
Concurrent Systems (ComS 527)

Ali Jannesari

Department of Computer Science

Iowa State University, Spring 2023

PARALLEL PERFORMANCE

Outline

- Performance metrics
- Amdahl's law
- Law of Gustavson
- Asymptotic complexity
- Little's formula

Primary performance metrics

- Response time, execution time
 - Time between start and completion of an event or program
- Throughput
 - Total amount of work done in a given time
- Energy (to solution)

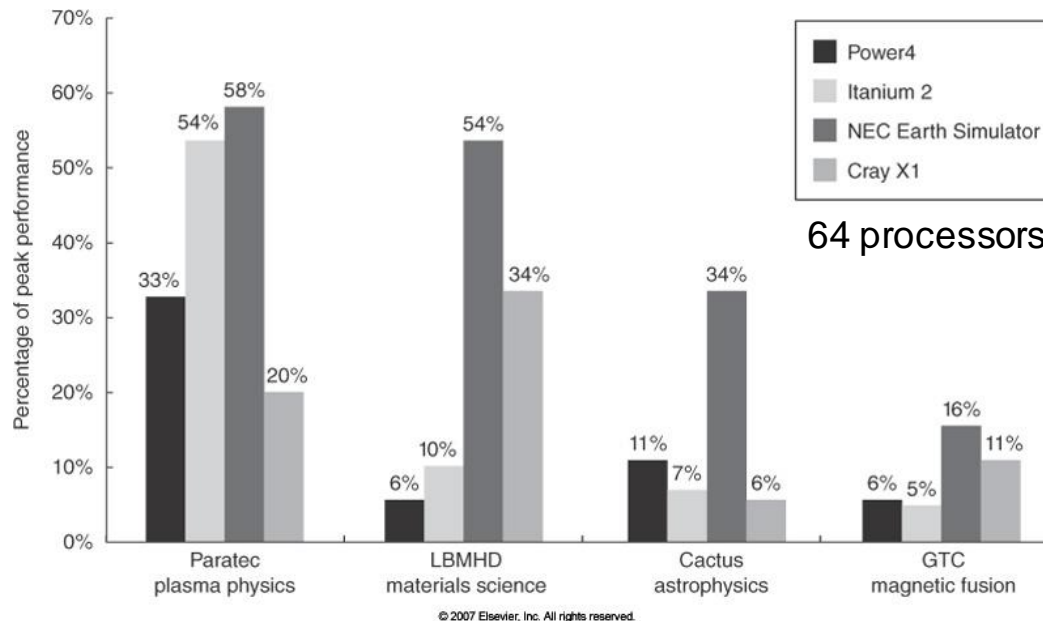


- Performance =
$$\frac{1}{\text{Resources to solution}}$$



Peak performance

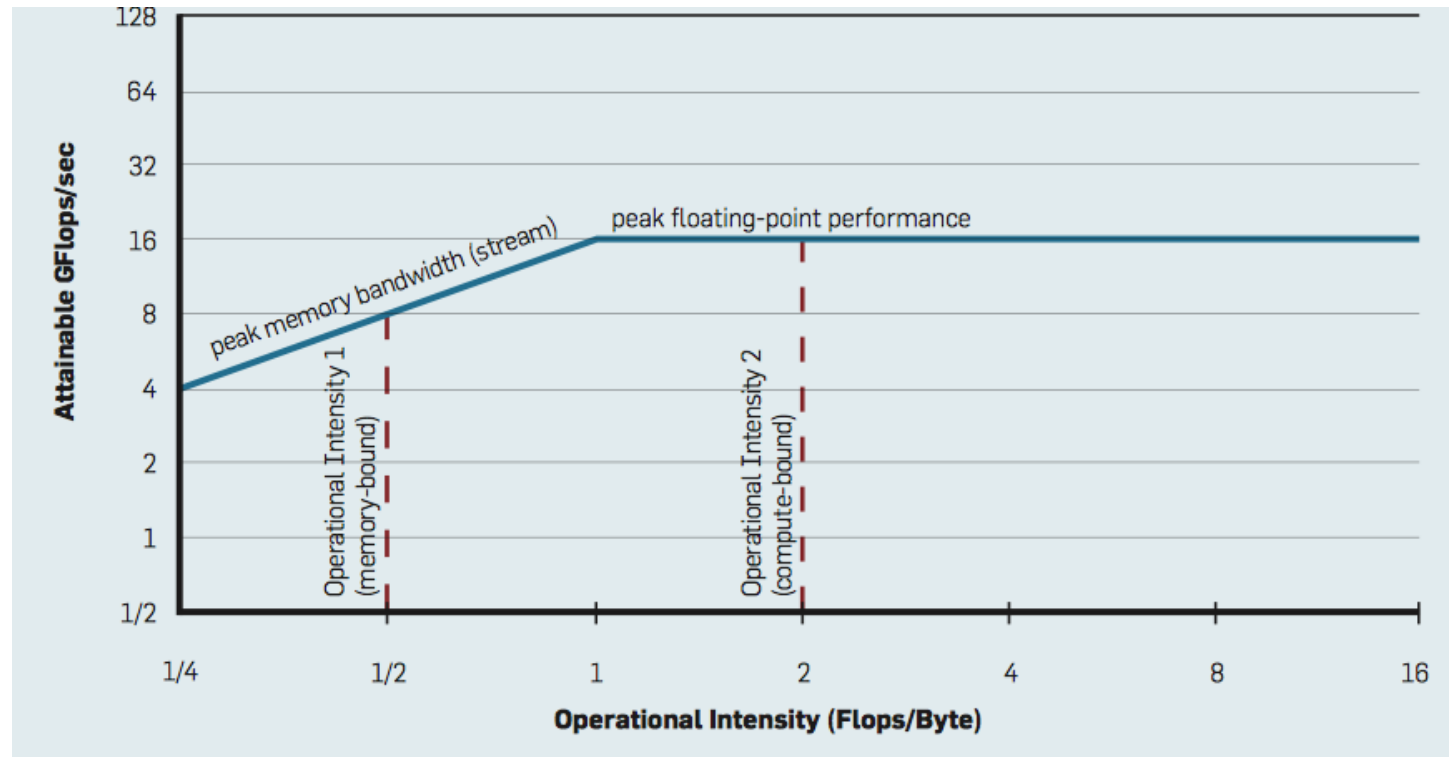
- Peak performance is the performance a computer is guaranteed not to exceed



Source: Hennessy, Patterson: Computer Architecture, 4th edition, Morgan Kaufmann

Actual performance on a computer limited by critical resource

Opteron X2



Source: S. Williams, A. Waterman, D. Patterson: Roofline: An Insightful Visual Performance Model for Multicore Architectures, Communications of the ACM, 52(4), 2009.

Speedup

- Improvement in execution time to be gained from using some faster mode of execution (e.g., parallel execution)

$$\text{Speedup: } S = \frac{\text{Execution time for the entire task without using the enhancement}}{\text{Execution time for the entire task using the enhancement when possible}}$$

- Depends on two factors
 - Fraction f of the original execution that can benefit from the enhancement
 - Improvement gained through enhancement during that fraction

Amdahl's law

$$S = \frac{1}{(1 - f_{enhanced}) + \frac{f_{enhanced}}{S_{enhanced}}}$$

$$S < \frac{1}{(1 - f_{enhanced})}$$

Example

- Function foo() of a program takes 20 % of the overall time
- How is the speedup if the time needed for foo() can be halved?

$$S = \frac{1}{(1 - 0.2) + \frac{0.2}{2}} = \frac{10}{9}$$

- How is the speedup if the time needed for foo() can almost be eliminated?

