Course Outlines

1st Semester

ADVANCED ALGORITHMS

Objectives
get to know a toolbox of different algorithmic paradigms and be able to recognize them.
be able to adjust known algorithmic approaches to a given problem.
be able to recognize structural properties of problems and determine a suitable algorithmic paradigm for solving them.
recognize and prove the hardness of NP-complete problems, as well as show the performance and correctness of (approximation) algorithms.
learn how to develop algorithms for different types of problems based on the concepts introduced in the course.
Contents
In this course, you will learn – or deepen your knowledge of – important algorithm design and analysis paradigms, and build a repertoire of algorithmic solutions to apply in different domains.

Some (tentative) topics are:

1. Introduction: short recap of Algorithm Analysis and Data Structures

2. Network Flows

3. NP and Polynomial-Time Reductions

4. Linear Programming

5. Approximation Algorithms

6. Randomized Algorithms

7. Online Algorithms

Recommended prior knowledge
Basic knowledge of complexity theory, data structures, graphs, and programming (no specific programming language needed). Familiarity with mathematical proofs, basic knowledge of probability and statistics.

Registration
More information about procedures and registration periods can be found at https://student.uva.nl/en/topics/course-registration

Teaching method and contact hours
Self-study
Lecture

Exercise Group
Study materials
Literature:
Jon Kleinberg, Eva Tardos: Algorithm Design
Assessment
The final grade is determined via performance in the final exam, as well as the exercise assignments.

ALGORITHMIC GAME THEORY

Objectives
Learn how to use game-theoretic models to study situations of strategic decision making
Understand and identify the computational limitations and algorithmic challenges that arise
Study and learn fundamental results on the existence, computation and inefficiency of equilibria
Build up a toolbox of models and techniques to study the impact of strategic decision making
Learn how to design simple and efficient algorithms for computationally hard mechanism design settings
Contents
Algorithmic game theory (AGT) is a research area that lies in the intersection of theoretical computer science, discrete mathematics and economic theory. The area builds upon game-theoretic foundations to study situations of strategic decision making, with a particular focus on computational and algorithmic aspects. It combines methodologies and techniques from several disciplines such as complexity theory, algorithm design, discrete and continuous optimization, auction and mechanism design, online decision making and learning, etc.

The overall aim of the course is to get familiar with fundamental methodologies, techniques and results in AGT. We study the existence, computation and inefficiency of equilibria (ranging from pure Nash equilibria to learning outcomes) for various classes of games. We also consider the problem of designing coordination mechanisms to reduce the inefficiency. Further, we touch upon the computational limitations that arise in classic auction theory, and learn about techniques to develop simple and efficient algorithms for fundamental mechanism design problems. Throughout the course, we will build up a toolbox of models and techniques to study the impact of strategic decision making in various settings.

We cover a broad range of topics (tentative list):

selfish routing games and the price of anarchy
Braess paradox, Stackelberg routing, network tolls
best-response dynamics and potential games
existence, computation and inefficiency of equilibria
equilibrium hierarchy, smoothness and extension theorems
sponsored search and combinatorial auctions
simple vs. optimal mechanism design
matching markets and envy-freeness
cost sharing mechanisms
Recommended prior knowledge
Solid knowledge of complexity theory, algorithms and optimization
Basic knowledge of game theory is advantageous
Mathematical maturity: students are expected to have prior experience with working out and writing up mathematical proofs
Registration
More information about procedures and registration periods can be found at https://student.uva.nl/en/topics/course-registration

Teaching method and contact hours
Lecture
Self-study
Tutorials
Study materials
Literature:
T. Roughgarden, Twenty Lectures on Algorithmic Game Theory, Cambridge University Press, 2016. (primary source)
N. Nisan, T. Roughgarden, E. Tardos, and V. V. Vazirani (Editors), Algorithmic Game Theory, Cambridge University Press, 2007. (secondary source)
Other:
Lecture Notes will be provided.
Assessment
Homework Assignments.

