COMPUTER ENGINEERING (LM55)

(Lecce - Università degli Studi)

Teaching PARALLEL ALGORITHMS

GenCod A003130

Owner professor Massimo CAFARO

Reference professors for teaching Massimo CAFARO, MARCO PULIMENO

Teaching in italian PARALLEL ALGORITHMS

Teaching PARALLEL ALGORITHMS

SSD code ING-INF/05

Reference course COMPUTER ENGINEERING

Course type Laurea Magistrale

Credits 9.0

Teaching hours Ore-Attivita-frontale: 81.0

For enrolled in 2021/2022

Taught in 2022/2023

Course year 2

Language INGLESE

Curriculum PERCORSO COMUNE

Location Lecce

Semester Secondo-Semestre

Exam type Orale

Assessment Voto-Finale

Course timetable https://easyroom.unisalento.it/Orario

BRIEF COURSE DESCRIPTION	The course provides a modern introduction to design, analysis and implementation of sequential and parallel algorithms. In particular, the course is based on a pragmatic approach to parallel programming of message-passing algorithms through the C language and the MPI library.
REQUIREMENTS	Calculus I and II, Probability Theory. Programming skills and working knowledge of the C programming language.

COURSE AIMS

Knowledge and understanding. Students must have a solid background with a broad spectrum of basic knowledge of sequential and parallel algorithms:

· the students must have the basic cognitive tools to think analytically, creatively, critically and in an inquiring way, and have the abstraction and problem-solving skills needed to cope with complex systems;
· they must have a solid knowledge of the design and implementation of sequential and parallel efficient algorithms;
· they must have the tools for analysing the resources used by algorithms;
· they must have a catalogue of the most well-known and efficient sequential and parallel algorithms for basic computational problems.

Applying knowledge and understanding. After the course the student should be able to:

· Describe and use the main design techniques for sequential algorithms;
· Design, prove the correctness and analyze the computational complexity of sequential algorithms;
· Understand the differences among several algorithms solving the same problem and recognize which one is better under different conditions;
· Describe and use basic sequential algorithms;
· Describe and use basic data structures; know about the existence of advanced data structures;
· Understand the difference between sequential and parallel algorithms;
· Design, implement and analyze message-passing based parallel algorithms in C using the MPI library;
· Describe and use basic parallel algorithms.

Making judgements. Students are guided to learn critically everything that is explained to them in class, to compare different approaches to solving algorithmic problems, and to identify and propose, in an autonomous way, the most efficient solution they find.

Communication. It is essential that students are able to communicate with a varied and composite audience, not culturally homogeneous, in a clear, logical and effective way, using the methodological tools acquired and their scientific knowledge and, in particular, the specialty vocabulary. The course promotes the development of the following skills of the student: ability to expose in precise and formal terms an abstract model of concrete problems, identifying the salient features of them and discarding the nonessential ones; ability to describe and analyze an efficient solution to the problem in question.

Learning skills. Students must acquire the critical ability to relate, with originality and autonomy, to the typical problems of data mining and, in general, cultural issues related to other similar areas. They should be able to develop and apply independently the knowledge and methods learnt with a view to possible continuation of studies at higher (doctoral) level or in the broader perspective of cultural and professional self-improvement of lifelong learning. Therefore, students should be able to switch to exhibition forms other than the source texts in order to memorize, summarize for themselves and for others, and disseminate scientific knowledge.

TEACHING METHODOLOGY

The course aims to enable students to abstract formal algorithmic models and problems from concrete computational problems, and to design efficient algorithmic solutions for them. This will be done using the following teaching method. Every computational problem will be introduced, motivating it with concrete examples. The presentation of each topic will be divided into four parts: 1. Description of the actual computational problem. 2. Modelling the real problem by means of an abstract problem. 3. Resolution of the abstract problem through an algorithm obtained through the application of the general techniques of design of algorithms introduced in the course. 4. Analysis of the resources used by the algorithm. The course consists of frontal lessons, and classroom exercises. There will be theoretical lessons aimed at learning the basic techniques for the project and analysis of algorithms, and a part of lessons devoted to exercises in which we will illustrate, with plenty of examples, how the theoretical knowledge acquired can be used in order to solve algorithmic problems of practical interest and implement parallel algorithms in C language through the MPI library.

ASSESSMENT TYPE

The exam consists of a written test and a prototype implementation of a parallel software. The written test (2 hours, 21 points out of 30) covers theoretical topics related to the design and analysis of sequential and parallel algorithms in order to verify the student's knowledge and understanding of the materials. In order to pass the written test, students must obtain at least 9 points out of 21. The parallel software project (9 points out of 30) is meant to verify the practice of parallel programming and the ability of the student to implement theoretical parallel algorithms or to parallelize a sequential algorithm. In order to pass the parallel programming challenge, students must obtain at least 4 points out of 9. Both the written test and the prototype implementation of parallel software are mandatory. Students must pass both the written test and the parallel programming challenge; however, the overall exam is passed only if the sum of points obtained in the written test and the parallel programming challenge is greater than or equal to 18 points.