$$S = \frac{1}{(1 - 0.2) + \frac{0.2}{\infty}} = \frac{10}{8}$$

Amdahl's law for parallelism

- Assumption – program can be parallelized on p processors except for a sequential fraction f_{seq} with $0 \leq f_{seq} \leq 1$

$$S = \frac{1}{(1 - f_{enhanced}) + \frac{f_{enhanced}}{S_{enhanced}}}$$

$$S = \frac{1}{(1 - (1 - f_{seq})) + \frac{1 - f_{seq}}{p}}$$

$$S < \frac{1}{f_{seq}}$$

Speedup limited by sequential fraction

Available parallelism

Amdahl's law

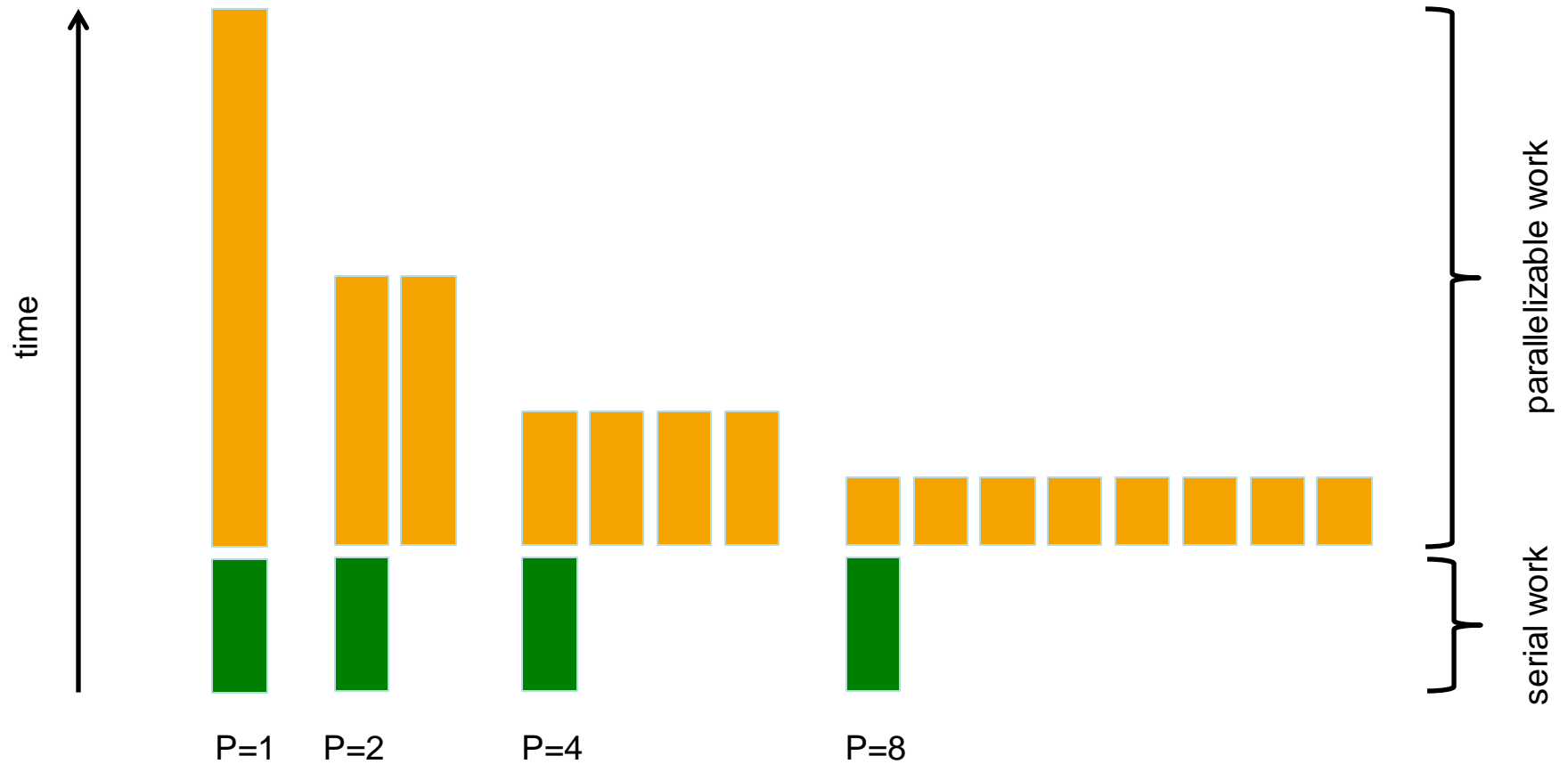
$$S = \frac{1}{f_{seq} + \frac{1 - f_{seq}}{p}}$$