Remarks
The link to the Canvas page will be provided in due time.

DATA PREPARATION

Objectives

Describe the lifecycle of data in systems employing machine learning and predictive analytics

Describe and employ fundamental methods for creating and maintaining datasets such as incremental view maintenance, fine-grained data provenance and differential computation

Implement efficient data preparation programs using state-of-the-art relational and dataflow processing systems

Identify and potentially correct data issues related to data quality, privacy violations or technical bias

Design and validate a scalable data architecture for preparing and maintaining data for predictive analytics

Contents

The course recaps fundamentals about relational data processing and dataflow systems as a pre-course for conveying in-depth knowledge about the life-cycle of data in production machine learning deployments.

Students learn about encoding, storing and managing vectorised feature representations of heterogeneous input data sources for predictive analytics, and the architecture of current state-of- the-art systems for this task such as Google's Tensorflow Extended Platform or Uber's Tecton feature store. Concurrently, the students will be exposed to foundational theory for this problem space, such as incremental view maintenance for relational data, fine-grained data provenance tracking via provenance semi-rings and differential computation.

In addition, students will learn to identify and quantify common quality issues with respect to the completeness and consistency of the data, and gain experience in addressing these issues with data cleaning techniques. Furthermore, they will learn about potential discrimination in data-driven decision-making resulting from technical bias in data preparation systems, and technical challenges with respect to the compliance with regulations for private data such as the "right-to-be-forgotten" from GDPR.

Finally, students will be exposed to ongoing research efforts in this space, such as ML pipeline debugging or error detection techniques from data-centric AI.

Recommended prior knowledge

The course requires knowledge of basic notions of computer science in the fields of relational databases, large-scale data engineering and machine learning. Strong programming skills are required.

Registration

More information about procedures and registration periods can be found at https://student.uva.nl/en/topics/course-registration

Teaching method and contact hours

Lecture

Seminar

Computer lab session/practical training

Self-study
Working independently on e.g. a project or thesis
Assessment
The grade is based on individual assignments and a group project. The assignments involve practical programming exercises, theoretical questions and the comprehension of research papers. The project involves the design and implementation of a data preparation system, as well as giving a presentation.

Remarks
The field of data preparation is subject to very active research and is quickly changing. Students will be exposed to current research topics. The contents of the course may slightly change to reflect the latest developments.

HIGH PERFORMANCE COMPUTING AND BIG DATA

Objectives

Identify the appropriate HPC methods and tools to scale up scientific applications
Learn to use Big Data Methods, techniques, and tools to solve data intensive applications
Develop skills to use HPC, and Cloud facilities

Contents

The course covers a number of key topics in the field of high performance computing and big data engineering. The course is organized as a lectures and workshops which help the students to develop both theoretical and practical skills. Following is the list of topics covers in the course:

Introduction to parallel programming models and Big Data
Cluster computing
General-purpose graphics programing unit
Machine Learning
MPI/OpenMP
Data management
Local/Remote Visualisation for Data intensive application
HPC Cloud
Note:

the topics of the workshop may change depending on the availablity of the workshop organisers, or the emergence of a new interesting topic.

Recommended prior knowledge

Students must be able to program in Python, Java, and basic C (or be able to get the needed skills on the fly).

Registration

More information about procedures and registration periods can be found at https://student.uva.nl/en/topics/course-registration

Teaching method and contact hours

Hoorcollege
Workshop

Study materials

Other:
Course is based on Scientific publications and other Material available online

Assessment

The final grade will be based on the results of the

Practical assignments
Reading assignment

Note: since the course is workshop driven, there is no practical or separate lab sessions, lectures and hands-on are given during the workshops.

Remarks

This is an elective course for which the students must seek the commitment of the Examination Board.

