Parallel Computing

Academic year – 2020/21, spring semester

Computer science

Lecture 1

Lecturer, instructor:

Balakshin Pavel Valerievich

(pvbalakshin@itmo.ru; pvbalakshin@hdu.edu.cn)

Assistant:

TBD

(TBD@hdu.edu.cn)

About myself

- PhD in Computer Science
- 9 years of teaching experience
- 15 years of IT industry
- Associate professor at ITMO University
- Lead RPA Developer at Masterdata
- Scientific interests: RPA, speech recognition, new IT inventions

Course structure

- **7-8** lectures
- **4** lab works about the following topics:
  - Automatic parallelization of programs
  - Research the effectiveness of multi-threading libraries for C programs
  - OpenMP technology
- **7-8** intermediate test after each lesson
- **1** final test (=exam)

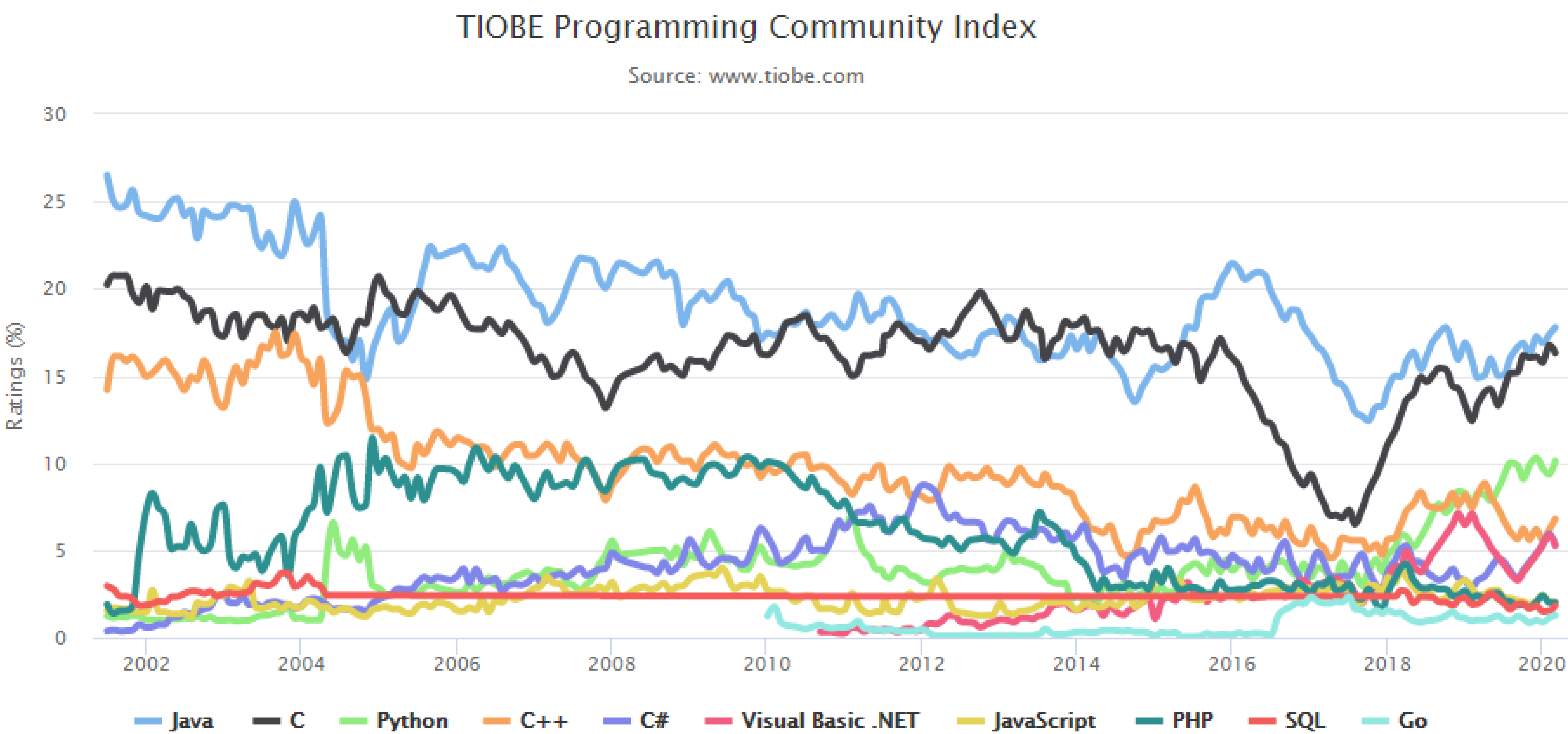
Breakdown of grades

Result score	Result grade	Result mark	Explanation
[0; 59]	2F	Fail	Unsatisfactory
[60;67]	3E	Pass	Satisfactory
[68;74]	3D	Pass	Satisfactory
[75;83]	4C	Pass	Good
[84;90]	4B	Pass	Very good
[91;100]	5A	Pass	Excellent