Overall speedup of 80 on 100 processors

$$80 = \frac{1}{f_{seq} + \frac{1 - f_{seq}}{100}}$$

$$\Rightarrow f_{seq} = 0.0025$$

Amdahl's law visualized



Parallel efficiency

$$E(p) = \frac{S(p)}{p}$$

- Metric for cost of parallelization (e.g., communication)
- Without super-linear speedup

$$E(p) \leq 1$$

Super-linear speedup possible

- Critical data structures may fit into the aggregate cache

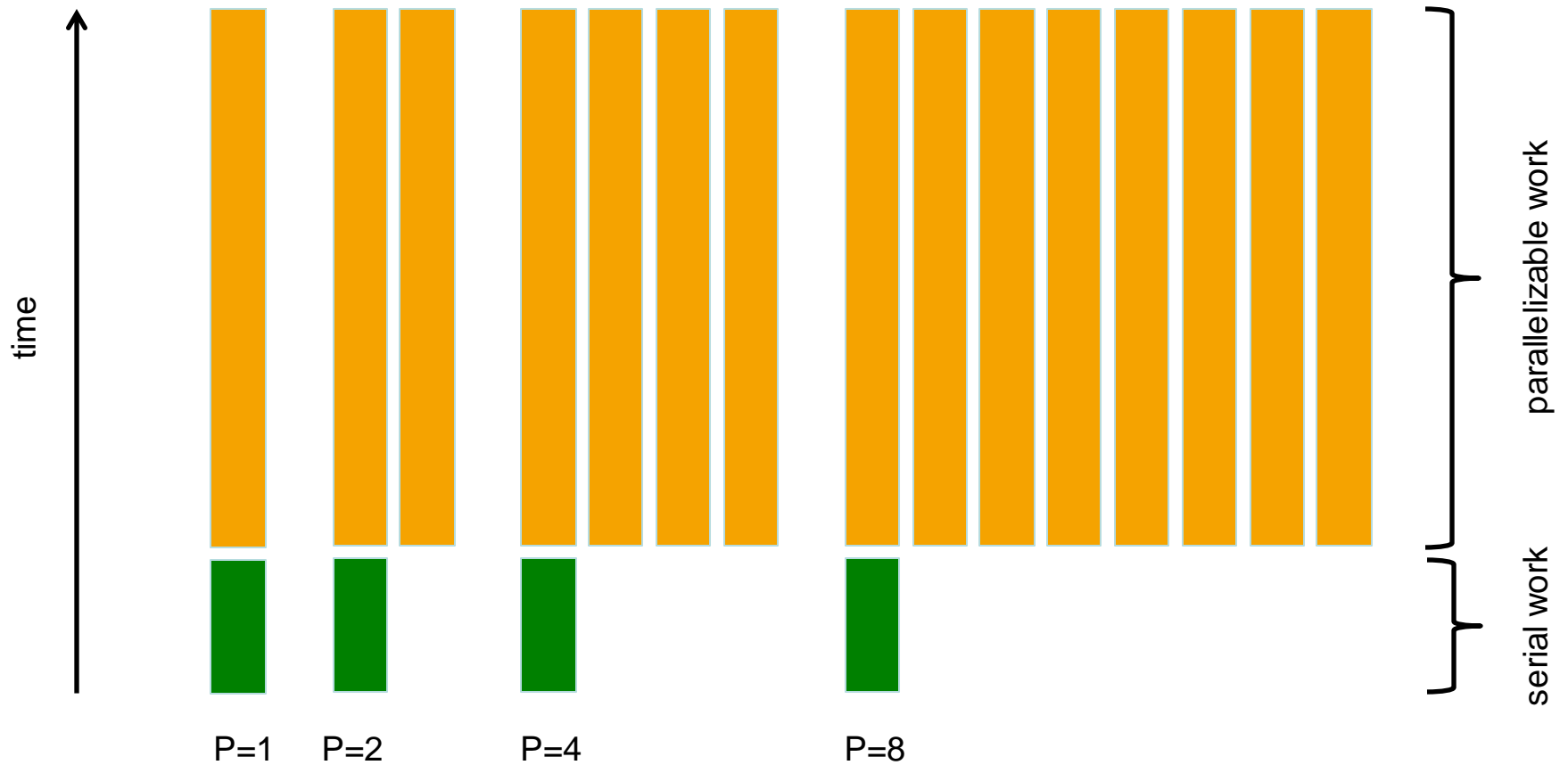
Law of Gustafson

- Amdahl's Law ignores increasing problem size
 - Parallelism often applied to calculate bigger problems instead of calculating a given problem faster
- Fraction of sequential part may be function of problem size
- Assumption
 - Sequential part has constant runtime τ_f
 - Parallel part has runtime $\tau_v(n, p)$
- Speedup

$$S(n, p) = \frac{t_f + t_v(n, 1)}{t_f + t_v(n, p)} \xrightarrow{n \rightarrow \infty} p$$

If parallel part can be perfectly parallelized

Law of Gustafson (2)



Scalability

Weak scaling

- Ability to solve a larger input problem by using more resources (here: processors)
- Problem size per processor remains constant
- Example: larger domain, more particles, higher resolution

Strong scaling

- Ability to solve the same input problem faster as more resources are used
- Usually more challenging
- Limited by Amdahl's law and communication demand

- Example: dot product of two vectors of length $N \geq P$

- $$\begin{array}{ccccccc} \mathfrak{A} & a_1 & \ddot{\emptyset} & \mathfrak{A} & b_1 & \ddot{\emptyset} \\ \zeta & & \div & \zeta & & \div \\ \zeta & \square & \div & * & \zeta & \square & \div \\ \mathfrak{C} & & \dot{\div} & \mathfrak{C} & & \dot{\div} \\ \mathfrak{C} & a_n & \dot{\div} & \mathfrak{C} & b_n & \dot{\div} \\ \mathfrak{E} & & \emptyset & \mathfrak{E} & & \emptyset \end{array}$$

