TECHNISCHE UNIVERSITÄT DARMSTADT

Parallel Programming

Parallel Programming
- Parallel Performance -

Prof. Dr. Felix Wolf
Technical University of Darmstadt

Outline

TECHNISCHE UNIVERSITÄT DARMSTADT

Parallel Programming

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

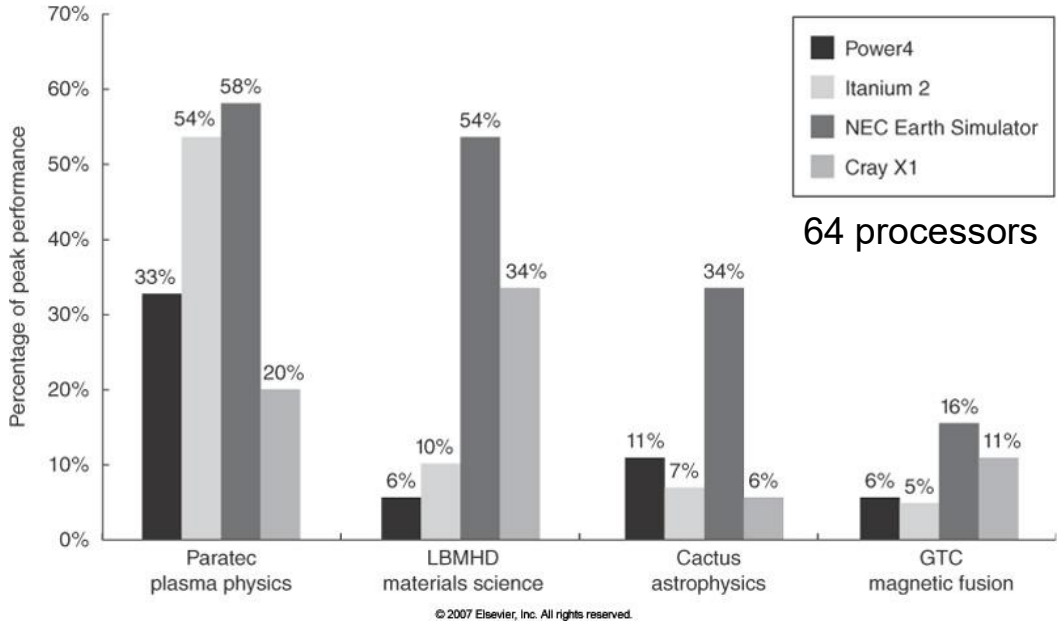
