Language Abstractions for Concurrent and Parallel Programming

Introduction

Dave Clarke
dave.clarke@it.uu.se

Welcome!

Dave Clarke
Teacher

George Kalamatianos
Lab Assistant

Today

- Practical matters
- General information
- Introduction to concurrent and parallel computing
- Project details

This Course Is Fairly New

- Given for the first time in 2014
- Part of the UPMASTER Specialization in Concurrent and Parallel Programming (see `http://www.it.uu.se/upmaster`)
- Split in two in 2017 – this is the second half of prior instances.

Course Registration

To get credit for the course, you must be admitted and registered!

If you are admitted already, please web-register yourself for this course on the Student Portal as soon as possible.

If you are admitted but cannot web-register for any reason, contact the Student Office (`it-kansli@it.uu.se`).

Late Admission

If you are not admitted yet:

- Master students should contact their programme counsellor.
- Exchange students should contact Ulrika Jaresund (`Ulrika.Jaresund@it.uu.se`), the exchange student coordinator at the IT Department.
- Students with an older registration, who want to re-register, should contact the Student Office (`it-kansli@it.uu.se`).
- Students on the reserve list will be contacted by the Student Office if they are admitted.
- Everyone else should apply at `http://www.antagning.se`, via Late application ("Sen anmälan").

Dropping Courses

We hope that you will enjoy this course!

However, if you decide to drop it, you must inform the Student Office (`it-kansli@it.uu.se`).

- If less than 3 weeks have passed since the course started, your course registration will simply be removed.
- After 3 weeks a "course intermission" will be reported to UPPDOK instead.

Teachers cannot help with admission/registration. Please contact the Student Office (IT-kansliet) if you have questions about these formalities.

Special Needs

If you have special needs (for instance, if you need more time on exams), please contact the responsible coordinator: see

`http://teknat.uu.se/student/stod-och-service/sarskilt-stod/`

Also consider informing your teachers.

Course Structure

5 ECTS credits (hp) $\approx$ 135 hours of work
(33% pace during periods 2):

- Introduction + 3 modules ($\sim$ 10 lectures, 3 labs in total)
- 3 assignments
- Final project

The course will end just before Christmas, though some assignments are due in week 2, 2018.

Web page: `https://studentportalen.uu.se/`

Language: English

The Student Portal

Lecture slides, example programs, assignments etc. will be made available on the Student Portal.

Solutions to assignments must be submitted via the Student Portal.

Of course, you can also contact your teachers by email and in person.

Your Expectations

What do you expect to learn in this course?

What would you like to learn?

Your Background

- How familiar are you with C/C++?

(A) I have heard of it.
(B) I have used it before, at least once.
(C) I use it regularly.

Your Background

- How familiar are you with C/C++?
- How familiar are you with Java?

(A) I have heard of it.
(B) I have used it before, at least once.
(C) I use it regularly.

Your Background

- How familiar are you with C/C++?
- How familiar are you with Java?
- How familiar are you with Erlang?

(A) I have heard of it.
(B) I have used it before, at least once.
(C) I use it regularly.

Your Background

- How familiar are you with C/C++?
- How familiar are you with Java?
- How familiar are you with Erlang?
- How familiar are you with Scala?

(A) I have heard of it.
(B) I have used it before, at least once.
(C) I use it regularly.

Your Background

- How familiar are you with C/C++?
- How familiar are you with Java?
- How familiar are you with Erlang?
- How familiar are you with Scala?
- How familiar are you with Python?

(A) I have heard of it.
(B) I have used it before, at least once.
(C) I use it regularly.

Your Background (cntd.)

- How familiar are you with concurrent/parallel programming?

(A) I have heard of it.
(B) I have used it before, at least once.
(C) I use it regularly.

Your Background (cntd.)

- How familiar are you with concurrent/parallel programming?
- How familiar are you with threads and locks?

(A) I have heard of it.
(B) I have used it before, at least once.
(C) I use it regularly.

Your Background (cntd.)

- How familiar are you with concurrent/parallel programming?
- How familiar are you with threads and locks?
- How familiar are you with task-based parallelism?