Penalty

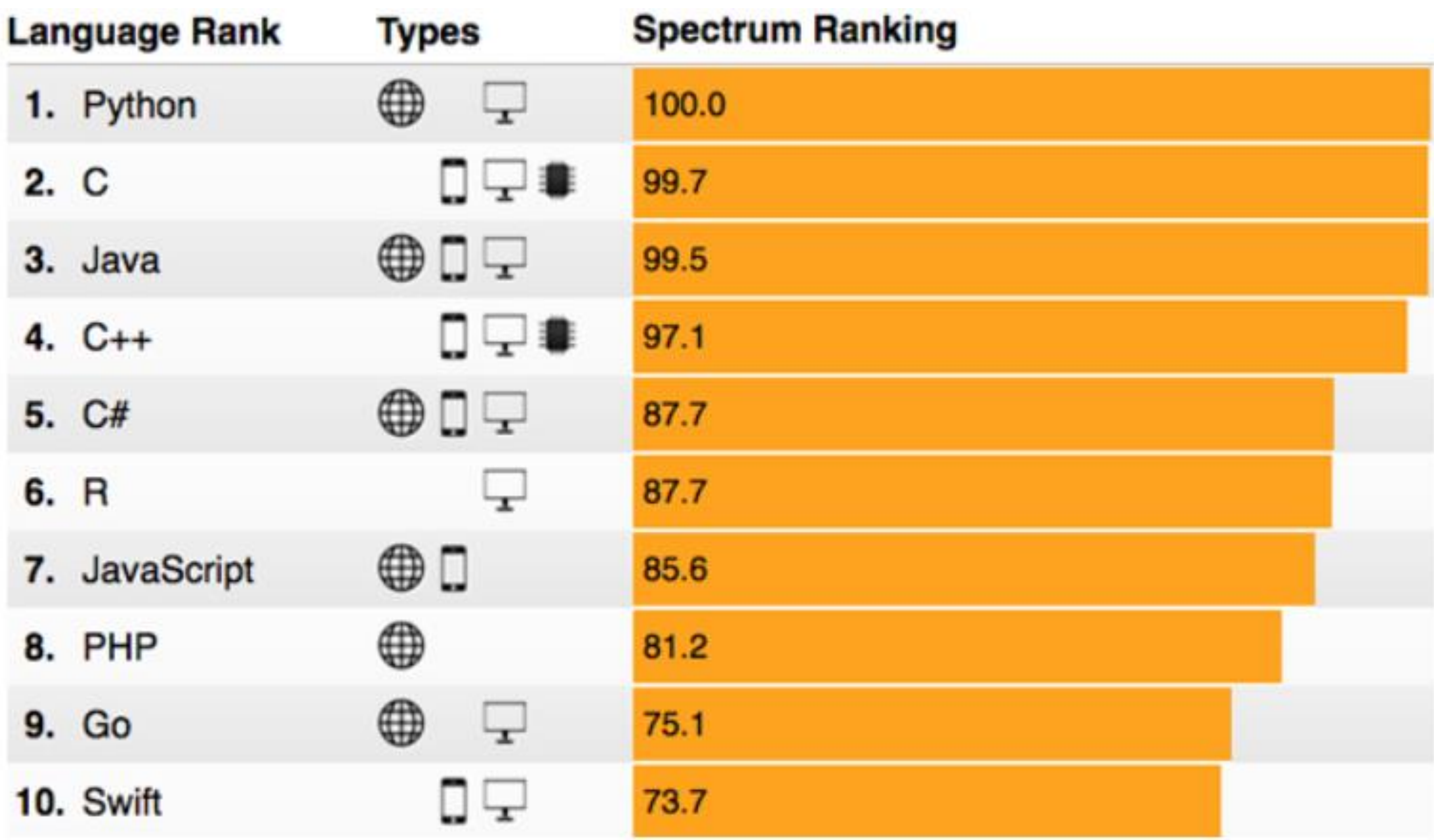
- Participants who successfully complete 70% of the attendance and 70% of class work will be allowed to the exam.