Primary performance metrics

- 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

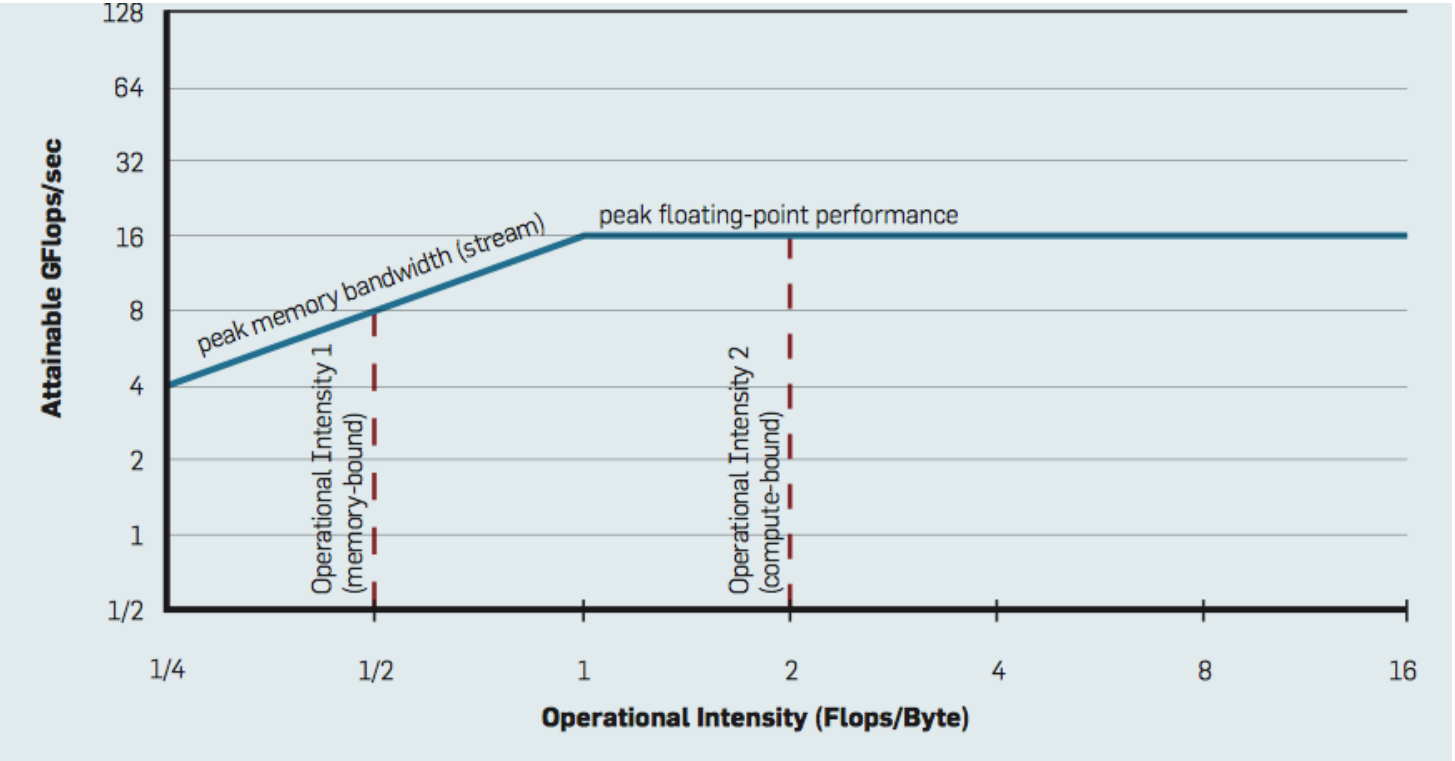
- Peak performance is the performance a computer is guaranteed not to exceed



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

Actual performance 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)

