I. Course Overview

Advanced Computer Architecture (ACA) covers advanced topics in computer architecture focusing on multicore, graphics-processor unit (GPU), and heterogeneous SOC multiprocessor architectures and their implementation issues (architect's perspective). A range of levels are explored from deep submicron CMOS characteristics, microarchitecture, compiler optimization, parallel programming, run-time optimization, performance analysis & tuning, fault tolerance, and power-aware computing techniques.

The objective of the course is to provide in-depth coverage of current and emerging trends in computer architecture focusing on performance and the hardware/software interface. The course emphasis is on analyzing fundamental issues in architecture design and their impact on application performance. To enable a better understanding of the concepts, hands-on assignments are used to explore issues in multicore and GPU architecture systems. Students have options in exploring their own interests in custom projects and assignments.

New recorded video lectures in Spring 2017

New projects in Spring 2017: Students work in groups of up to two people, for projects related to acceleration and performance tuning of machine learning, computer vision, and deep learning. Students taking the course can investigate projects with access to NVIDIA, Xilinx, and Raspberry Pi resources:

NVIDIA Jetson TX1 ([http://www.nvidia.com/object/jetson-tx1-module.html](http://www.nvidia.com/object/jetson-tx1-module.html)) is the world’s leading AI computing platform for GPU-accelerated parallel processing in the mobile embedded systems market. Its high-performance, low-energy computing for deep learning and computer vision makes Jetson the ideal solution for compute-intensive embedded projects. Jetson TX1 is a supercomputer on a module that's the size of a credit card. It features the new NVIDIA Maxwell™ architecture: GPU 1 TFLOP/s 256-cores, with CPU 64-bit ARM® A57 CPUs Memory 4 GB LPDDR4 | 25.6 GB/s

- Project potential: Drones & Unmanned Aerial Vehicles (UAVs), Autonomous Robotic Systems, Mobile Medical Imaging, Intelligent Video Analytics (IVA)
PYNQ- Python Productivity for Xilinx Zynq Programmable Hardware

http://www.pynq.io/

PYNQ is an open-source project from Xilinx that makes it easy to design embedded systems with Zynq All Programmable Systems on Chips (APSoCs). Using the Python language and libraries, designers can exploit the benefits of programmable logic and microprocessors in Zynq to build more capable and exciting embedded systems. Using the Python language and libraries, designers can exploit the benefits of programmable logic and microprocessors in Zynq to build more capable and exciting embedded systems.

PYNQ users can now create high performance embedded applications with
- parallel hardware execution
- hardware accelerated algorithms
- real-time signal processing
- high bandwidth IO and low latency control

Zynq-7000 All Programmable SoC Features
- Dual ARM® Cortex™-A9 MPCore™ with CoreSight™
- 32 KB Instruction, 32 KB Data per processor L1 Cache
- 512 KB unified L2 Cache, 256 KB On-Chip Memory, 630 KB of fast block RAM
- 85K logic cells (13300 logic slices, each with four 6-input LUTs and 8 flip-flops)

- Low Transistor Count
- Low Power Consumption/Heat Production
- Used in most mobile devices: phones and small digital devices
- Raspberry Pi has similar requirements to mobile devices
II. Course Prerequisites
This course requires the understanding of design of processors, specifically computer organization and the instruction set architecture (ISA): ECEN 4593 (Computer Organization) or an equivalent first course in computer organization and design. Students should already understand some computer instruction set and know how to design a control unit, arithmetic unit, memory (cache and virtual), and various input/output interfaces.

III. Course Outline
1. Introduction to Computer Design and Quantitative Principles of Architecture Performance Analysis
   • Technology and computer trends
   • Measuring computer system performance
   • Benchmarks and metrics
2. Instruction Set Principles and Examples
   • Classification of Instruction Set Architectures (ISA) – RISC, CISC, VLIW, EPIC
   • Predicated execution and compiler-controlled speculation
3. Advanced Microarchitecture and Instruction-Level Parallelism
   • Superscalar and pipeline operation
   • Instruction-Level Parallelism (ILP)
   • Dynamic instruction scheduling (Tomasulo, scoreboard, reservation station design)
   • Overcoming control hazard - branch prediction (2-bit, two-level)
   • Compiler optimization and analysis
4. Memory-Hierarchy Design
   • Multi-level cache design issues
   • Performance evaluation
   • Memory prefetching techniques
5. Thread-Level Parallelism
   • Multicore systems
   • Thread control models (fine-grained, coarse-grained, hyper-threading)
6. Data-Level Parallelism
   • Vector processing
   • Graphics Processing Units (GPU)
   • NVIDIA architecture models – Fermi, Tesla, Kepler, Maxwell, Pascal
   • CUDA/OpenCL programming
7. Performance-tuning and Analysis of Modern Applications
   • Run-time optimization
   • Binary instrumentation
   • Hardware performance monitoring
   • Performance tuning
8. Architecture Implementation Issues and Analysis
   • Power- Dynamic Voltage Frequency Scaling (DVFS), Energy-Delay Product (EDP)
   • Architecture physical layer concepts including device&layout, manufacturing constraints, architectures, defect tolerance, and design variability.
Course Schedule

WEEK 1 - Introduction, Instruction Set Architecture, and Pipelines

Topics:
- Description of architecture, micro-architecture and instruction set architectures.
- Pipelining Review - basic concept of pipeline and two different types of hazards.
- Pipeline CPI
- Processor Pipeline Hazards
- Computer Architecture & Tech Trends
- Processor Speed, Cost, Power
- Measuring Performance
- Benchmarks Standards
- Iron Law of Performance
- Moore’s Law
- Amdahl’s Law
- Lhadma’s Law
- Gustafson’s law

