جامعة الملك عبدالله
للعلوم والتقنية
King Abdullah University of
Science and Technology

Syllabus: System Architecture & Performance

Division	**CEMSE**
Course Number	**CS 258**
Course Title	**System Architecture and Performance**
Instructor	
Name	**Elmootazbellah N. Elnozahy**
Phone	**544 700 171**
Email	**mootaz@kaust.edu.sa**
Prerequisite Course Number:	At least one undergraduate class that covers material taught in CS 205, CS 207, and ECE 201
Comprehensive Course Description	This course has the primary goal to provide the students with the knowledge of computing system aspects as they relate to performance, with a secondary goal of developing skills on how to evaluate performance and design hardware/software systems/perform that perform efficiently. The first part of the course reviews system architecture with focus on aspects that impact performance directly. These aspects include but not limited to: pipelining, cache memory design, virtual memory support, interconnect topologies, cloud computing support, virtualization, modern storage systems, and accelerators. The second part of the course covers low-level software aspects of systems, including runtime systems, programming language issues, power management, and general parallelization techniques . The third part covers performance evaluation techniques including spreadsheet models, experimental evaluation, simulation, and analytic modeling. Topics covered: Part 1 - Processor architecture: Pipelining, bubbles, speed demons, super-scalar design, out-of-order execution and speculative execution. - Memory architecture: Cache architecture, load-store units, virtual memory support, high-bandwidth memory, and hybrid memory systems. - Interconnect architecture: Coherence and consistency management, latency and bandwidth issues. - Storage systems: SSD systems, logging, and hierarchical storage. - Cloud computing support: Nested page tables, deduplication support, and hypervisor systems. - Parallel processing: Multicore systems, multithreaded processor architecture, and accelerators. - Power management: Voltage and frequency management and impact on programming performance.

	Part 2 - Runtime support: Virtual machine concepts applied to programming languages, just-in-time compilation, garbage collection, and performance issues related to programming languages. - Parallel programming: General and quick introduction to parallelization, including design choices concerning the right architecture model for a particular application and how to efficiently program it. Part 3: - Fundamental theory of performance analysis: Amdahl's law, Little's law, introduction to queueing theory. - Spreadsheet models: How to quickly build performance models and then perform step-wise refinement as an iterative performance engineering. - Experimental evaluation: How to set up a testbed, confidence intervals, repeatability, documentation, and sensitivity analysis. - Simulation: Monte-Carlo simulation concepts, random variables, event-based, vs. time-based, vs. process-based simulation, and how to validate simulation studies. - Analytic Modeling: General introduction to analytic model and understanding of why it failed in practice. - General approach to performance evaluation: Profiling, measurements, and how to combine different techniques to achieve better performance.
Goals and Objectives:	Whether writing the next computational fluid dynamics program, a neural network solver or constructing a website, programmers are usually well trained in the specific domain that they are solving, although they tend to rely on hardware as an abstract black box to provide acceptable performance. This approach leads to resource inefficiency, heightened costs, and in industry, difficulty in career advancement! Whether planning to continue research toward a Ph.D., a career in academic research, or a jump to industry after graduation, students will benefit from this course in unmasking some of the black-box approach to yield better code and hopefully more successful careers. This course is thus designed for entry-level graduate students who will be using computing to support their research in the future, or those who are planning a career in advanced development in industry. The course aims to help the students acquire the following skills: - Understanding of system architecture, including the interactions of software and hardware, and impact on the codes they write. - Capacity planning of systems solutions to meet performance requirements. - How to identify performance bottlenecks in the code they write, including knowledge of profiling and performance debugging techniques. - Performance evaluation techniques and when to use each. - How to write faster, more efficient code. - Some power management techniques.

	The course should be suitable to students enrolled in any program that relies on computing to drive research, such as computer science, computational sciences in applied mathematics, and students interested in using high-performance computing. It should also be a nice complement to electrical and computer engineering students studying in the CE track in ECE.
Required Knowledge	**Good programming skills, knowledge and comfort in at least one of the following: C, C++, Python or Rust.**
Reference Texts:	**Computer Architecture: A Quantitative Approach, 5th Edition, by Hennessy and Patterson.** **A variety of papers that will be distributed by instructor.** **Instructor notes will be made available to students on an on-going basis.**
Method of evaluation	**10 Homework problem sets: 30%** **First term examination: 15%** **Second term examination: 15%** **Third term examination: 15%** **Project: 25%**
Nature of the assignments	Homework problems will be a combination of paper/pencil answers to questions or small programming assignments within the scope of 1 to 2 weeks each. Examinations will be in class exams lasting 60 minutes each. Project will be released in the second half of the class. There is no final exam.
Course Policies	Students are encouraged to seek help from instructor. Group working on homework problems is encouraged. Using resources from the Internet is permitted with proper attributions. Two sure ways to get an F in the class is to 1) cheat, or 2) get in a group to free-load.
Additional Information:	Grading is done on an absolute scale, not a curve. This is a course that aim to teach practical skills, so some effort is expected.

NOTE
The instructor reserves the right to make changes to this syllabus as necessary.

Tentative Course Schedule:

(Time, topic/emphasis & resources)

Week/Lecture	Topic
1	Introductions, class organization, student assessment, and a high-level presentation of the course material
2	Processor architecture review: The instruction lifecycle, pipelining, models of execution, and design choices. Impact on program performance.

3	Memory architecture issues: The latency vs. bandwidth problems. Physical limitations. The processor side: Load/store units and pre-fetching. The memory side: Caching and address translation.
4	Storage systems: Logging, SSD, physical limitations. Space vs. time efficiencies. Interconnect systems: Topologies, system scalability issues, coherence, synchronization.
5	Cloud computing support: Performance at large. Performance of cloud systems. Architectural support: Deduplication, nested page tables, hypervisor systems. Power management
6	Putting it together: A system approach to understanding performance. Impact on user programs. First term exam.
7,8	Parallel processing: From processor to system levels: Instruction-level parallelization, VLIW (and why it failed), data flow models. Modern concepts: Multithreaded architecture, and multicore systems. Data exchanges via memory vs. messages (RDMA, message passing, Open/MP). Vector and accelerators.
9	Low-level software issues: Runtime support, executing code for modern programming languages in virtual machine mode, just-in-time compiling, garbage collection and automatic memory management.
10	Project release. Buffer lecture. Second term exam.
11	Performance evaluation: Fundamental theories. Spreadsheet models. Repeatability. Confidence intervals. Variability and how it can be dealt with. Introduction to analytic modeling.
12	Experimental evaluation. How to design experiments. Testbeds. Measurements. Profiling. Tools.
13	Simulation: Various types of simulation (time, event, process). Generation of input. Interpretation of output. Validation.
14	Putting it together: Hands on session on performance debugging. Buffer lecture.
15	Project review. Third term exam.