(A) I have heard of it.
(B) I have used it before, at least once.
(C) I use it regularly.

Your Background (cntd.)

- How familiar are you with concurrent/parallel programming?
- How familiar are you with threads and locks?
- How familiar are you with task-based parallelism?
- How familiar are you with transactional memory?

(A) I have heard of it.
(B) I have used it before, at least once.
(C) I use it regularly.

Your Background (cntd.)

- How familiar are you with concurrent/parallel programming?
- How familiar are you with threads and locks?
- How familiar are you with task-based parallelism?
- How familiar are you with transactional memory?
- How familiar are you with actors?

(A) I have heard of it.
(B) I have used it before, at least once.
(C) I use it regularly.

Your Background (cntd.)

- How familiar are you with concurrent/parallel programming?
- How familiar are you with threads and locks?
- How familiar are you with task-based parallelism?
- How familiar are you with transactional memory?
- How familiar are you with actors?
- How familiar are you with map-reduce/spark?

(A) I have heard of it.
(B) I have used it before, at least once.
(C) I use it regularly.

Course Contents

Some different models for concurrency and parallel programming:

- transactional memory,
- actors,
- map-reduce and Spark.

A number of different programming languages to illustrate these models, such as C++ (Transactional Memory), Erlang (Actors), Java/Python/Scala (MapReduce/Spark).

Closer study of each model and its implementation in some language, plus laboratory work that experiments with the models.

Learning Outcomes

To pass the course, the student should be able to

- write and modify programs using different models and language abstractions for concurrent and parallel programming,
- analyze models with respect to which specific problems they address, where they can be used and where they should be avoided,
- explain the difference between a model and its implementation in a specific language,
- explain how some languages implement hybrids or multiple models and how these can interact.

Assessment

To pass the course, you must pass all three assignments and the final project.

Each assignment is worth 10 points. The project consist of two components (code and written report) and is worth 20 points. (Total: 50 points)

To pass the course, you must achieve 50% of the points in **each** assignment and **each** project component. (Attending the lectures is optional, but recommended!)

For a final course grade of 4, you must obtain at least 35 points. For a final course grade of 5, you must obtain at least 42 points.

Cheating

Plagiarism and cheating are serious academic offenses and can lead to suspension from the university for six months.

Cheating includes:

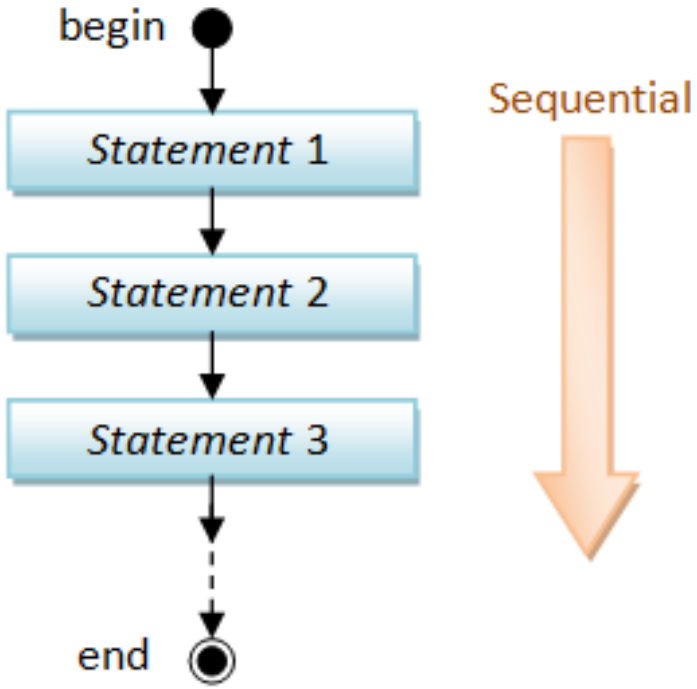
- Knowingly using some fellow student's solution to an exercise while solving it.
- Knowingly submitting a (changed) copy of some fellow student's solution.
- Knowingly submitting a solution based on a hardcopy or Internet publication without citing it.
- Knowingly helping some student to do any of the three actions above.