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

- 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

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

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

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 $0 \leq f_{seq} \leq 1$

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

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

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

Speedup limited by sequential fraction

Available parallelism

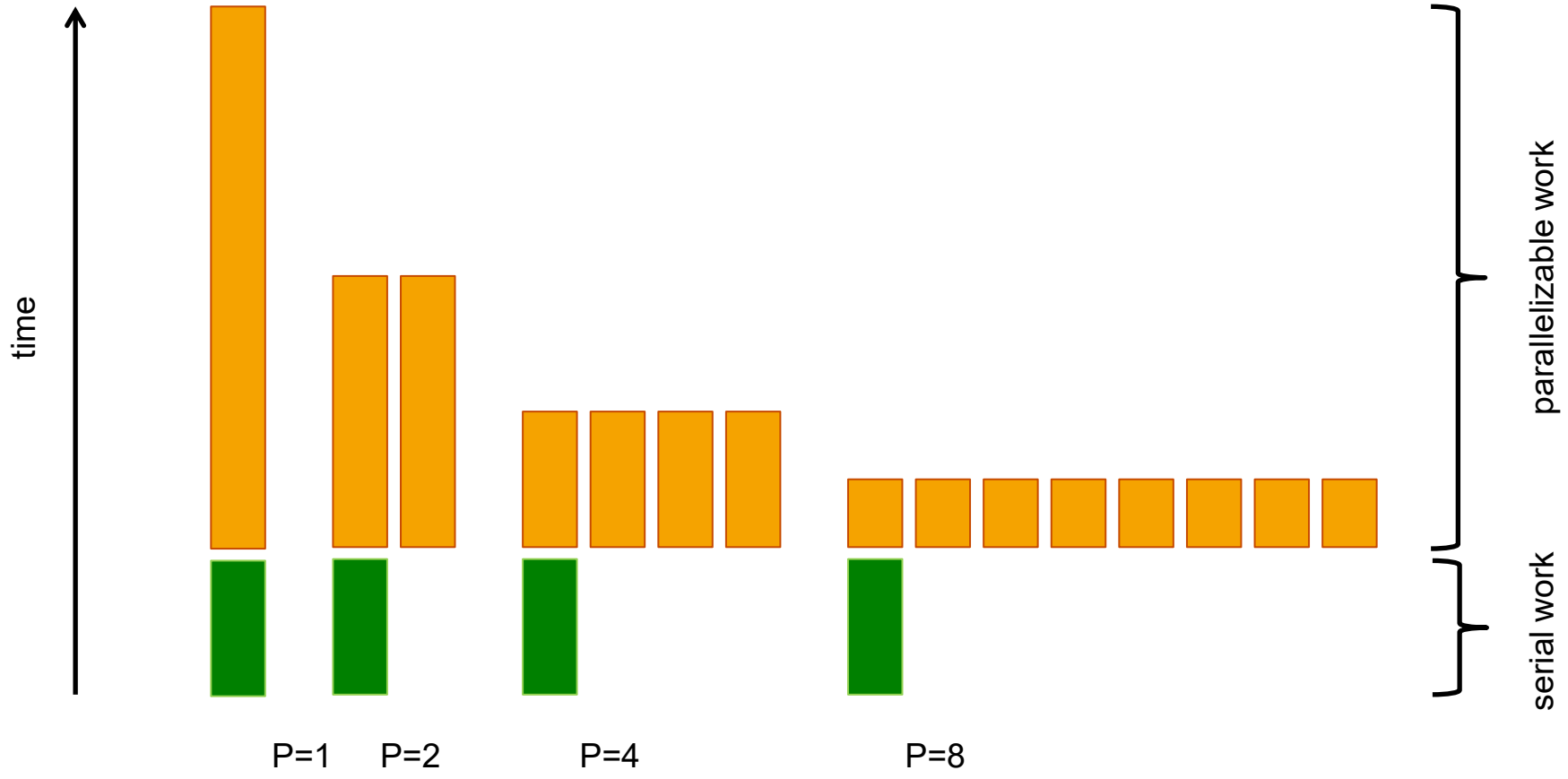
Amdahl's law

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

Overall speedup of 80 on 100 processors

$$80 = \frac{1}{f_{\text{seq}} + \frac{1 - f_{\text{seq}}}{100}} \qquad \Rightarrow f_{\text{seq}} = 0.0025$$