The problem to be solved in parallel (or a sequential algorithm to be implemented in parallel) is assigned upon request to the student. The assigned parallel software project can be discussed only if a written test has been successfully passed; moreover, the parallel software project must be mandatorily discussed into the same trial in which the written test has been passed.

The report on the assigned parallel software project must be structured as follows.

1. Introduction. The student must provide an accurate description of the assigned project, including an analysis of the sequential algorithm that solves the problem addressed in the project. If the student deems it important for understanding, then pseudo-code, examples, graphs, figures, application instances etc. may be included;

2. Parallel design. This is a preliminary study aimed at highlighting the opportunities for parallelism inherent in the problem and/or sequential algorithm. Starting from the pseudo-code, it is then necessary to determine the most appropriate functionalities, code parts and/or data structures that can be considered for parallelisation. At this stage, alternative designs for different parallelisation strategies must be considered and discussed, duly justifying why some operations lend themselves to effective parallelisation and others do not. Furthermore, the expected results must be reported, including a worst-case parallel complexity analysis. Clarity of exposition regarding the parallelisation strategy is required, therefore the data structures used must be described in full, justifying the choices made to minimise the burden of communication and synchronisation. The theoretical analytical evaluation of the scalability of the parallel version is mandatory, and must be performed using both the isoefficiency and the scalability function.

3. Implementation. The parallel design adopted must be implemented using the programming language C or, possibly, C++ (so as to use the data structures present in the STL of C++). The code must necessarily be appropriately commented. In the event that the student verifies the existence of multiple parallelisation strategies for the same assigned problem, an implementation can be provided for each strategy, discussing the advantages and disadvantages of each strategy. Note that the parallel code does not have to be given in full in the text of the report, as the project code - source and Makefile for the build (alternatively CMake can be used) - must in any case be delivered separately. However, the report may include some code snippets relating to the most critical and interesting parts.

4. Debugging and testing. Debugging and testing of the parallel code is recommended in order to reasonably verify the absence of bugs and the correctness of the parallel algorithm in relation to the output produced. The student is encouraged to design and verify unit tests deemed suitable for assessing the goodness of the code. The student is explicitly warned that the implementation of an incorrect parallelisation strategy and/or the presence of bugs that crash the parallel application at

runtime will result in failing the exam, while the presence of bugs that affect the correctness of the algorithm will result in a penalty relative to the mark that will be awarded.

5. Performance and scalability analysis. The student must analyse the performance of the developed implementation in terms of execution time, memory occupation (if necessary), speedup and efficiency. Both strong scalability (Amdahl) and weak scalability (Gustafson) must be assessed where possible.

6. Synthesis. The report must be as concise as possible, but without detriment to clarity of presentation.

7. Project evaluation. The project will be assessed on a scale of 1 to 9 points. The grade will only be determined at the end of the project discussion. The complexity of the problem assigned will be taken into account in the evaluation; students dealing with simpler problems should therefore expect less leniency in the evaluation than students dealing with more difficult problems.

Assessment will also take into account:

- Clarity and effectiveness of presentation;
- Depth, correctness and originality of theoretical analysis;
- Technical ability and documentation of implementation;
- Number and quality of multiple parallel strategies, if any;
- Critical thinking in performance evaluation and analysis;

Significance of results: since parallel programming is primarily concerned with performance, a good project must also provide a significant speedup.

8. Plagiarism. The student is explicitly warned that plagiarism is very serious, and is taken very seriously. The use of external sources of any kind (internet, books, handouts, previous work by other students etc.) is only allowed for marginal contributions in the project (e.g. for the initial description of the assigned project), and each individual occurrence must be explicitly cited and reported in the report. Violation of this code of conduct will not be tolerated in any way, and will result in the immediate cancellation of the examination and the referral of the student to the University Disciplinary Committee, with the commencement of the relevant disciplinary proceedings in accordance with the provisions of Title V (DISCIPLINARY PROCEDIMENTS AND SANCTIONS) of the University Regulations for Students (D.R. 672 of 5/12/2017, which came into force on 8/12/2017).

OTHER USEFUL INFORMATION

Office Hours

By appointment; contact the instructor by email or at the end of class meetings.

FULL SYLLABUS

Parallel Algorithms

Introduction. Parallel Architectures. Parallel algorithm design. Message-Passing Programming. Sieve of Erathostenes. Floyd all-pairs shortest path algorithm. Performance analysis. Matrix-vector multiplication. Document classification. Matrix multiplication. Linear Systems. Finite Difference Methods. Sorting.

Parallel Programming

Message-Passing programming using MPI.

Sequential A*lgorithms*

Introduction. Order of growth. Analysis of algorithms. Decrease and conquer. Divide and conquer. Recurrences. Randomized algorithms. Transform and conquer. Dynamic programming. Greedy algorithms. Single Source Shortest Paths. Dijkstra algorithm. Breadth-First Search. Bellman-Ford Algorithm. Complexity and computability. NP-Completeness. The transition from sequential to parallel computing. Parallel complexity.

REFERENCE TEXT BOOKS

Introduction to Algorithms. Fourth edition. Cormen, Leiserson, Rivest, Stein. The MIT Press
Parallel Programming in C with MPI and OpenMP (International Edition). Michael J. Quinn. McGraw-Hill