Golden rule: Give credit when you use someone else's ideas.

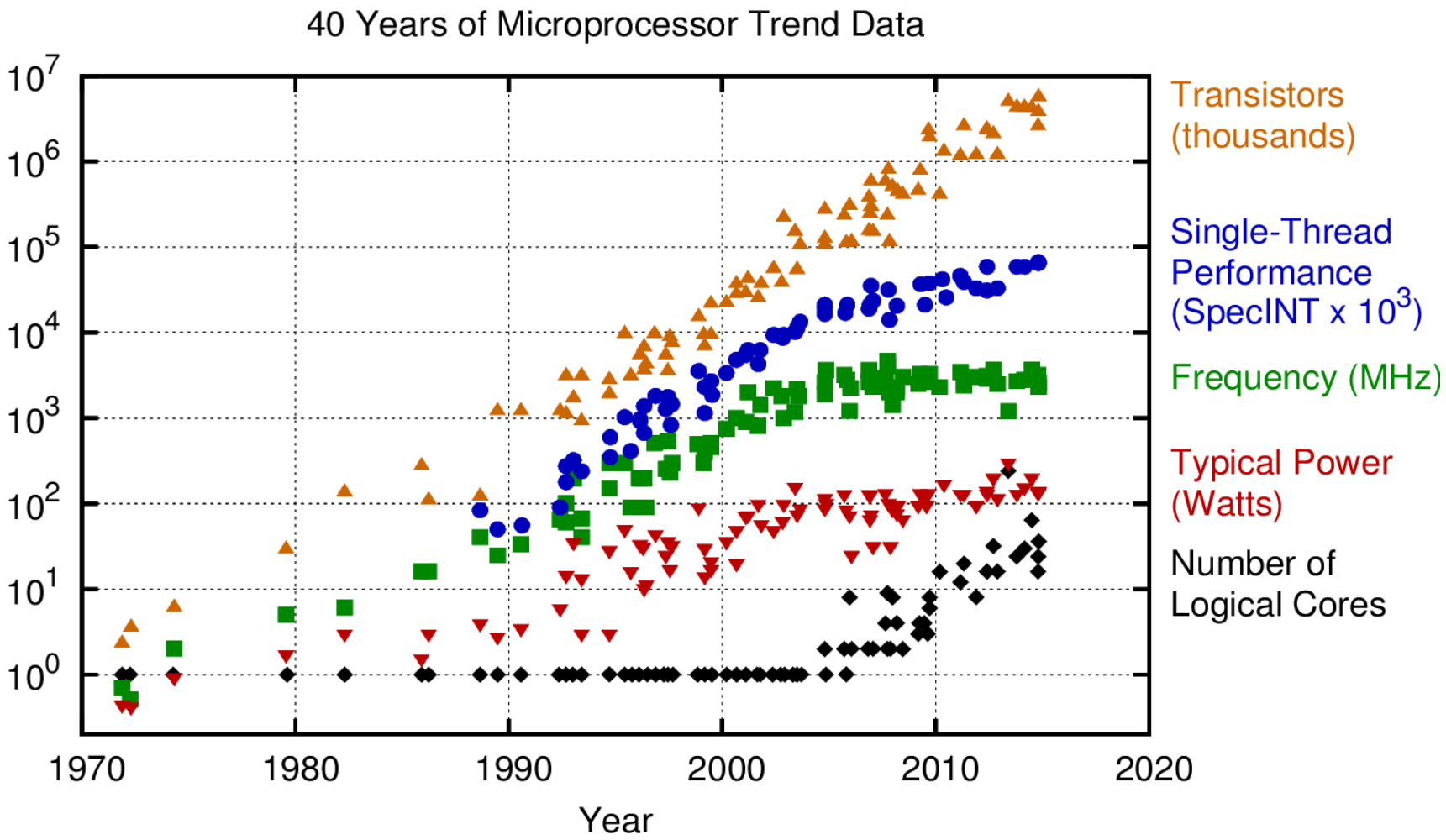
Introduction to Concurrent and Parallel Computing

Sequential Programs

A sequential program consists of a sequence (ordered list) of instructions. These must be executed in their given order, one after another.