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

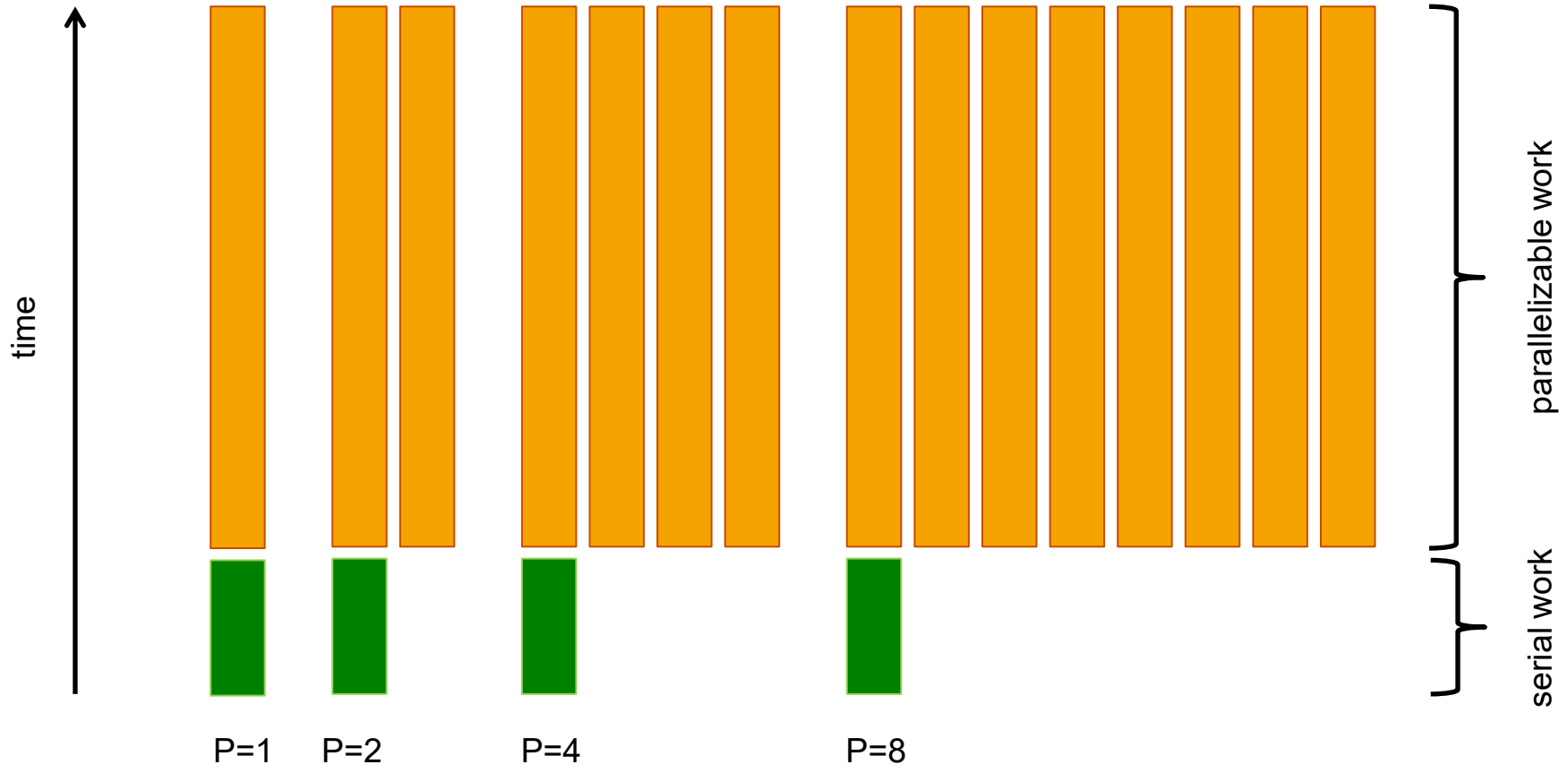
Law of Gustafson

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

$$S(n,p) = \frac{\tau_f + \tau_v(n,1)}{\tau_f + \tau_v(n,p)} \xrightarrow[n\to\infty]{} p$$

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

Asymptotic complexity notation

Set of functions with lower / upper bound

- Defined as set of all functions *g(N)* such that there exist positive constants … with …

Name	Symbol	Bound	Constants	Formula
Big **O**	O(f(N))	Upper	c, N_0	$\|g(N)\| \leq c^* \|f(N)\|$ for $N \geq N_0$
Big **Omega**	Ω(f(N))	Lower	c, N_0	$\|g(N)\| \geq c^* \|f(N)\|$ for $N \geq N_0$
Big **Teta**	θ(f(N))	Both	c_1, c_2, N_0	$c_1^* \|f(N)\| \leq \|g(N)\| \leq c_2^* \|f(N)\|$ for $N \geq N_0$

Asymptotic complexity

- Example: dot product of two vectors of length $N \geq P$
  - Split vector into pieces of length N/P
  - Calculate subproducts in parallel
  - Calculate global sum in tree-like fashion

$$\begin{pmatrix} a_1 \\ \vdots \\ a_n \end{pmatrix} * \begin{pmatrix} b_1 \\ \vdots \\ b_n \end{pmatrix}$$

- Asymptotic complexity

$$T(N,P) = \Theta(N/P + \lg P)$$

Asymptotic speedup

- Asymptotic speedup

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

- Asymptotic efficiency

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

Little's formula

Relates throughput and latency of a system to concurrency

$$C = R \cdot L$$

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