

5.00 credits

30.0 h + 15.0 h

Q2

Teacher(s)	Barbette Tom ;
Language :	English > French-friendly
Place of the course	Louvain-la-Neuve
Prerequisites	Required#: principles of computer systems, as targeted in course LINFO1252
Main themes	For a long time, general-purpose CPUs focused on supporting efficiently a single thread of execution. Improvements in chip manufacturing allowed packing more transistors on the same surface of a silicon wafer, and run resulting CPUs at higher frequencies. Single-threaded applications would simply run faster with every new processor generation. This era is now over. The industry hit several limitations known as the power wall, the memory wall and the ILP wall. No longer able to vertically scale-up CPUs supporting a single thread of execution, chip manufacturers have started packing together multiple, simpler units of execution, or cores. Exploiting the power of multiple cores requires exploiting parallelism in applications using multiple threads. Writing concurrent code requires identifying and managing concurrency, and introducing the necessary synchronization for correctness. Writing scalable and performant concurrent code requires understanding this tradeoff between synchronization and parallelism and mastering efficient implementations of shared data structures and algorithms for concurrent execution. Finally, multicore CPUs employ a complex memory layout, and the assumption of uniform memory access times is no longer valid. Understanding the impact of non-uniform memory accesses (NUMA) is therefore important to write efficient code for multicore CPUs. This course will provide students with the necessary tools and knowledge to write efficient and scalable code for modern multicore CPUs. It will detail the mechanisms available for synchronization, starting from the implementation of language constructs such as locks, monitors or condition variables, to the direct uses of CPU-provided synchronization primitives (e.g. compare-and-swap) to build efficient and scalable data structures. It will emphasize the performance aspects of multicore programming: the impact of synchronization primitives, the impact of non-uniform memory access, and the impact of multiple-level memory hierarchies. It will finally offer an opening to the future of multicore programming with an introduction to transactional memory and to the support for speculative execution in modern CPUs (e.g. Intel Haswell), and discuss the execution model for concurrent code running on GPUs.
Learning outcomes	At the end of this learning unit, the student is able to : Given the learning outcomes of the "Master in Computer Science and Engineering" program, this course contributes to the development, acquisition and evaluation of the following learning outcomes: • INFO1.1 • INFO3.1 • INFO6.4 Given the learning outcomes of the "Master [120] in Computer Science" program, this course contributes to the development, acquisition and evaluation of the following learning outcomes: • SINF1.M3 • SINF3.1 • SINF6.4 The students shall acquire advanced theoretical knowledge and technical competences about the topics covered in the course.

Evaluation methods	Assessment methods: • Projects and continuous assessment, including a final presentation • Written exam consisting of two parts: - o Part one: course-based questions (open-ended questions and multiple-choice questions) - o Part two: questions based on one or more research articles in the field, provided to students in advance. The weighting applied to continuous assessment (with the remainder of the grade based on the individual exam) is calculated as follows: • 40/100 if the individual written exam grade is above 10/20; • 0 if the individual written exam grade is below 5/20; • Linearly scaled between 0 (for an exam grade of 5/20) and 40/100 (for an exam grade of 10/20). For the second session (August), the project and continuous assessment components may be retaken as a single new project, but only on an individual basis. Assessment of submitted work (projects, practical assignments, individual work, and generally any activity subject to evaluation) is based on the materials (code, reports, and any other items) submitted by the student, accompanied by a final presentation. Unless explicitly stated otherwise, all assigned work must remain confidential and individual. Sharing information and/or code with other students (except with a designated partner, where applicable) constitutes plagiarism. Grades are always assigned individually, even for work done in pairs. Rules regarding the use of artificial intelligence (AI) for continuous assessment activities: The use of AI to complete practical assignments and the mini-project is discouraged, as completing them manually is the best way to acquire the skills targeted by the course. If such use is permitted (whether for generating code or for documentation), it is subject to the following rules: • Students must be fully responsible for their work and able to explain orally all code and deliverables (documentation, deployment scripts, etc.) submitted as part of the practical assignments. • The use of AI must be precisely documented in the submission's documentation, in a dedicated section specifying which AI tools were used and for which parts. Submitting code or documentation—whether partially or entirely AI-generated—without documenting such use will be considered plagiarism. Students who did not use AI must also state this in the section. • Excessive use of AI that undermines the acquisition of the knowledge targeted by the project may be deemed an irregularity under Section 7 (Articles 107 et seq.) of the General Regulations on Studies and Examinations (RGEE), entailing all associated consequences as set out in Articles 111 et seq. of the RGEE. In the event of suspected excessive AI use in the submitted project, or an incomplete or inaccurate report regarding AI usage, the course instructor may summon the student concerned for a supplementary oral assessment and take appropriate measures in consultation with the head of the EPL examination boards.
Teaching methods	• Lectures • Readings and/or production of tutorial videos • Practical sessions • Mini project
Content	This course targets programming models and methods for scalable applications on modern multi-processor and multi-core architectures. In a first short part, it provides the necessary elements of theory and defines consistency protocols, in order to be able to understand the challenges and tradeoffs associated with shared-memory concurrent programming. The emphasis is on performance and scalability aspects (efficient simultaneous use of multiple cores). The rest of the course surveys a number of fundamental algorithmic techniques for building shared memory concurrent data structures. It studies the performance implications of these data structures and algorithmic constructs in the context of modern architectures, taking into account various aspects such as the memory and cache hierarchy, hardware consistency protocols, and non-uniform memory accesses (NUMA). The course is accompanied by a number of practical projects. A multi-core machine is available for the experiments. Students will be able to evaluate the performance and scalability of various algorithms and data structures seen in class.
Bibliography	The Art of Multiprocessor Programming, Maurice Herlihy and Nir Shavit, Morgan Kaufmann. ISBN 978-0-12-370591-4. UCL library reference 10.620.426
Other infos	All relevant course material and slides as well as practical information related to the course will be accessible on Moodle, which will also be the primary means of communication between the teacher(s) and the students.
Faculty or entity in charge	INFO

Programmes containing this learning unit (UE)				
Program title	Acronym	Credits	Prerequisite	Learning outcomes
Master [120] in Computer Science and Engineering	INFO2M	5		
Master [120] in Computer Science	SINF2M	5		
Master [120] in Data Science Engineering	DATE2M	5		
Master [120] in Data Science: Information Technology	DAT12M	5		