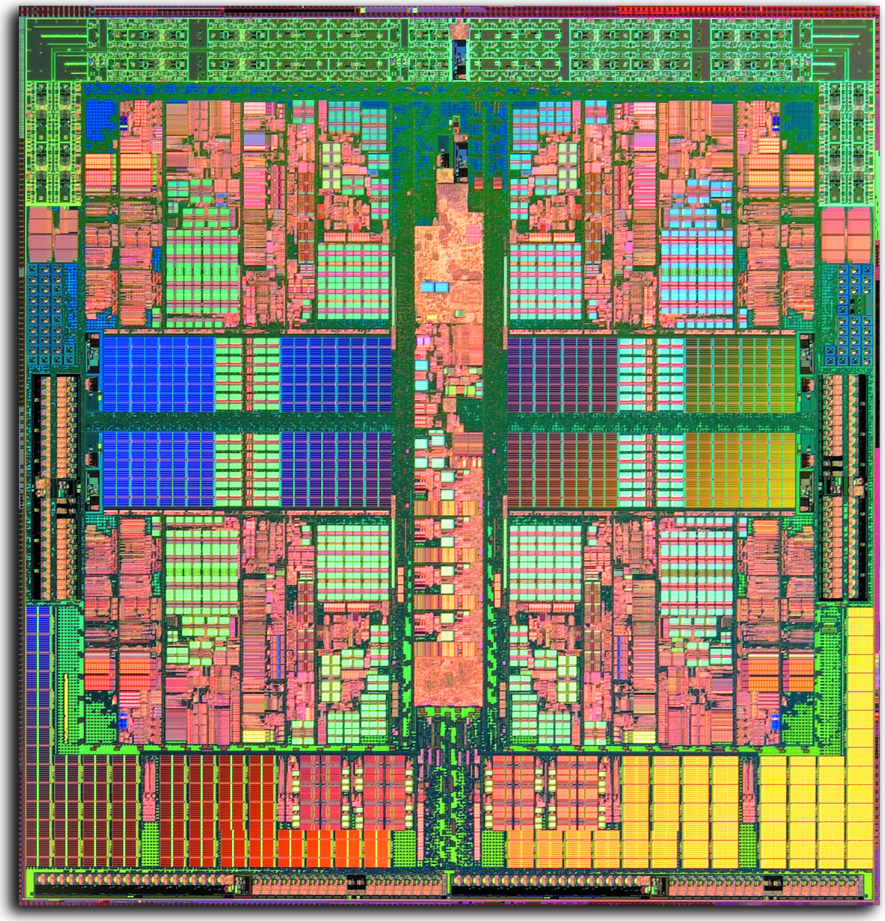


Intel CPU Trends



Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten
New plot and data collected for 2010-2015 by K. Rupp

Motivation (1)



AMD Opteron processor (4 cores)

Motivation (2)

IBM Blue Gene/P (163,840 cores)

Motivation (3)

Elastic cloud computing (??? cores)

Motivation: Summary

- Single-core performance is no longer improving dramatically. Clock speeds have remained almost constant since around 2004.
- However, the number of transistors in CPUs continues to grow exponentially. Multi-core hardware has become ubiquitous.

To take full advantage of modern hardware, software must be concurrent.

Concurrent Programming $\neq$ Parallel Programming

A concurrent program consists of independent tasks, which *may* execute during overlapping time periods.

A parallel computation executes several operations at the very same instant.

Concurrency is mostly about distribution, redundancy, etc., while parallelism is mostly about speed.

(Unfortunately, even experts are not always very careful about this distinction.)

Concurrency vs. Parallelism: Questions

Discuss:

1. Can a concurrent program be executed on a single-core CPU?
2. Are the tasks of a concurrent program always executed in parallel?
3. Can parallel computation happen on a single-core CPU?
4. Is parallel computation always concurrent?

Speedup

Speedup is a metric for relative performance improvement.

Given the old execution time T_{old} and the new execution time T_{new} for a program, the speedup is

$$S = \frac{T_{old}}{T_{new}}$$

Amdahl's Law: Motivation

If we execute a program on a quad-core CPU, can we expect a speedup of 4 (relative to a single-core CPU)?