Why C/C++



TIOBE Programming Community Index
Source: www.tiobe.com
Ratings (%)
30
25
20
15
10
5
0
2002
2004
2006
2008
2010
2012
2014
2016
2018
2020
Java
C
Python
C++
C#
Visual Basic .NET
JavaScript
PHP
SQL
Go

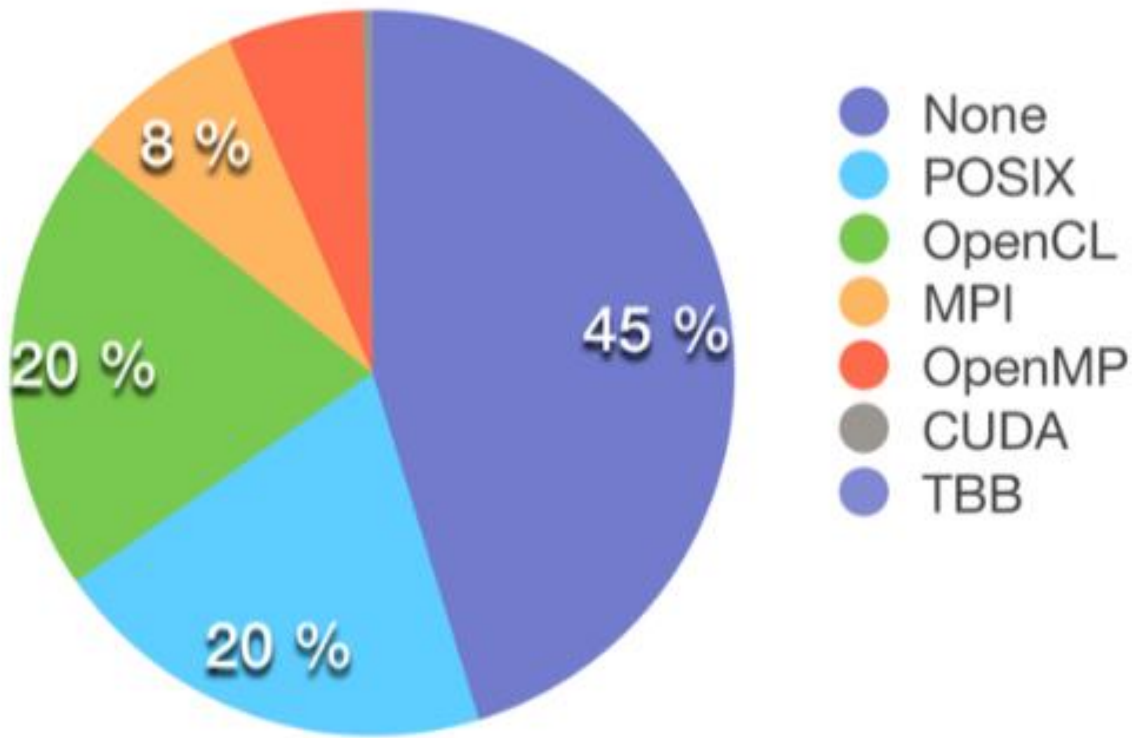
Why C/C++ (2)



Language Rank	Types	Spectrum Ranking
1. Python		100.0
2. C		99.7
3. Java		99.5
4. C++		97.1
5. C#		87.7
6. R		87.7
7. JavaScript		85.6
8. PHP		81.2
9. Go		75.1
10. Swift		73.7