16

Asymptotic complexity notation

- Big **O** notation – denotes a set of functions with an upper bound.
 $O(f(N))$ = set of all functions $g(N)$ such that there exists positive constants c, N_0 with $|g(N)| \leq c * |f(N)|$ for $N \geq N_0$.
- Big **Omega** notation – denotes a set of functions with a lower bound.
 $\Omega(f(N))$ = set of all functions $g(N)$ such that there exists positive constants c, N_0 with $|g(N)| \geq c * |f(N)|$ for $N \geq N_0$.
- Big **Teta** notation – denotes a set of functions with both lower and upper bounds. $\theta(f(N))$ = set of all functions $g(N)$ such that there exists positive constants c_1, c_2, N_0 with $c_1 * |f(N)| \leq |g(N)| \leq c_2 * |f(N)|$ for $N \geq N_0$.

Asymptotic speedup

- Asymptotic speedup

$$\frac{T_1}{T_P} = \frac{Q(N)}{Q(N/P + \lg P)} = O\left(\frac{N}{N/P + \lg P}\right)$$

- Asymptotic efficiency

$$\frac{T_1}{P \times T_P} = O\left(\frac{N}{N + P \lg P}\right)$$

Little's formula

Relates throughput and latency of a system to concurrency

$$C = R \times L$$

Applies to system in steady state

- Desired throughput rate is R items per unit time
- Latency to process each item is L
- Number of items concurrently in the system is C

Determines concurrency required to hide latency

Summary

- Amdahl's law applies to strong scaling
- Law of Gustavson takes weak scaling into account
- Asymptotic complexity is an effective instrument to reason about an algorithm's scalability
- Little's formula tells how to hide latency