The course is composed of set of introductory lectures followed by workshops including hands-on. The workshops are organized by UvA researchers as well as guests from the national Supercomputing Centre (SURFsara). During these workshops, students will have access HPC facilities available at Surf (https://www.surf.nl/en/which-compute-service-for-which-research-question).
(*) list of all available workshops can be found on the course web site: http://www.hpc.uva.nl/Workshops/

INFORMATION THEORETIC LEARNING

Objectives

Identify interpretations of fundamental concepts like "information" and "probability".
Explain the relationships between information theory, Bayesian statistics, Kolmogorov complexity, frequentist statistics and machine learning.
Use the MDL principle to assess under which circumstances a given model will be effective.
Apply efficient coding techniques to data analysis.

Contents

The Minimum Description Length principle is a philosophy of learning that is slightly different from common approaches such as Bayesian inference, classical statistical methods, or other machine learning solutions. It is based on the idea that any pattern or structure you detect in any data set can be translated into a more efficient representation of that data set: the idea is to exploit the strong link between understanding data and being able to compress data. This allows a mathematically rigorous interpretation of Occam's Razor: if several explanations of the data are available, Occam's razor tells us to look for the explanation that offers the best compression.

The philosophy is directly practical and can be applied to statistical questions, such as model selection, parameter estimation, and prediction tasks. Should we choose a complex model with many parameters, or a simpler model that is more easily trained? In this course you will learn to use MDL to solve practical problems, which also provides a new perspective on classical statistical issues.

The course borrows ideas and insights from many different fields, and a major theme is to go into the connections between such diverse fields as information theory, Bayesian statistics, frequentist statistics, Kolmogorov complexity, online learning, machine learning, and finance. The course is mostly theoretical but the homework also includes programming assignments.

Recommended prior knowledge

A course that covers basic probability theory is a necessary prerequisite. Further experience with statistics or information theory is helpful but not required. Students should also have a comfortable command of at least one programming language.

Teaching method and contact hours

Lecture
Seminar
Self-study
Computer lab session/practical training

Study materials

Literature:

All required reading materials will be provided via Canvas. However, the following textbooks are relevant and can be considered by students who already know they wish to learn more about these topics.

P. Grünwald: The Minimum Description Length Principle. MIT press, 2007.
T. Cover and J. Thomas: Elements of Information Theory. 2nd ed. Wiley-Interscience, 2006.
M. Li and P. Vitanyi: An Introduction to Kolmogorov Complexity and Its Applications, 2nd ed. Springer 1997.

Assessment

There is graded homework which includes theoretical questions as well as programming exercises and a final written exam.

INTRODUCTION TO COMPUTATIONAL SCIENCE

Objectives

The aim of this course is to provide an overview of Computational Science and modelling techniques available to the Computational Scientist
The student will be able to describe and apply the basic concepts of modelling and simulation (Validation, Verification, Experimental Frame, etc.)
The student will be able to design and analyse ODEs as an approach to understand and model real world systems (particularly epidemics)
The student will be able to apply methods of network science to model real world phenomena.
The student will be able to compare modelling approaches and understand under which conditions each approach is most appropriate.
The student will build a basic understanding of epidemics and be able to understand the factors in epidemics that make them challenging to model.
The student will be able to implement and analyse computer simulations using Python (and associated libraries)

Contents

An introduction to Modelling and Simulation methods (including Ordinary Differential Equations, Networks, Agent-based Models) applied to the topic of epidemics. The students will have an overview of different modelling techniques that come back later in the programme. They are also introduced to the basic Modelling & Simulation concepts.

Recommended prior knowledge

Basic Calculus, Linear algebra, Programming (python)

Registration

More information about procedures and registration periods can be found at https://student.uva.nl/en/topics/course-registration

Teaching method and contact hours

Lecture
Computer lab session/practical training
Working independently on e.g. a project or thesis

Study materials

Other:
Lecture notes, hand outs, on-line material.
Book (Chapters)

Assessment

written exam (50%)
lab assignments (1 individual (25%) and 1 in pairs (25%))

Remarks

Prerequisites: Academic Bachelor in one of the sciences.

2nd Semester
COMPUTATIONAL COMPLEXITY

Objectives
compare different models of resource-bounded computation
analyze the running time of an algorithm in terms of the input size
formulate how a computational task can be modeled as a decision problem
argue why the approach of distinguishing different complexity classes is useful
critize the framework of worst-case asymptotic complexity analysis
explain the concepts of nondeterminism, alternation, and randomized algorithms
produce a proof of NP-completeness for variants of known NP-complete problems
predict what the implications are if certain complexity classes coincide
classify the location of a decision problem in the landscape of computational complexity
explain how oracles can be used to show the limits of relativizing proofs, and what constraints this places on any proof for P != NP
explain how various complexity-theoretic assumptions (e.g., P != NP, Exponential Time Hypothesis) can be used to rule out efficient algorithms for computational problems
Contents
Computational complexity theory deals with the fundamental question of how many resources (such as time, memory, communication, randomness, etc.) are needed to perform a computational task. A fundamental open problem in the area is the well-known P versus NP problem, one of the Clay Millennium problems. In this course we will treat the basics of complexity theory, including: NP-completeness, diagonalization, non-deterministic computation, alternation, Boolean circuits, randomized computation, approximation algorithms, subexponential-time complexity.

Recommended prior knowledge
Basic math and computer science, and basic knowledge of propositional logic.

Registration
More information about procedures and registration periods can be found at https://student.uva.nl/en/topics/course-registration

Teaching method and contact hours
Lecture
Self-study
Study materials
Literature:
S. Arora and B. Barak. Computational Complexity: A Modern Approach. Cambridge University Press, 2009.
Assessment
The final grade will be based on the results of homework assignments and a final exam.

Remarks
A Canvas site will be available.

COMPUTATIONAL COMPLEXITY

Objectives

compare different models of resource-bounded computation
analyze the running time of an algorithm in terms of the input size
formulate how a computational task can be modeled as a decision problem
argue why the approach of distinguishing different complexity classes is useful
critize the framework of worst-case asymptotic complexity analysis
explain the concepts of nondeterminism, alternation, and randomized algorithms
produce a proof of NP-completeness for variants of known NP-complete problems
predict what the implications are if certain complexity classes coincide
classify the location of a decision problem in the landscape of computational complexity
explain how oracles can be used to show the limits of relativizing proofs, and what constraints this places on any proof for P != NP
explain how various complexity-theoretic assumptions (e.g., P != NP, Exponential Time Hypothesis) can be used to rule out efficient algorithms for computational problems

Contents

Computational complexity theory deals with the fundamental question of how many resources (such as time, memory, communication, randomness, etc.) are needed to perform a computational task. A fundamental open problem in the area is the well-known P versus NP problem, one of the Clay Millennium problems. In this course we will treat the basics of complexity theory, including: NP-completeness, diagonalization, non-deterministic computation, alternation, Boolean circuits, randomized computation, approximation algorithms, subexponential-time complexity.

Recommended prior knowledge

Basic math and computer science, and basic knowledge of propositional logic.

Registration

More information about procedures and registration periods can be found at https://student.uva.nl/en/topics/course-registration

Teaching method and contact hours

Lecture
Self-study

Study materials

Literature:
S. Arora and B. Barak. Computational Complexity: A Modern Approach. Cambridge University Press, 2009.

Assessment

The final grade will be based on the results of homework assignments and a final exam.

Remarks

A Canvas site will be available.

CRYPTOGRAPHIC ENGINEERING

Objectives

Understanding, applying and analyzing the challenges associated with the practical implementation and use of cryptographic primitives (performance vs security trade offs, physical attacks, lightweight designs, etc).