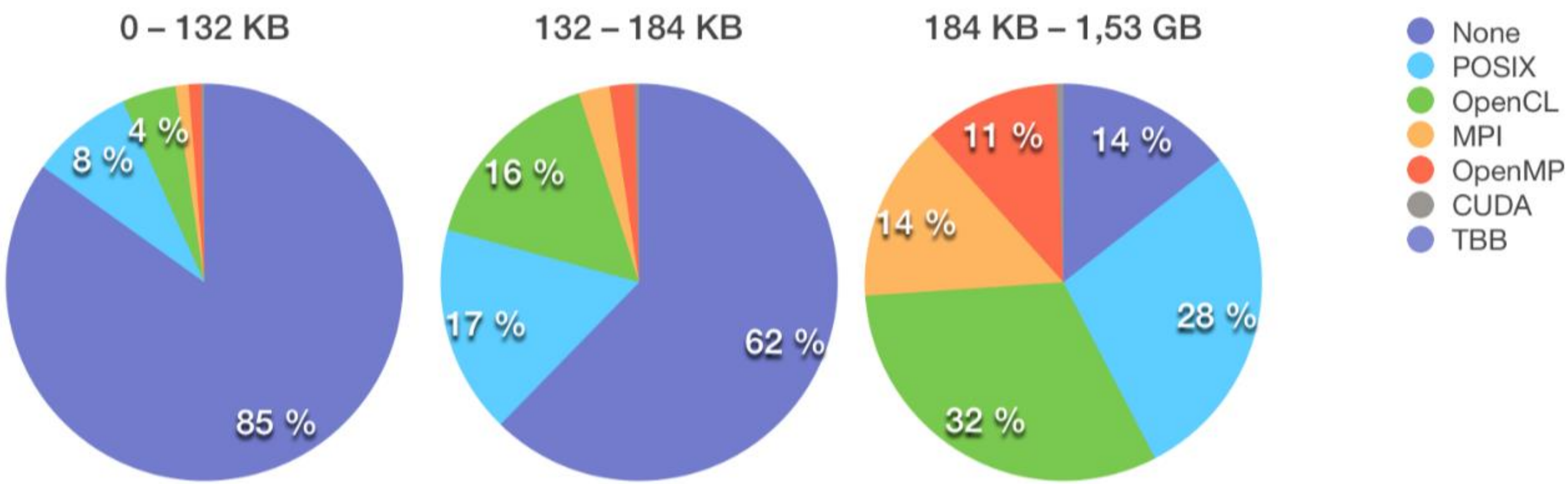
https://spectrum.ieee.org/computing/software/the-2017-top-programming-languages

Choosing a language for parallel programming technology



According to the research of Bernhardt G.V. (2016)

Choosing a language for parallel programming technology(2)



Distribution based on repository size

According to the research of Bernhardt G.V. (2016)

Definitions

Concurrent computing is a way of organizing calculations on one or more computers, where life periods of several tasks intersect.

Sequential computing – there are no overlapping periods of tasks.

Parallel computing – tasks are executed physically simultaneously on different processors and/or cores of the same computer.

Multicore computing – computations when each processor has more then one core.

Shared memory processing (SMP) – refers to the work of parallel programs on systems with shared memory. In such systems all the processors/cores share common memory of one computer.

Distributed computing – sort of parallel computing, at which to calculations take place on processors located on different computers connected by a network, means one has to transfer programs and/or data through a network to perform calculations.

Definitions (2)

Parallel calculations are always concurrent ones.

Not all concurrent calculations are parallel.

Parallel computing = multicore computing = SMP.

Not in scope of the course:

- Bit-level parallelism
- Parallelism at the operand level – concurrency at the data level
- Parallelism at the instruction level – superscalarity
- Preemptive multitasking

Definitions (3)

During the existence of computer technology, the speed of triggering of elements has increased 10^6 times, and the speed of calculations has increased 10^9 times.

The performance of single-processor systems increased by 1.5 times annually from 1986 to 2002.

Since 2002 this increase is only 1.2 times.

Conclusion: Computer science development is history of architectural excellence and practical use of parallelism.

Why do we need parallel computing?

1. Problems of Grand Challenge solution (performance of existing computing systems is not enough > 1 Tflops):

- *climate modeling;*
- *genetic engineering;*
- *integrated circuit design;*
- *environmental pollution analysis;*
- *drug development.*

2. Software development for modern smartphones.

3. Gaming industry.

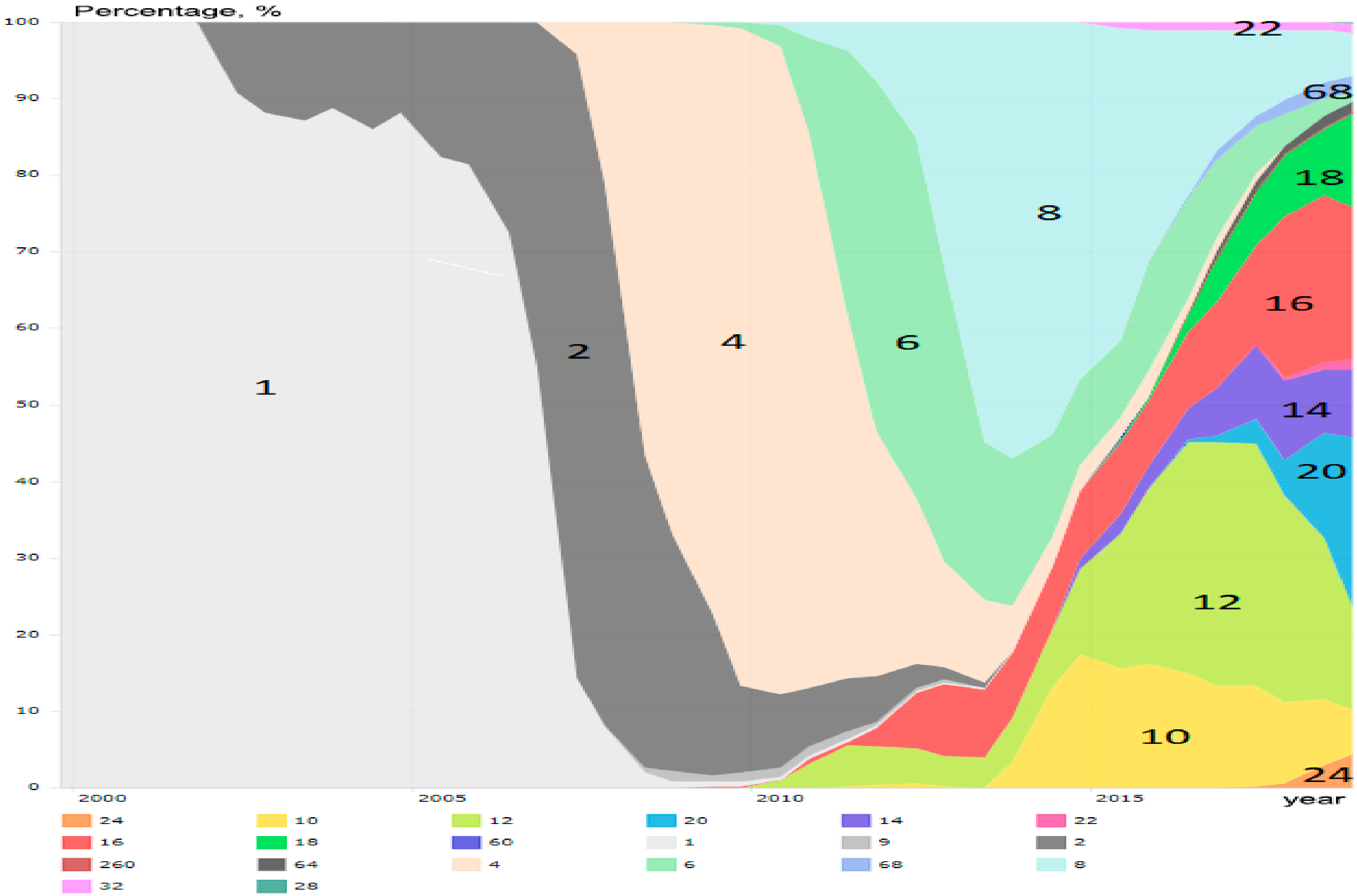
Classification of parallel systems (architectures)

- SMP (Shared Memory Parallelism, Symmetric MultiProcessor system) – multiprocessor, multicore, GPGPU.

- MPP (Massively Parallel Processing) – cluster systems, GRID (distributed computing).

History of SMP-systems development

Number of occupied cores during supercomputers creation



According to top500.org

What **contributes** to the development of parallel computing

- Theoretical limited growth in performance of non-parallel computers.
- Sharp reduction in the cost of multiprocessor (parallel) computing systems.
  - 1 x Cray T90: 1.8 Gflops = $2,5 million (1995)
  - 8 x IBM SP2: 2.1 GFlops = $0.5 million (~2000)
- Appearance of a multi-core processor building paradigm.

According to materials of Prof. V.P. Gergel' (2001)

What slows down the development of parallel computing (1)

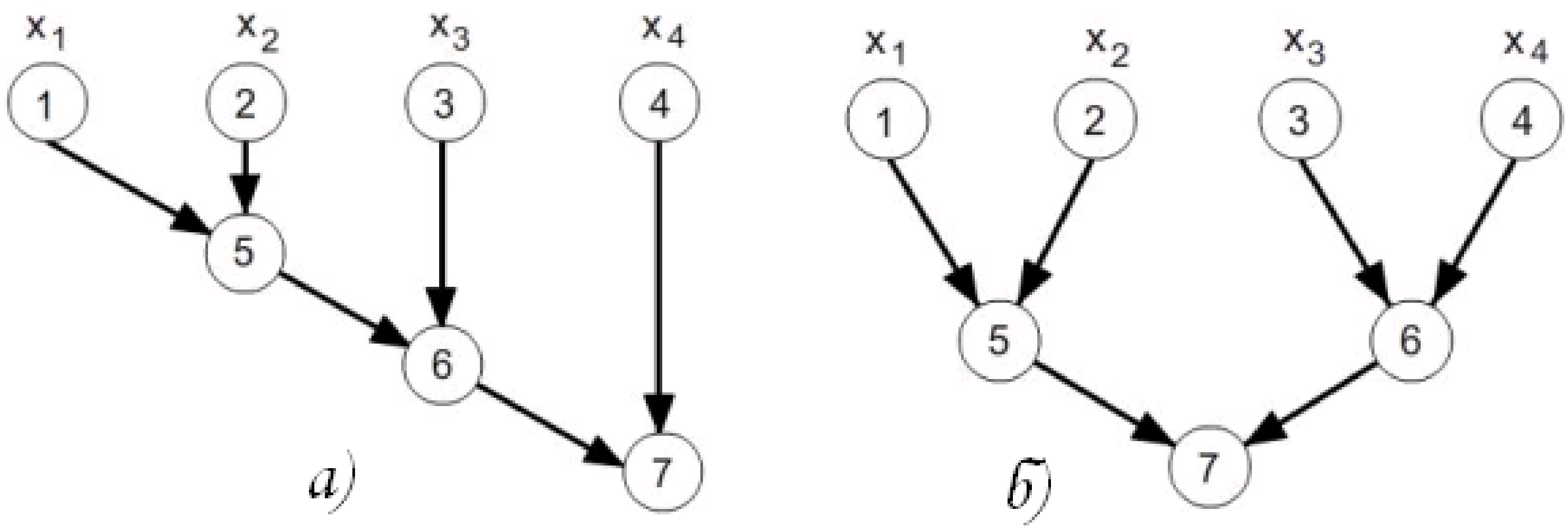
- **Marvin Minsky's hypothesis (1971)**: acceleration of a parallel system is proportional to the binary logarithm of the number of processors.
- **Gordon Moore's law (1975)**: the number of transistors in a dense integrated circuit doubles about every 18 months.
- **Herb Grosch's law (1953)**: computer performance increases as the square of the cost.
- **Difficult to learn** the principles of parallel programming.

What **slows down** the development of parallel computing (2)

- **Gene Amdahl's law (1967)**: any program has a non-parallelized part.
- **Non-universality (not cross-platform) of parallelism**: while programming you should take into account characteristic features of concrete parallel systems.
- **Existing software** is oriented on sequential computers.

Example of paralleling an algorithm (1)

