Computer Architecture

Academic Year: (2019 / 2020) **Review date: 29/04/2019 09:03:43**

Department assigned to the subject: Computer Science and Engineering Department

Coordinating teacher: GARCIA SANCHEZ, JOSE DANIEL

Type: Compulsory ECTS Credits : 6.0

Year : 3 Semester : 1

Branch of knowledge: Engineering and Architecture

REQUIREMENTS (SUBJECTS THAT ARE ASSUMED TO BE KNOWN)

+ Programming
+ Computer structure.
+ Operating Systems.

OBJECTIVES

The goal of this course is to achieve that students know the basic concepts related to computer architecture and the impact that these concepts have on the performance of applications and computer systems.

To achieve this goal, the student will gain insight in the following competences:

* CECRI9: Ability to know, understand and evaluate computer architecture, as well as its basic components.

* CECRI14: Knowledge and application of fundamental principles and basic techniques for concurrent and parallel programming.

* CEIC3: Ability to analyze and evaluate computer architectures, including parallel platforms, as well as to be able to develop and optimize software for those architectures.

DESCRIPTION OF CONTENTS: PROGRAMME

1. Fundamentals of computer design.
1.1. Introduction and fundamentals of computer design.
1.1.1. Introduction.
1.1.2. Historical perspective.
1.1.3. Classification of computers.
1.1.4. Parallelism.
1.1.5. Computer Architecture.

2. Performance evaluation of computer systems.
2.1. Architecture classification and evaluation.
2.1.1. Technology trends.
2.1.2. Power and energy trends.
2.1.3. Cost trends.
2.1.4. Performance evaluation and metrics.
2.1.5. Benchmarks.
2.1.6. Amdahl's Law.
2.1.7. Processor performance.
2.2. Reliability of computer systems
2.2.1. Introduction to reliability.
2.2.2. Difference between reliability and availability.

3. Instruction level parallelism.
3.1. Introduction to instruction level parallelism.
3.1.1. Introduction to pipelined architectures.
3.1.2. Hazards: structural, data, and control.
3.1.3. Branch prediction.
3.1.4. Multi-cycle operations.
3.2. Exploitation of instruction level parallelism.
3.2.1. Compilation techniques and ILP.
3.2.2. Branch prediction advanced techniques.
3.2.3. Introduction to dynamic scheduling.
3.2.4. Speculation.
3.2.5. Multiple issue techniques.
3.2.6. Limits of ILP.
3.2.7. Thread level parallelism.

4. Memory hierarchy.
4.1. Cache memory optimizations.
4.1.1. Principles of cache memory design.
4.1.2. Policies and strategies in cache memory.
4.1.3. Basic optimizations in cache memory.
4.2. Advanced optimizations in cache memory.
4.2.1. Simple and small caches.
4.2.2. Way prediction.
4.2.3. Pipelined access.
4.2.4. Non-blocking caches.
4.2.5. Multi-bank caches.
4.2.6. Critical word first and early restart.
4.2.7. Write buffer merge.
4.2.8. Compiler optimizations.
4.2.9. Hardware prefetching.
4.3. Virtual memory and virtual machines.
4.3.1. Virtual memory.
4.3.2. Virtual memory policies.
4.3.3. Page tables.
4.3.4. Virtual machines.
4.3.5. VMM: Virtual machine monitors.
4.3.6. Virtualization hardware support.
4.3.7. Virtualization technologies.

5. Multiprocessors
5.1. Symmetric shared memory architectures.
5.1.1. Introduction to multiprocessor architectures.
5.1.2. Centralized shared memory architectures.
5.1.3. Cache coherence alternatives.
5.1.4. Snooping protocols.
5.1.5. Performance in SMPs.
5.2. Distributed shared memory.
5.2.1. Introduction to distributed shared memory.
5.2.2. Bases of directory based protocols.
5.2.3. Implementation of directory based protocols.
5.3. Synchronization in shared memory systems.
5.3.1. Introduction to synchronization.
5.3.2. Hardware primitives.
5.3.3. Locks.
5.3.4. Barriers.
5.4. Consistency memory models.
5.4.1. Memory models.
5.4.2. Sequential consistency.
5.4.3. Other consistency models.
5.4.4. Use case: Intel processors.