Understanding and applying the various techniques and best practice that can be used to realize efficient and secure implementation of cryptographic algorithms in software (using C and assembly)

Understanding and applying the various techniques and best practice that can be used to realize efficient and secure implementation of cryptographic algorithms in hardware (using VHDL or Verilog)

Understanding and applying physical attacks (side-channels and faults) against cryptographic devices both in a local and in a cloud setting and the countermeasures to mitigate them.

Understanding recent topics on a the domain of the course (the topic will be communicated at the begin of the course and can be privacy preserving techniques, security of shared FPGAs, Physically Unclonable Functions)

Contents

The course explains the principles of cryptographic engineering. The course starts from the definition of some cryptographic algorithms (whose mathematical background will be recalled in the first week of the course). Students will then learn how to efficiently implement them in software and in hardware, and will get familiar with physical attacks, that are the attacks that target the implementation of an algorithm, rather than its mathematical formulation.

Concerning software implementation, students will learn the basic cryptographic implementation techniques, that will be demonstrated on the RISC-V architecture by means of examples, targeting symmetric cryptography (lookup-tables, Sbox and permutation implementations) and asymmetric cryptography (exponentiation, scalar multiplication). The overview of software techniques will be completed by the discussion of optimization on parallel devices and by the presentation of efficient techniques such as bistlicing.

Concerning hardware implementations, students will learn the basic of hardware design flow (simulation, synthesis, place and route), the basics of design in VHDL or Verilog, the methodology for evaluation of hardware designs (area occupation, energy efficiency, low power) and the implementation of finite field arithmetic in hardware. The overview of hardware techniques will be concluded by the presentation of hardware/software co-design of cryptographic primitives.

Concerning the physical attacks, students will learn the most popular side channel and fault attacks, on the embedded devices and on the cloud, and will learn how to implement countermeasures to counteract them.

The course will conclude with a glance on recent topics in the cryptographic engineering domain, possible with the involvement of guest lecturers.

Recommended prior knowledge
The course requires knowledge of basic notions of algebra, computer science, and programming. The course provides RISC-V programming and HDL tutorials, and provides an introduction to finite field arithmetic.

Registration
More information about procedures and registration periods can be found at https://student.uva.nl/en/topics/course-registration

Teaching method and contact hours
Lecture
Computer lab session/practical training
Seminar
Self-study
Study materials
Literature:
Study material will include all the lecture material (slides, lecture notes), online resources, research papers, book chapters, manuals, and code examples.
Other:
Assessment

The final grade is composed based on:

a final exam
practical assignments
research project

Remarks
The language for the course is English.

DATA PROTECTION TECHNOLOGIES

Objectives

Explain the notion of data protection and its connection to related notions (security, privacy, anonymity etc.).
Describe the legal requirements on data protection.
Relate different types of threats to data protection to trust, security, and privacy models.
Explain how data protection technologies based on cryptography, based on the data itself, and based on the IT infrastructure work.
Select appropriate technical measures for data protection in specific application areas.
Evaluate the impact of technical measures on data protection objectives on one hand, and on other quality attributes (such as performance) on the other hand.

Contents

The course (1) explains the context of data protection technologies from multiple angles, (2) conveys in-depth knowledge on fundamental data protection technologies, and (3) showcases selected applications of data protection technologies.

To understand the context of data protection technologies, the basic notions are defined, explaining the relation to and difference from other notions, such as privacy, information security, anonymity, and confidentiality. The students learn about the legal background, different types of security and privacy models, possible attacks on data, and ways to measure the extent to which data is protected. Furthermore, potential conflicts between data protection and other quality attributes (performance, utility, energy consumption) are explored.

The biggest part of the course is about specific data protection technologies. This includes several advanced cryptographic techniques, such as homomorphic encryption and secure multi-party computation. Also methods based on the data itself are discussed, such as anonymization, pseudonymization, and differential privacy. The course also covers methods based on the IT infrastructure for data processing, such as access control and trusted execution environments.

Finally, different applications of data protection technologies are covered. Application areas include privacy-preserving machine learning and location privacy.