Amdahl's Law: Motivation

If we execute a program on a quad-core CPU, can we expect a speedup of 4 (relative to a single-core CPU)?

Usually not. The speedup of a program using multiple processors in parallel computing is limited by the sequential fraction of the program.

> *"The bearing of a child takes nine months, no matter how many women are assigned."*
>
> F. Brooks, *The mythical man month*

Amdahl's Law

Given

- $B \in [0, 1]$, the fraction of an algorithm that is strictly serial,
- $n \in \mathbb{N}$, the number of threads of execution,

the time that it takes the algorithm to finish when executed on n threads is

$$T(n) = T(1)\left(B + \frac{1}{n}(1 - B)\right)$$

Consequently, the corresponding speedup is

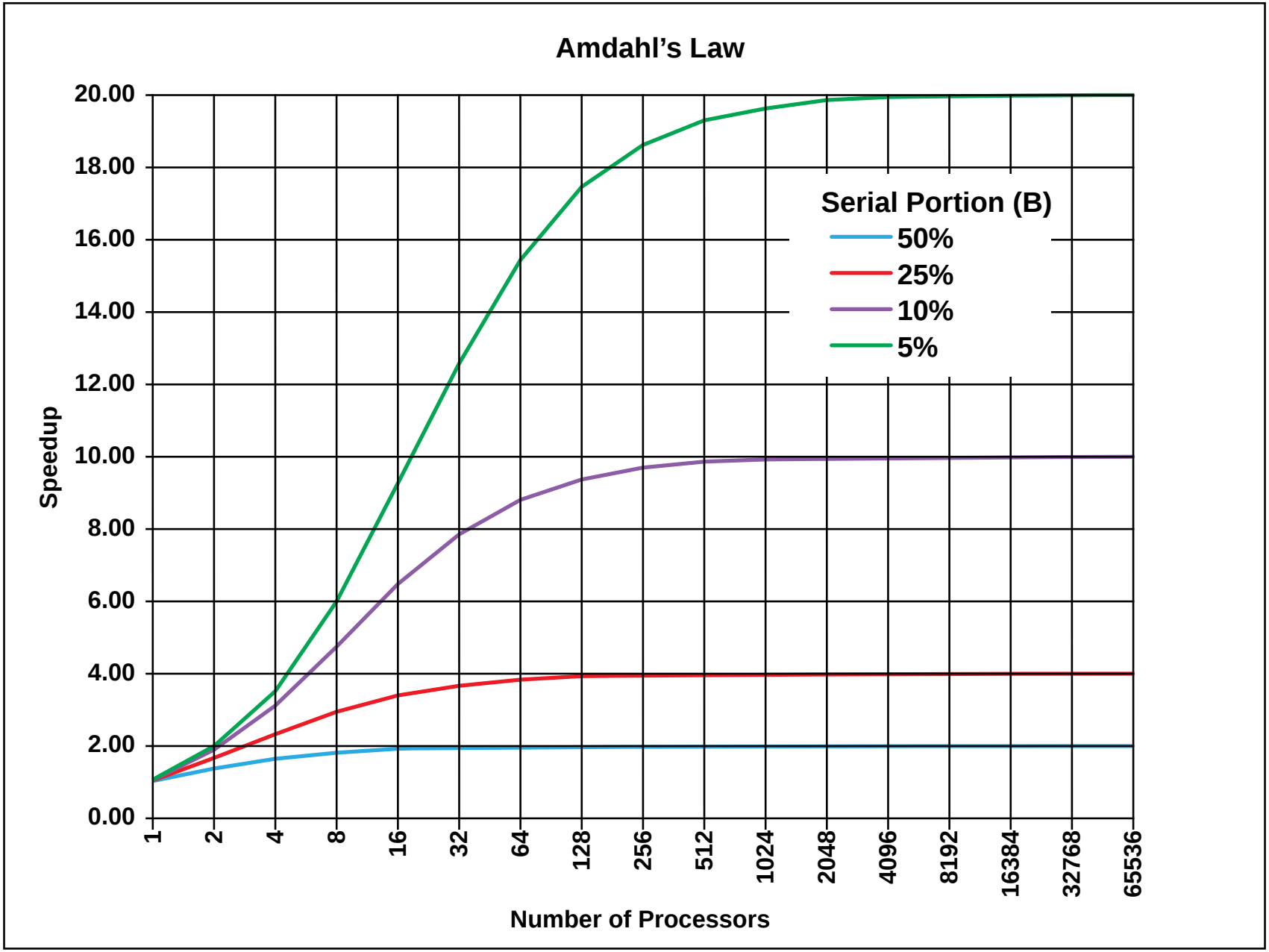
$$S(n) = \frac{T(1)}{T(n)} = \frac{1}{B + \frac{1}{n}(1 - B)}$$

