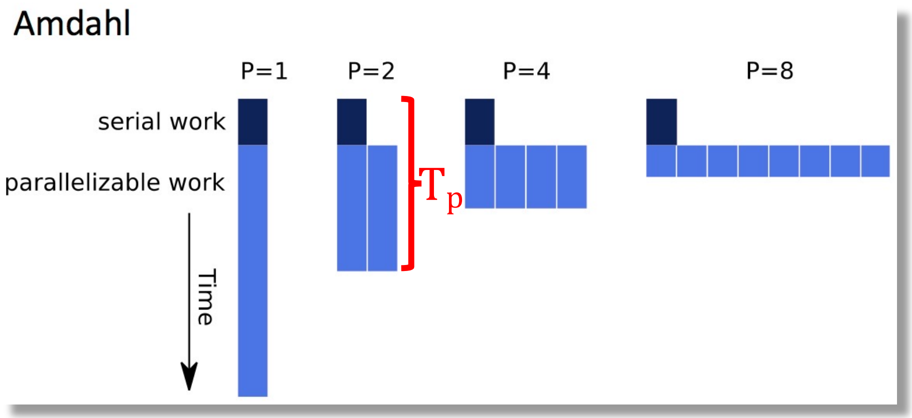
Gustafson-Barsis' Law

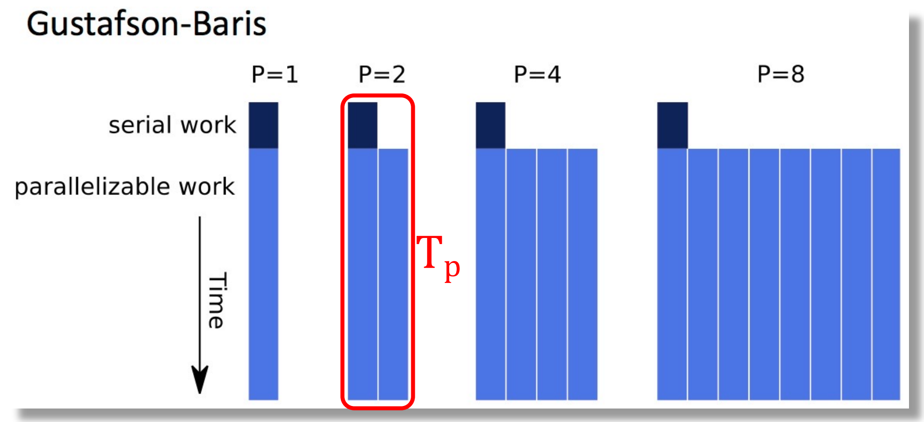


Gustafson-Barsis' Law and Scalability

- Scalability
  - Ability of parallel algorithm to achieve performance gains proportional to the number of processors and the size of the problem
- When does Gustafson's Law apply?
  - When the problem size can increase when the number of processors increases
  - Speedup function includes the number of processors!!!
  - Can maintain or increase parallel efficiency as the problem scales

Amdahl versus Gustafson-Baris





- Time: wall clock time
- Sequential part tends to dominate computation
- Upper-bound on scalability

- Time: CPU time
- Sequential part tends to become irrelevant
- No upper-bound on scalability

The END