Recommended prior knowledge

The course requires knowledge of basic notions of computer science in the fields of algorithms, communication networks, computer architecture, cryptography, databases, machine learning, and programming. Also basic mathematical knowledge is required.

Registration

More information about procedures and registration periods can be found at https://student.uva.nl/en/topics/course-registration

Teaching method and contact hours

Lecture
Seminar
Self-study
Working independently on e.g. a project or thesis

Laptop seminar
Study materials
Literature:
Pointers to research papers will be provided
Syllabus:
Lecture material (slides, lecture notes) will be provided
Min/max participants
80 students maximum

Assessment
The grade is based on assignments during the course and a written exam at the end of the course.

Remarks
The field of data protection technologies is subject to very active research and is quickly changing. Students will be exposed to current research topics. The contents of the course may slightly change to reflect the newest developments.

ENERGY-EFFICIENT EDGE COMPUTING

Objectives

The students understand the importance of low-energy design in a resource-constrained environment at the edge and how it ties to extra-functional properties of edge devices. They understand different hardware and software knobs available on edge devices for improving their energy-efficiency with real-world case studies. They understand importance of hardware-software co-design within edge devices for improving their energy-efficiency.
The students can setup a energy simulation environment in terms of both hardware and software. The students can then perform detailed interval energy simulations.
The students can do a detailed energy-performance analysis of applications on different micro-architectures, and then use that analysis to create energy-efficient hardware-software system design.
The students can evaluate the efficacy of their design in terms of energy-efficiency, overhead, and scalability.

Contents

The course begins by introducing the role of low-energy design in edge computing with an emphasis on energy-efficiency at the system level. It then explains different hardware/software knobs (and techniques) commonly used in edge devices to save energy. This explanation is followed by two detailed real-world research case studies – mobile gaming and Edge AI – wherein these knobs are employed successfully to save power (energy) at the edge.

The course then introduces the use of hardware-software co-design for designing energy-efficient edge devices. Using specimens, the course explains how architecture-specific low-level system software works in tandem with underlying micro-architecture to reduce the energy consumption of edge devices. The course also emphasizes the importance of the scalability of the proposed design with examples.

The last leg of the course focuses on the importance of low-energy design in designing thermally safe and physically reliable edge devices. The course explains the use of different state-of-the-art tools used to study thermals of edge devices, and how these tools are used to perform their thermal management and to increase their reliability (lifetime).

The project (and the assignments) for the course will be based on interval simulation tool-chain called HotSniper. Assignments will be in form of milestone deliverables for the larger project.

Recommended prior knowledge

The course requires knowledge of basic notions of computer science in the fields of computer architecture and operating systems. Also, some mathematical knowledge is required. The course project requires major experience in programming in C++ and minor experience in programming with Python.

Registration

More information about procedures and registration periods can be found at https://student.uva.nl/en/topics/course-registration

Teaching method and contact hours
Lecture
Laptop seminar
Seminar
Working independently on e.g. a project or thesis
Self-study
Study materials
Software:
https://github.com/anujpathania/HotSniper
Assessment
The grade is based on the project deliverables and a written exam. Students must pass both parts of the assessment to pass the course.

Remarks
This course is based on ongoing cutting-edge research on low energy design. There is no text book for the course, but will be based on published research and white papers. The course will not be recorded.

FUNCTIONAL PROGRAMMING

Objectives

Students will understand the basics of Haskell: types, functions, lists and recursion.
Students will understand and be able to use type classes and abstractions such as applicatives and monads to write concise and well-structured Haskell programs.
Students will be able to implement model checking and other logic algorithms.
Students will be able to use formal specification and property-based testing to test programs and to refute conjectures.
Students will be able to use functional programming to investigate a research question.
Students will be able to explain and document their own programs.

Contents

Functional Programming is a paradigm that focuses on functions instead of objects and states. Functional programs consist of definitions instead of instructions: We tell the compiler what we want to compute and not how to compute it.

Haskell is a lazy, strongly typed and purely functional programming language used in academia and industry. Based on lambda calculus and borrowing ideas from category theory, Haskell is an excellent tool for the mathematically inclined programmer. From a logician's point of view, the algebraic data types in Haskell make it very suitable to work with formal languages.

This course starts with a fast introduction to Haskell and then focuses on how it can be used as a research tool. One of the application topics discussed are implementations of explicit and symbolic model checking for modal logics.

The course finishes with a group project for which a list of topics will be provided, but in consultation with the lecturer students may also choose their own topic from the whole Master of Logic spectrum.

Recommended prior knowledge

Basic knowledge of modal logic and set theory is expected. Courses on lambda calculus or category theory are a plus, but not necessary.

No programming experience is expected. On the contrary: If you have programmed in imperative languages before, be prepared to unlearn a few things.

Registration

More information about procedures and registration periods can be found at https://student.uva.nl/en/topics/course-registration

Teaching method and contact hours

Lecture
Laptop seminar
Self-study
Presentation/symposium
Working independently on e.g. a project or thesis

Study materials

Literature:

Miran Lipovača (2011): Learn You a Haskell for Great Good!
Bryan O'Sullivan, Don Stewart, and John Goerzen (2008): Real World Haskell
Bartosz Milewski (2014): Category Theory for Programmers
Malvin Gattinger (2018): New Directions in Model Checking Dynamic Epistemic Logic
Software:
Haskell, to be installed using GHCup
Other:
Additional material and links will be provided on Canvas.
Assessment
There will be (bi)weekly homeworks, a midterm exam and a group project delivering a presentation and a report.

Remarks
For all information during the course, please refer to the Canvas page.

MULTI-CORE PROCESSOR SYSTEMS

Objectives

To understand the principles and design of modern multi-core microprocessors and their microarchitectures.
To understand how such processors can be deployed in systems targeted for different application fields, such a as embedded / edge systems-on-a-chip or parallel/distributed computing systems.
Building a simulation model of a cache-coherent multi-core system, with which cache behavior in such systems can be analyzed and evaluated.
To develop an understanding of how such multi-core processors can be programmed.

Contents

This course builds on a basic knowledge of microprocessor architecture. It develops this with and emphasis on instruction-level and thread-level concurrency in (multi-core) microprocessor design as well as concurrency in memory systems. The topics that are covered include superscalar and VLIW processor architectures, instruction- and thread-level parallelism, memory hierarchy, distributed- and shared-memory parallel computers.

The course also provides an introduction into programming models and techniques for multi-core processors found in everyday laptops to large-scale server systems. Here, the focus will be on multithreaded programming models for shared address space systems, where we discuss both OpenMP compiler directives and Posix threads.

Recommended prior knowledge

The course requires a solid background in system-oriented programming using the C or C++ programming language, and basic knowledge on computer organization.

Registration

More information about procedures and registration periods can be found at https://student.uva.nl/en/topics/course-registration

Teaching method and contact hours

Lecture
Self-study
Computer lab session/practical training

Study materials

Literature:
John Hennessy and David Patterson, Computer Architecture: a Quantitative Approach, Morgan Kaufmann, 3rd ed. or later.
Practical training material:
Study material for the part on parallel programming will be provided during the course.

Assessment

Practical assignments, theory tests/assignments, review of a scientific paper.

SECURE COMPUTATION

Objectives

Explain the foundational techniques and building blocks used in designing secure computing protocols such as garbled circuits, secret sharing and oblivious transfer.

Assess the security of the protocols using the standard simulation-based security model

Describe protocols designed for special secure computing tasks such as zero-knowledge proofs, private set intersection and private information retrieval

Examine the complexity and security tradeoffs with respect to different adversarial models and protocol design approaches

Relate the concepts studied to real-world applications in finance and healthcare.

