Teaching Statement

Scott Beamer

My enthusiasm for teaching is one of my main motivations for seeking a faculty position. I have found teaching to be rewarding, so much so that I have taught extra times and pursued a teaching minor. I am committed to ensuring my classroom is an inclusive environment, and supportive of all students, whether they have different prior computing experiences, different learning styles, or come from diverse backgrounds.

Experience
I have taught in many different roles, which in turn has given me insight into the teaching process as a whole. My teaching experiences at UC Berkeley came from two courses: CS 61C – Great Ideas in Computer Architecture (lower-division architecture and systems) and CS 152 – Computer Architecture and Engineering (upper-division architecture). The summer before I started my graduate studies, I was the instructor for CS 61C, and I had complete responsibility for the couse of over 100 students. I lectured for 6 hours per week and worked with 3 TAs teaching 3 sections. I earned that opportunity based on teaching well as an undergraduate, both as a lab TA and a head TA. As a graduate student, I continued to teach as a TA, a head TA, and a guest lecturer. In addition to teaching, I worked to support my department's instructional efforts. I was on the curriculum redesign committee to overhaul CS 61C's content. For three years, I also taught a one day workshop to first-time Computer Science TAs to prepare them for the semester.

To obtain these teaching experiences, I not only had the benefit of great colleagues and mentors, but I also took formal training via three semester-long courses along the way. In addition to the required teaching course intended for TAs, I also took an advanced course intended for career instructors which included designing a mock course. Following that, I took a course from the Education department on how to best evaluate and integrate educational technologies.

Philosophy
My teaching style is built on a foundation of preparation, enthusiasm, and a deep knowledge of the subject matter. I am strong believer in letting students learn hands-on as soon as possible. Computers have become ubiquitous, so there is no reason why every student can not be simultaneously learning by doing, experimenting, and exploring. Furthermore, by enabling students to create things, we can tap into their motivations to encourage self-directed learning.

To support different learning styles, I make sure both my instruction and assessments use a variety of methods. To make discussion sections more interactive, I include activities that require students to move around the room and work together. For example, to practice encoding numbers in binary, I had each student represent a bit, and stand or sit to represent whether it was one or zero. After giving a group a decimal number, they had to work together to figure out who should stand or sit. Another example is a jigsaw activity I created to help students appreciate how the parts of a processor work internally and fit together. Using all of the whiteboards in the room, the students drew schematics for a simple processor they designed. Each group was assigned a different part of the processor, and they had to work with other groups to ensure their components were compatible and connected. It was gratifying to see their satisfaction with covering the walls with a complete processor design.

One of my goals as an instructor is to reduce the amount of memorization my students need to do. One reason computer science is a great field is that most of it is derivable, so to me, asking students to memorize is often an indication the instructor has not sufficiently explained why something is the way it is. For example, at the end of semester, I guide the students through a full period discussion contrasting architectures. Every architecture we cover in the course has some advantages, so an integral part of this discussion is guiding them in making more specific metrics to expose each architecture's tradeoffs. This experience helps students remember all of the little mechanisms for each architecture because they understand how each architecture compares to others. More generally, this sort of comparison process is a useful design skill.

Another one of my goals as an instructor is to be constantly improving the course as well as my own teaching abilities. When I was a head TA as an undergraduate, I wrote an extensive guide for the next head TA, which included details of running the course, specifics on using the instructional infrastructure, and what to improve in the next offering. It was great to find that same document updated and expanded by the subsequent staff the next time I was head TA for the course, and the course had improved.

I have been able to infuse my research into my teaching to increase student engagement by giving them hands-on experience with real-world applications for what they are learning. A highlight was a MapReduce project I created, which was not only a more interesting problem to solve than prior course offerings, but it also exposed students to social network analysis. For the project, students implemented BFS with MapReduce, which allowed them to compute distances between vertices in a graph. By giving them access to interesting real-world graphs from Wikipedia and Twitter, they were able to observe the small-world property (6-degrees of separation) in practice. I even gave them access to a graph representing all movie actors' co-starring relationships, and with it, they could compute 6-degrees of Kevin Bacon (a trivia game).

To help balance out differences in prior computing experience, I added a "Tools of the Trade" segment to my discussion sections. Computer science is a rapidly changing field, so as instructors we often emphasize core concepts over practical skills that will be soon obsolete. However, there are some skills (e.g. command-line utilities) that are not necessary to understand the core concepts, but whose omission can be a source of uneven learning and frustration. Looking over students shoulders' as they worked, especially during labs, I saw how not knowing a simple command could greatly hinder their productivity. If students are able to eventually complete their tasks, they may not wonder if there is a better way to do them. During the segment of only a few minutes, I would briefly introduce a tool and discuss its potential uses. The intention was to make students aware of these tools so they could seek out more information on their own. Each week I chose the tool to either be an application of a higher level concept they were currently learning or immediately applicable to their current assignment. The benefit of this segment was most apparent when inexperienced students listed it as a valuable component of the section or inquired why I left it out when dedicating an entire section to midterm review.

Future Directions
Based on my teaching and research experiences, I feel well qualified to teach courses (undergraduate and graduate) in computer architecture, digital design, and parallel computing. I also enjoy teaching lower-division courses to get students started in and excited about the field.

In addition to a graduate seminar on my research topic of improving communication efficiency, I am interested in creating an upper-division course on performance programming. On the surface, the course teaches many useful optimization techniques such as parallelization, profiling, auto-tuning, and blocking. However, the focus on performance optimization will serve as a great vehicle to more deeply understand architecture and its interaction with software. To fully appreciate an optimization, one must simultaneously understand how the target processor is bottlenecked by software without the optimization and how the optimization sufficiently transforms software to ameliorate that bottleneck. By modifying software and being able to immediately observe changes in how the processor is exercised, the course's approach teaches architecture with a kinesthetic learning style. The most important thing students will get out of the course is a culture of measurement. Throughout the course, students will carefully measure performance and model performance limits to pick the most profitable optimization targets. More generally, this develops an extremely useful skill of using critical analysis to determine the maximal potential benefit from solving a problem.