Teaching Statement

Joseph Devietti, University of Pennsylvania

Since arriving at Penn, I have worked to improve the quality of computer engineering education by updating the undergraduate and graduate computer architecture curriculum. I have also designed and taught a series of new graduate-level seminars. Outside the classroom, I have worked to create a vibrant and diverse computer architecture community through teaching-focused outreach activities.

CIS 371: Undergraduate Computer Architecture

CIS 371 at Penn was *the* course that, when I took it as an undergraduate, convinced me to pursue a career in computer architecture. I have tried to recreate that seminal experience for students in my version of the class, while also introducing more digital design rigor. In this class, students design a 5-stage pipelined, 2-way superscalar processor in Verilog to execute a simple RISC-like assembly language called LC3[1]. As this is a required class in our undergraduate computer science curriculum, most students will go on to careers in software instead of hardware. From the first day, I describe the value of taking this class as two-fold: first, hardware pipelines do not admit many useful internal abstractions so students must grapple with the complexity of the entire pipeline to get it working; second, hardware design offers myriad opportunities to hone debugging skills.

Historically, the focus in CIS 371 has been exclusively on correctness, not performance. However, designing hardware must always consider performance (with power and area as first-class concerns as well). I now walk students through timing analysis of their designs, ensuring that they reach timing closure by relaxing the clock if needed. I've also worked to remove some of the bottlenecks in the assignments that led to highly unintuitive results, e.g., moving from a single-cycle to a 5-stage pipeline design originally resulted in a ~5% frequency boost due to severe stage imbalance. Finally, I've added more interactive elements to the class so that students can see their design running on an FPGA board and manipulate it via switches and LEDs.

CIS 501: Graduate Computer Architecture

During my time at Penn, I have increased the rigor in the graduate computer architecture course. Over my first few years with the course, I revamped the homework assignments to task students with building a detailed uniprocessor pipeline model, with caches and branch prediction, over a series of cumulative assignments. Crucially, the students are required to integrate the different components into a larger design, which gives them a better overview of how the entire processor works.

Starting this year I have revised the assignments again to have the students build a processor in Verilog, mirroring the assignments in the undergraduate version of the class. This removed a peculiarity in our curriculum where undergraduate architecture was more rigorous than the graduate version.

[1] Yale N. Patt and Sanjay J. Patel. *Introduction to Computing Systems: From Bits and Gates to C and Beyond*. 2nd Edition. McGraw-Hill Education.

Several students who have taken CIS 501 have gone on to take an advanced architecture seminar with me and to pursue research in my group, which has led to two published papers thus far. Two other 501 students have gone on to computer architecture jobs in industry, both working for Oracle on chip design, and have mentioned that CIS 501 was valuable to advancing their careers.

New Seminar Courses

I have taught a number of new seminar courses at Penn, helping to end a drought of computer architecture seminar courses extending back three years before my arrival. I have taught seminars on multicore programmability, security issues in multicore architectures such as side-channel attacks, and twice a seminar on GPGPU programming. My GPGPU course teaches students Nvidia's CUDA language and programming model for GPUs. I place a special emphasis on the correctness and performance bugs that are possible with GPU code, and the work done in this seminar has had a tight connection with my work on detecting safety and performance bugs in GPU code. One of the class projects even resulted in a new race detector for GPU code, which was later published in PLDI 2017. Students regularly approach me to ask about the next iteration of this course, as many students across engineering are using GPUs for general-purpose computing in both research and their post-graduation careers.

Teaching-Centric Department Service: CIS Minicourses

I serve as one of two faculty coordinators of our department's "minicourses", which are 1-hour-per-week courses on a range of practical topics like iOS programming and JavaScript. Minicourses in our department are an enormously popular way for students to gain exposure to relevant technologies in a formal classroom setting. Total enrollment across the minicourses regularly approaches 200 students per semester. A unique feature of these minicourses in our department is that they are taught by graduate or undergraduate students. Graduate student instructors gain valuable preparation for a pedagogical career – several former minicourse instructors have gone on to teaching faculty positions at institutions including Grinnell College, Bryn Mawr College, and the US Naval Academy. Undergraduate instructors are invariably prior minicourse enrollees and treasure the opportunity to train "the next generation" of industry developers. As most minicourse instructors have never taught before, we host monthly meetups for all instructors to discuss pedagogy and share best practices.

Teaching-Centric Outreach Efforts

To foster participation in computer science beyond my institution I have volunteered at our department's annual Women in Computer Science High School Day for Girls for the past five years, a full-day outreach event that brings over 100 local female high school students to Penn to learn what computer science is and how rewarding a computer science career can be. My contribution is a 1-hour lesson on "thinking in parallel". I teach 20-30 students about data parallelism and pipeline parallelism through interactive demonstrations such as counting a large collection of Starburst candies and making origami frogs in an assembly line. While forms of outreach like this are necessarily longerterm in impact, I have found these activities to be personally both rewarding and rejuvenating, and I intend to continue my participation going forward.

To help make core computer architecture concepts like branch prediction and caching more accessible, I have built a series of web-based visualizations to demonstrate how these structures operate. These visualizations are available at http://comparchviz.com. I have found that explaining these topics via whiteboard or PowerPoint slides provides space for only a limited set of examples – e.g., one or two branches flowing through a branch direction predictor and updating its internal state. Other cases can only be discussed briefly. With the visualizations I've built, however, students can supply their own inputs to see how the machinery responds – the visualizations are fully "executable". This allows students to explore a wider range of cases with relative ease, deepening their understanding. I first used these visualizations in my Spring 2019 architecture class, and received informal positive feedback from students and TAs in the course. Going forward, in addition to building out the library of visualizations, I plan to evaluate their effectiveness more formally in collaboration with teaching faculty in my department, to see how these visualizations can be improved further.