6. Parallel and concurrent programming models.
6.1. Introduction to parallel programming with OpenMP

6.1.1. Threads in OpenMP.
6.1.2. Syncrhonization.
6.1.3. Parallel loops.
6.1.4. Synchronization with master.
6.1.5. Data sharing.
6.1.6. Sections and scheduling.
6.2. Concurrent programming model: Threads in ISO C++11/14.
6.2.1. Introduction to concurrency model in C++.
6.2.2. Threads.
6.2.3. Shared data access.
6.2.4. Waiting.
6.2.5. Asynchronous tasks.
6.2.6. Mutex objects and condition variables.
6.3. Non-sequential consistency in C++11.
6.3.1. C++ memory model.
6.3.2. Atomic types.
6.3.3. Ordering relations.
6.3.4. Consistency models.
6.3.5. Barriers.
6.4. Lock-free programming.
6.4.1. Lock-free data structures.
6.4.2. Static lock-free data structures.
6.4.3. Dynamic lock-free data structures.

LEARNING ACTIVITIES AND METHODOLOGY

1. Theoretical classes, where knowledge to be acquired by students is presented. To ease its progress, students will be provided with references to basic texts. Those materials will allow students to get deeper understanding of topics they are more interested in.
* Emphasis will be put in the impact of computer systems performance in different application domains and its impact on organizations.
* Included examples will emphasize the acknowledgment of the rapid evolution of computer architectures and the need for longlife learning to keep being an active professional.

2. Lab projects, where students will analyze and develop diverse assignments using concepts from the course. Those projects will be carried out by teams, to foster the ability to do team work.

* Depending on the case students will be required to write a report on each project or to complete an evaluation test.
* Labs may include, among others, the performance evaluation of a computer system, the evaluation of the instruction level parallelism impact on performance, the evaluation of the design parameters in a cache memory system, the study of the impact of consistency memory models (including relaxed models) in applications performance, as well as the study of the impact of parallel programming models.

3. Exercise resolutions done by lecturers to allow a better understanding of course concepts given in theoretical classes.

4. Exercise resolutions done by students to allow student autoevaluation and the acquisition of needed competences.

5. Exams.

ASSESSMENT SYSTEM

% end-of-term-examination/test:	50
% of continuous assessment (assigments, laboratory, practicals…):	50

50% of final score is obtained by a final exam to evaluate acquired knowledge.

Remaining 50% of score is obtained as a result of a continuous evaluation process.

Continuous evaluation process includes:

* Evaluation tests on knowledge acquisition during the course giving 15% of final score.
* Labs during the term giving 35% of the final score.

% end-of-term-examination/test:	50
% of continuous assessment (assigments, laboratory, practicals…):	50

FINAL EXAM IN ORDINARY ROUND

In ordinary round, students will perform a final exam including all the course contents, both theoretical and practical. For those students who followed the continuous evaluation process final exam will weight 50% of final score. The remaining 50% will be obtained through the continuous evaluation process.

A student has followed the continuous evaluation process if:

* Has obtained a minimum of 2 points over 10 in each project.
* Has obtained a weighted average for all labs of 4.5 points over 10.

To pass the course, the final exam is mandatory for every student. Accreditation of a minimum performance in the final exam is needed obtaining at least 4 points over 10. Otherwise, the student will have not passed the subject and the average mark will not be computed.

Final mark will be incremented in 1 point to those students who have done all the continuous evaluation, have obtained 7 over 10 points in continuous evaluation and at least 4 points over 10 in the final exam.

For those students not having completed the continuous evaluation process, final exam will have a value of 60% of final scoring for the course. Thus, to be able to pass, the student will need to get a scoring higher than 8.33 over 10 in this exam.

Students will know, before starting exam period, the scoring obtained in the continuous evaluation process.

EXTRAORDINARY ROUND

Students not passing in the ordinary round will carry out a new exam in the extraordinary round. The scoring in extraordinary round will be performed according to the following rules:

a) If the student completed the continuous evaluation process, the final exam for this round will weight 50% and the other 50% will be obtained from scoring in the continuous evaluation process, provided that the mark in the exam is at least 4 points over 10.

b) If student did not complete the continuous evaluation process, the final exam for this round will weight 100% of final scoring.

c) When the student has completed the continuous evaluation process, the final exam for this round will weight 100% if that criteria gives a higher mark.

BASIC BIBLIOGRAPHY

- Hennessy, JL y Patterson, DA. Computer Architecture: A Quantitative Approach. 6th Edition., Morgan Kaufmann,, 2017

ADDITIONAL BIBLIOGRAPHY

- Barlas, G. Multicore and GPU Programming, Morgan Kaufmann, 2014

- Mattson, TG, Sanders, BA y Massingill, BL. Patterns for Parallel Programming., Addison-Wesley., 2004

- Pacheco, P. An introduction to parallel programming, Morgan Kaufmann, 2011

- Stallings, W. Computer Organization and Architecture. 9th Edition., Addison-Wesley., 2012

- Williams, A. C++ Concurrency in Action. Practical Multithreading. 2nd Edition, Manning., 2018