(G. Amdahl, 1967)

Amdahl's Law: Exercise

Choose a $B \in \{0.05, 0.10, 0.25, 0.5\}$. (In other words, 5%, 10%, 25% or 50% of the algorithm are strictly serial.)

- Compute $S(n)$ for $n = 1, 2, 4, 8, 16, 1024$.
- Calculate $\lim_{n \to \infty} S(n)$.


Amdahl's Law
Speedup
20.00
18.00
16.00
14.00
12.00
10.00
8.00
6.00
4.00
2.00
0.00
Serial Portion (B)
50%
25%
10%
5%
1
2
4
8
16
32
64
128
256
512
1024
2048
4096
8192
16384
32768
65536
Number of Processors

Amdahl's Law: Limitations

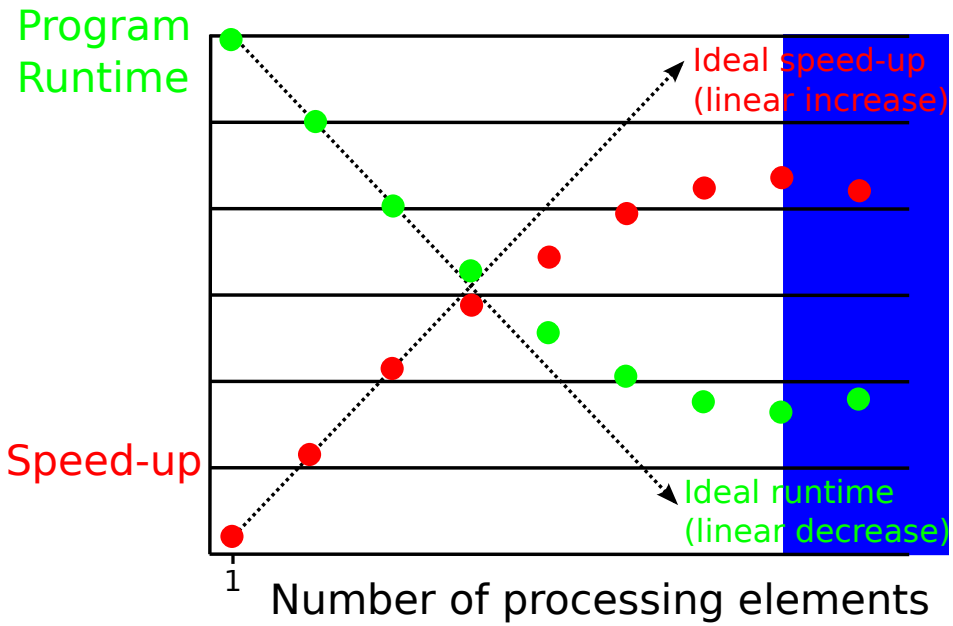
Amdahl's Law assumes that the total amount of work remains the same when parallelized. In practice, this is often not the case.

A concurrent program usually incurs some overhead for

- task creation, task management;
- communication between tasks.

Parallel Slowdown

Parallel slowdown occurs when parallelization beyond a certain point causes the program to run slower. (This is typically caused by a communications bottleneck.)



Super-Linear Speedup

- Let $n \in N$. According to Amdahl's Law, what is the maximal speedup $S(n)$? For which $B \in [0, 1]$ is this speedup attained?
- In practice, is it possible to observe a higher speedup (for some programs)?

  Hint: