

Parallel Programming

Programowanie równoległe

Lecture 1: Introduction. Basic notions of parallel processing

Paweł Rzązewski

Grading

- ▶ laboratories (4 tasks, each for 3-4 weeks) – total 50 points,
- ▶ final test (the last class) – max. 50 points,
- ▶ your final grade is given by the sum of these,
- ▶ each part has to be passed ($\geq 50\%$)

Schedule of lectures

no	date	topic
1	23.02	basic notions of parallel processing
	02.03	the lecture is cancelled
2	09.03	classical problems in synchronization, semaphores
3	16.03	semaphores, continued
4	23.03	monitors
	30.03	spring holidays
5	06.04	parallel computational models, parallel algorithms
6	13.04	parallel algorithms, continued
7	20.04	models of distributed computation, distributed algorithms
8	27.04	final test

Schedule of labs

week	task
26.02	task 1 is given
19.03	task 1 deadline task 2 is given
09.04	task 2 deadline task 3 is given
07.05	task 3 deadline task 4 is given
04.06	task 4 deadline
11.06	the last deadline for anything

The remaining classes are consultations.

A little more about the labs

task no	max. grade	topic
1	10	semaphores
2	10	monitors
3	15	parallel algorithms
4	15	distributed algorithms

Late submission: -5 pts for each (started) week.

The submission consists of two parts:

- ▶ submitting the files before the deadline (7 AM),
- ▶ coming to the class and presenting the solution.

Submitting files and not presenting them is equivalent to not submitting at all. It is not necessary to submit all tasks in order to pass.

What is this course about?

- ▶ introduction to basic notions of parallelism,
- ▶ typical problems in parallel programs,
- ▶ classical solutions to these problems,
- ▶ ways to describe and compare parallel algorithms,
- ▶ basic parallel algorithms: numerical, graph algorithms,
- ▶ typical problems and solutions in distributed processing,
- ▶ basic distributed algorithms

What is this course **not** about?

- ▶ specific technologies and technology-specific techniques,
- ▶ programming in any specific language,
- ▶ synchronization methods offered by modern high-level languages

What is this course **not** about?





- ▶ specific technologies and technology-specific techniques,
- ▶ programming in any specific language,
- ▶ synchronization methods offered by modern high-level languages

Do not expect to be experts in parallel programming and parallel algorithms design after finishing this short and basic course!

What should you know already?

- ▶ programming in C/C++/C#/Java,
- ▶ basics of operating systems,
- ▶ knowledge about processes, threads etc.,
- ▶ basics of inter-process synchronization (semaphores),
- ▶ how to design, describe and analyze (sequential) algorithms,
- ▶ basic numerical algorithms,
- ▶ basic notions in graph theory

Literature

-  Barney, B.:
[Introduction to Parallel Computing](http://computing.llnl.gov/tutorials/parallel_comp/)
computing.llnl.gov/tutorials/parallel_comp/
-  Cormen, T., Leiserson, C.E., Rivest, R.L., Stein, C.:
[Introduction to Algorithms, 3rd ed.](#),
MIT Press 2009
-  Lynch, N.:
[Distributed Algorithms](#),
Morgan Kaufmann Publishers 1996
-  Grama, A., Gupta, A., Karypis, G., Kumar, V.:
[Introduction to Parallel Computing, 2nd ed.](#),
Addison Wesley 2003

Why parallel processing?

The general idea is to break down the problem into a number of independent pieces, perform the computation concurrently (as separate processes, threads etc.) and then combine the results (see the similarity to *divide & conquer paradigm?*).

Why parallel processing?

It is a lot easier to harness 100 horses than to grow one that's 100 times bigger.

M. Dertouzos

- ▶ we are approaching the limits of current technology
- ▶ computers are cheap, but supercomputers are very expensive

Why parallel processing?

It is a lot easier to harness 100 horses than to grow one that's 100 times bigger.

M. Dertouzos

- ▶ we are approaching the limits of current technology
- ▶ computers are cheap, but supercomputers are very expensive

Parallelism does not substitute modern technologies, but accelerates them.

Pros and cons of parallel processing

- + we can get the results now, without waiting for better technology,
- + in general it's cheaper to buy several „small” computing units than one supercomputer,
- + you can distribute the computation

- parallel programs are difficult to design and analyze (non-determinism),
- debugging is hard,
- parallel programming yields many problems and difficulties (deadlock, starvation), which do not appear in sequential programming

Flynn's taxonomy (1966)

Classification of computer architectures. Many modern architectures are hybrid, but this model is still widely used.

Single	Instruction	Single	Data Stream
Multi		Multi	

Flynn's taxonomy (1966)

- SISD** (Single Instruction, Single Data) – typical sequential machine (older PCs, servers etc.)
- SIMD** (Single Instruction, Multi Data) – the same instructions are performed on different data, often used for numerical algorithms like matrix processing (vector machines, GPUs)
- MISD** (Multi Instruction, Single Data) – different instructions are performed on the same data, this model is rather theoretical and not widely used (space shuttle program – safety critical systems, final results should agree)
- MIMD** (Multi Instruction, Multi Data) – processing units perform computation independently, on different inputs (most multicore architectures)

How to design a parallel solution?

Understand the problem thoroughly. Can it be parallelized?

How to design a parallel solution?

Understand the problem thoroughly. Can it be parallelized?

- ▶ Compute n -th Fibonacci number, $F(n) = F(n-1) + F(n-2)$.

How to design a parallel solution?

Understand the problem thoroughly. Can it be parallelized?

- ▶ Compute n -th Fibonacci number, $F(n) = F(n-1) + F(n-2)$.
- ▶ Calculate the potential energy for each of several thousand independent states of a molecule. When done, find the minimum energy state.

How to design a parallel solution?

Understand the problem thoroughly. Can it be parallelized?

- ▶ Compute n -th Fibonacci number, $F(n) = F(n-1) + F(n-2)$.
- ▶ Calculate the potential energy for each of several thousand independent states of a molecule. When done, find the minimum energy state.

embarrassingly parallel problem – solving many independent tasks with little or no coordination

communication – exchanging the data between processes/threads (shared memory, messages, network packets)

synchronization – coordination of parallel tasks in real time, usually involves waiting for other tasks to finish and communication between processors/threads

Granularity

Granularity describes the ratio between the communication time and the computation time. In *fine-grained* systems single tasks tend to be small, so the processors communicate frequently. On the other hand, in *coarse-grained* systems, tasks are big and the communication happens rarely .

Speed-up

The simplest and the most commonly used measure of parallel programme's performance is the *observed speed-up*.

$$\text{speed-up} = \frac{\text{wall-clock-time-of-serial-execution}}{\text{wall-clock-time-of-parallel-execution}}$$

Note that it is an experimental indicator. We will learn about theoretical ones later.

Scalability

Scalability

Scalability describes how adding more processing units affects the performance.

Strong scaling. Increasing the number of processors does not increase the size of the problem. We want to be able to solve **the same problem** faster.

Perfect situation: doubling the number of processors reduces the computation time by half.

Weak scaling. Increasing the number of processors does not increase the size of the problem **per processor**. We want to be able to solve **larger instances** in the same time.

Perfect situation: doubling the number of processors allows us to solve problems of double size.

Amdahl's law

The scalability has its limits. Let α be the fraction of computation that is (and has to be) sequential. Thus $1 - \alpha$ of computation can be parallelized. Let p denote the number of processors.

Amdahl's law. The maximum speed-up is

$$\begin{aligned}\text{speed-up} &= \frac{\text{wall-clock-time-of-serial-execution}}{\text{wall-clock-time-of-parallel-execution}} \\ &\leq \frac{\alpha \text{ total-num-of-steps} + (1 - \alpha) \text{ total-num-of-steps}}{\alpha \text{ total-num-of-steps} + \frac{1-\alpha}{p} \text{ total-num-of-steps}} \\ &= \frac{1}{\alpha + \frac{1-\alpha}{p}}\end{aligned}$$

Amdahl's law, continued

Suppose that 75% of a program can be parallelized, the rest is necessary to combine the results. With 4 processors, we obtain speed-up at most 2.29. With 8 processors, we get 2.91. Thus, doubling the number of processors gives us the speed-up of 1.27.

Amdahl's law, continued

Suppose that 75% of a program can be parallelized, the rest is necessary to combine the results. With 4 processors, we obtain speed-up at most 2.29. With 8 processors, we get 2.91. Thus, doubling the number of processors gives us the speed-up of 1.27.

num. of proc.	$\alpha = 0.5$	$\alpha = 0.9$	$\alpha = 0.95$	$\alpha = 0.99$
10	1.82	5.26	6.89	9.17
100	1.98	9.17	16.80	50.25
1 000	1.99	9.91	19.62	90.99
10 000	1.99	9.91	19.96	99.02
100 000	1.99	9.99	19.99	99.90

How do we measure speed of computers?

FLOPS = Floating-point Operations Per Second

MFLOPS = 10^6 FLOPS

GFLOPS = 10^9 FLOPS

...

How do we measure speed of computers?

FLOPS = FLloating-point Operations Per Second

MFLOPS = 10^6 FLOPS

GFLOPS = 10^9 FLOPS

...

Currently fastest supercomputer: Sunway TaihuLight (China),
93 PFLOPS = $93 \cdot 10^{15}$ FLOPS

Distributed systems

In distributed systems, the computing elements are independent units (e.g. PCs) connected via some network.

Why do we use distributed systems?

- ▶ they are relatively cheap,
- ▶ load sharing (better usage of computing power),
- ▶ resource sharing,
- ▶ good flexibility – we can add and remove units,
- ▶ safety (even if one machine crashes, we can still perform the computation)
 - ▶ redundancy in memory.

Possible problems in parallel and distributed systems

What are possible problems?

- ▶ system selling plane tickets,

Possible problems in parallel and distributed systems

What are possible problems?

- ▶ system selling plane tickets,
- ▶ sharing network printer,

Possible problems in parallel and distributed systems

What are possible problems?

- ▶ system selling plane tickets,
- ▶ sharing network printer,
- ▶ sharing documents (as in Dropbox),

Possible problems in parallel and distributed systems

What are possible problems?

- ▶ system selling plane tickets,
- ▶ sharing network printer,
- ▶ sharing documents (as in Dropbox),
- ▶ Bitcoin system

Possible problems in parallel and distributed systems

What are possible problems?

- ▶ system selling plane tickets,
- ▶ sharing network printer,
- ▶ sharing documents (as in Dropbox),
- ▶ Bitcoin system

There are only two hard problems in distributed systems:

2. Exactly-once delivery
1. Guaranteed order of messages
2. Exactly-once delivery

Possible problems in parallel and distributed systems

What are possible problems?

- ▶ system selling plane tickets,
- ▶ sharing network printer,
- ▶ sharing documents (as in Dropbox),
- ▶ Bitcoin system

There are only two hard problems in distributed systems:

2. Exactly-once delivery
1. Guaranteed order of messages
2. Exactly-once delivery

There are only two hard problems in computer science: we only have one joke and it's not funny.