WEEK 2 - Control Hazards

Topics:
- Misprediction Penalties
- Branch Prediction Techniques
- Two-level Correlation Predictors: PAg, GAg
- Hybrid Predictors
- Return Address Stack
- Loop Prediction
- Understanding Code Execution and Coding Practices for Branch Prediction

WEEK 3 and WEEK 4 – Base Cache Memory, Dynamic Execution and Superscalar Model

Topics:
- Cache memory characteristics
- Instruction Level Parallelism (ILP)
- Out-of-order execution- common methods used to improve the performance of out-of-order processors including register renaming and memory disambiguation.
- Common issues for superscalar architecture.
- Kinds of architectures for out-of-order processors.

WEEK 5 and WEEK 6 – VLIW, EPIC, and ILP Compiler Optimizations for Architectures

Topics:
- Traditional Compiler Optimization: Peephole, Loop Unrolling, Inter-procedural, and Inlining
- Compiler Optimization for Instruction Level Parallelism (ILP) and Profile-Directed Techniques
- Out-of-order execution- common methods used to improve the performance of out-of-order processors including register renaming and memory disambiguation.
**WEEK 7 - Multicore Architectures and Vector/Multimedia Instruction Sets**

**Topics:**
- Simultaneous multithreaded (SMT) architectures
- SMT Architecture Alternatives
- SMT architecture: OS impact and adaptive architectures
- Multi-core Architectures
- Single Instruction Multiple Data (SIMD)
- Intel Architecture Development: MMX, SSE
- Inline Assembly and Assembly Intrinsics

**WEEK 8 thru WEEK 13– Graphics Processing Unit (GPU) Architecture**

**Topics:**
- NVIDIA CUDA/GPU Programming Model
- GPU Hardware and Parallel Communication
- GPU Fundamental Parallel Algorithms
- Optimizing GPU Programs
- The Frontiers and Future of GPU Computing
- OpenCL – Open Compute Language
- Mobile GPU System Architecture Exploration: NVIDIA TX1

**WEEK 14– Runtime Optimization and Compilation**

**Topics:**
- Dynamic compilation and Code Translations

---

**IV. Learning Outcomes**

A student who has successfully completed this course should be able to:

1. Analyze various performance characteristics of a computer system.
2. Apply digital design techniques to the microarchitecture construction of a processor.
3. Translate assembly language programs to/from high-level language codes and algorithms.
4. Analyze hardware & software trade-offs to design the instruction set architecture (ISA) interface.
5. Understand advanced issues in design of computer processors, caches, and memory.
6. Analyze performance trade-offs in computer design.
7. Apply knowledge of processor design to improve performance in algorithms and software systems.
8. Acquire experience with tools for statistical analysis of instruction set trade-offs.
9. Gain the ability to develop parallel GPGPU solutions of CUDA and OpenCL

**V. Required Text and Materials**


**VI. Assessment & Assignments**

**Assignments.** The following programming assignments are scheduled:

- **Pin** – Binary instrumentation tool to analyze program behaviors
  - Choice of branch prediction or cache design simulation.
- CUDA programming - Vector addition
- CUDA programming - Histogram generation
- CUDA programming - Image filtering
**Reading Assignments:** There are several technical papers (conference proceedings, journal articles, and technical reports) assigned through the semester. Reading technical papers in the field of computer architecture is imperative to understanding future directions in the field. Assignments will require students to write brief overviews or answer technical questions about the papers assigned. Subject matter from the reading assignments are likely to be covered in exams.

**Final Exam:** There will a take-home final exam that covers the concepts of the course. The exam problems are closely related to the lectures, homework assignments, and assigned readings. The final exam will be cumulative, covering all subject topics.

**Final Project:** There will be a project for you to work on as an individual or in a group of two people. The project will count as 15% of your grade, and will be a significant amount of work. The assignment is to extend the semester projector to analyze some interesting data or new architecture feature. Students are able to write survey papers as a second option to the project. The project will be divided into several milestones, one checkpoint being a presentation of work. Details about the project and schedule will be announced later in the semester.

**Basis for Final Grade**
Student's grades will be assessed based on their completed homework, quizzes, project, in-class exams, and the final exam. Homework assignments are designed to provide active learning for the student by exercising the various topics covered by the course. Exams will be designed to assess the student's ability to master the different topic areas, and their aptitude in each of the learning outcomes. The percentage given to each assessment method is given by Table 1.

<table>
<thead>
<tr>
<th>Assessment</th>
<th>% of Final Grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reading Assignments</td>
<td>10 %</td>
</tr>
<tr>
<td>Assignments &amp; Checkpoints</td>
<td>40 %</td>
</tr>
<tr>
<td>Project</td>
<td>20 %</td>
</tr>
<tr>
<td>Final Exam</td>
<td>30 %</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>100 %</strong></td>
</tr>
</tbody>
</table>

**Course Policies**

**Late Work Policy:** Homework assignments must be turned in at the beginning of class, else it will be considered late. A student's score will be reduced by a 20% penalty for submitting work, one second to 24 hours late.

**Student Honor Code:** Students should be familiar with the College of Engineering and Applied Sciences student honor code. All honor code rules will be adhered to in this class.

**Appointments:** Students are encouraged to make at least one appointment with the professor during the semester. Appointments can be made by email. Students are encouraged to explore research opportunities, expressing concerns, offering suggestions, and seeking advice are among the welcome topics.