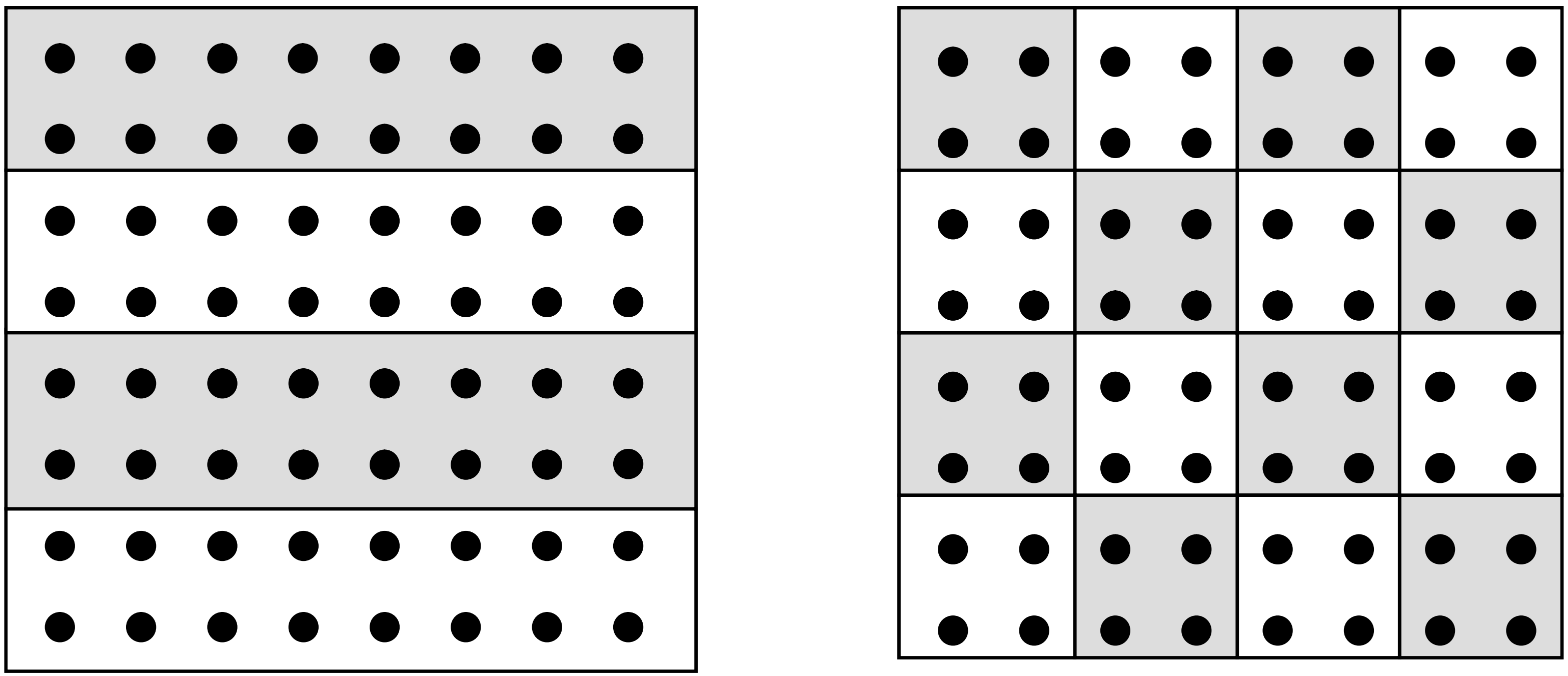
Sequential and cascading summation



According to materials of Prof. V.P. Gergel’ (2001)

Example of paralleling an algorithm (2)

Search for the maximum element of an array



a) б)

According to materials of Prof. V.P. Gergel' (2001)

Example of paralleling an algorithm (3)

Parallel sorting:

- Split the source array into two parts.
- Sort each part independently from its processor.
- Perform merging of the sorted pieces.

Computational complexity:

$C_1*N*N \quad \rightarrow \quad C_1*N/2*N/2 + C_2*N$

Performance indicators of parallel programs

p – number of available calculators (cores, processors)

V – average speed of program execution (arbitrary units of work per second)

$S(p) = V(p) / V(1)$ – parallel speedup

$E(p) = S(p) / p = V(p) / [p * V(1)]$ – parallel efficiency

Amdahl's law

$$S(p)|_{w=const} = \frac{t(1)}{t(p)} = \frac{t(1)}{\frac{k \cdot t(1)}{p} + (1-k) \cdot t(1)} = \frac{1}{\frac{k}{p} + 1 - k}$$

$$E_A(p) = (k + p - p \cdot k)^{-1}$$

w(p) – arbitrary total units of work

t(p) – runtime w using p processors

k – paralleling fraction / portion of parallel instructions

Gustafson–Barsis's law

$$S(p)|_{t=const} = \frac{w(p)}{w(1)} = p \cdot k + 1 - k$$

w(p) – arbitrary total units of work, performed by the program in the time t

Modification of the Amdahl' Law (by Prof. A.V. Boukhanovsky)

$$T_1(N) = t_c(N + M)$$

$$T_P(p, N) = T_{OMP} + \frac{t_c N}{p} + M \cdot t_c$$

$$T_{OMP} = \alpha(p-1)t_c N$$

$$\boxed{S(p, N) = \frac{T_1}{T_P} = \frac{N + M}{\alpha(p-1)N + \frac{N}{p} + M}}$$

N – number of paralleled instructions, M – number of non-paralleled instructions,
t_c – single instruction operation time, p – number of available calculators,
T_i – program execution time during usage of i parallel threads on i-calculators,
α – scaling factor

Comparison with Amdahl