Contents

Secure Computation is a central tool in cryptography that allows multiple distributed parties to jointly compute on their private inputs, with the guarantee that nothing beyond the output is revealed. The course (1) gives a comprehensive treatment of foundations of secure multi-party computation (2) covers protocols for special secure computing tasks that are widely used in practice.

The first part of the course focuses on foundations of secure multi-party computation (MPC), which includes both protocol design and formal security analysis. In more detail, the students would (a) learn classic MPC protocols such as GMW, BGW and Yao along with an extension to MPC in the pre-processing model (b) learn how to write and analyze proofs in the simulation-based paradigm, which is the standard security framework used in the context of MPC protocols. The protocols covered would expose students to popular building blocks in secure computing such as oblivious transfer, commitment schemes, secret sharing and garbling.

While the first part of the course involves protocols for general secure computing tasks, the second part of the course focuses on customized protocols for special computing tasks such as zero-knowledge proofs, private set intersection and private information retrieval. Use cases of real-world systems where these protocols have been / could be deployed are also discussed, including ZCash (a privacy-focused cryptocurrency) and the Apple CSAM Detection System.

In summary, the course would enable the students to acquire the basic concepts and skills needed to follow the current research trends in theory and practice of secure computation. Finally, via seminars, the students would get a glimpse into state-of-the-art protocols that are built upon the classic protocols covered during the course.

Recommended prior knowledge

The course requires basic mathematical knowledge. A recap of relevant mathematical concepts is covered during the course. A background in Cryptography is useful but not mandatory.

Teaching method and contact hours

Lecture

Seminar

Presentation/symposium

Working independently on e.g. a project or thesis
Self-study
Study materials
Literature:
Selected Chapters from the following textbooks: 1. Ronald Cramer, Ivan Damgård and Jesper Buus Nielsen. Secure Multiparty Computation and Secret Sharing. 2. David Evans, Vladimir Kolesnikov and Mike Rosulek. A Pragmatic Introduction to Secure Multi-Party Computation The main material for the students would be slides, lecture notes and pointers to the relevant parts of the above textbooks and other online resources.
Assessment
The grade is based on assignments and a written exam. The assignments would involve reading research papers (or other online resources such as lecture notes) and either presenting them and / or submitting a report. Depending on the number of students, the assignments would be designed either for an individual or a group.

WEB SERVICES AND CLOUD-BASED SYSTEMS

Objectives

Design and develop Service Oriented Architecture/Microservice Architecture
Design a distributed systems using Cloud appraoch
Develop cloud-aware applications
Read and analyze scientific publications on Cloud computing and Cloud systems
Write/Presentation a literature study Cloud related topic

Contents

Introduce students to the principles of web services and cloud systems. Students will learn about the different paradigms of cloud systems (IaaS, PaaS, SaaS), and understand the mechanisms and technologies behind each mode to successfully harness cloud resources. A number of real use case studies of existing cloud systems, and service-based appliations on clouds will be covered during the lectures. The course will also cover more advanced topics such as security of clouds and multi-clouds.

Registration

More information about procedures and registration periods can be found at https://student.uva.nl/en/topics/course-registration

Teaching method and contact hours

Lecture
Seminar
Computer lab session/practical training
Fieldwork/excursion
Study materials
Other:
Course is based on Scientific publications and other material available online.

Assessment

The final grade will be based on the results of the Literature study, Participation in the paper readings, Lab Session

Literature study (Report + Presentation): each group of students will have to select a topic, read scientific papers on the topic, write a report in the form of scientific paper (8 pages Max) and prepare a presentation (15mn). A list of topics will be provided, including one paper as suggestion of a starting point
Home work: three or four papers will be given as homework, students will have answer a quiz during the lecture sessions.
Lab sessions: students work in group of two or three students, during the demonstration of the outcome of the labs (small project), both students are requested to be present. Questions will be asked to both students.
Students will have to complete tutorial on certain topics covered during the lecture.

Remarks

Prerequisites: Minimal knowledge in progamming and scripting .