I-Ting Angelina Lee

Teaching Statement

This statement consists of two parts. In the first part, I discuss thoughts on how one may incorporate parallelism into the curriculum and structure courses that teach parallel computing. In the second part, I describe what I learned from my past teaching experience, which also helped shape my teaching philosophy.

Incorporating Parallelism into the Curriculum

Since my research focuses on making parallel programming accessible for everyone, I care deeply about educating the next generation of computer scientists in the art of parallel programming. With the advent and widespread deployment of multicores, writing parallel programs has become a necessity, and an adequate computer science education must prepare students for this shift. I would like to teach classes that not only equip industry-bound students with skills to develop parallel programs, but also inspire the next generation of researchers to investigate ways to simplify parallel programming.

In particular, I would like to develop a graduate-level course, which exposes junior graduate students to the various aspects of parallel computing, from languages to architecture. I would cover various parallel programming models, such as dynamic multithreading, message passing, streaming, and CUDA, and discuss how their corresponding hardware influences the programming model. Given that multicore hardware is ubiquitous, I would cover topics related to parallel programming on multicores in more depth, and discuss synchronization, load balancing, race detection, cache coherence protocols, and memory models for shared memory. The class would consist of hands-on projects with a heavy emphasis on the final project, and the students would be encouraged to select a topic that advances their own research interests and turn their final projects into publishable papers.

For the undergraduate curriculum, even though the community has reached a consensus regarding the need for teaching parallelism, how to best incorporate it into the curriculum is still a highly debated topic. Some academics advocate introducing parallel thinking from the start: teach parallel computing as the default with sequential computing being a special case, and incorporate parallelism into all core classes. Arguably, this option may be ideal, but it is difficult to realize in the short term since it requires a substantial modification and restructuring of existing courses.

In the short term, I advocate taking an incremental two-tiered approach — offer a sophomore-level course that focuses on algorithms, but has a component on parallel algorithms; and in addition, offer a senior-level course that focuses on the actual engineering of a high-performing parallel software system. The sophomore-level course allows one to introduce parallel thinking early on, that is, thinking about how an algorithm can do multiple things in parallel instead of serially one at a time. The senior-level course allows the opportunity to discuss all of the low-level details involved in building high-performing parallel software, such as memory models, synchronization, and cache effect. Teaching students low-level details has equal importance to teaching parallel thinking with high-level abstractions. One must understand what hides underneath the abstractions to achieve performance, and after all, the main purpose of parallel programming is to obtain performance.

Given the opportunity, I would love to design and teach these classes. Of course, they would need to work within the current curriculum. If there were no constraints, however, I would structure them as follows. The sophomore class would cover a wide range of topics, much like an ordinary introductory data structures and algorithms class. For a given problem domain, however, the class would possibly cover several algorithms that solve the same problem, some of which naturally lend themselves to parallelization. I would bring students' attentions to the fact that designing a parallel algorithm poses a different set of considerations from designing a serial algorithm. The class would also cover models for analyzing parallel algorithms, such as work and span analysis, and discuss races at a high level. I believe in cultivating skills,

and thus the class would have a hands-on component that would require students to implement parallel algorithms with little or no need for synchronization using a high-level language, such as Cilk or NESL.

The senior-level course would emphasize both the knowledge and skills involved in building high-performing parallel software. The class would discuss what the compiler can or cannot do in the context of parallelism (e.g., vectorization); how hardware caching works and cache-oblivious algorithms, so that students would be aware of the performance benefit from effective cache utilization; and various parallel programming models, including low-level mechanisms such as pthreads, locks, and conditional variables, to demystify how a higher-level model is implemented.

Teaching Experience and Philosophy

My desire to mentor students, undergraduate and graduate, is central to my choice of an academic career. This desire was inspired by my own undergraduate and graduate mentors, in particular my advisor, Prof. Charles Leiserson. His support and encouragement have been instrumental to my exploration of my own path, scientifically and otherwise, and I hope to do the same for others.

I have immensely enjoyed my interactions with students. The experience was most rewarding not only when I was able to help improve students' understanding of the material, but also when working with me helped them better understand their own interests and abilities. I was doubly rewarded for learning with them and gaining new insights for myself as I teach. I realized this when I served as an undergraduate tutor for the first time, and this realization led me to continue working as a tutor for the rest of my undergraduate years. As a graduate student, I served as a TA for three different classes — *Mathematics for Computer Science*, *Introduction to Algorithms*, and *Structure and Interpretation of Computer Programs* — some of which are more theoretical to broaden my teaching experience.

A good teacher not only imparts knowledge, but also cultivates the development of skills necessary to put the knowledge into practice. I learned this from my advisor Charles when I co-taught a class with him on *Performance Engineering of Software Systems*.[1] This experience taught me ways in which students can be motivated, and in turn how I can be a better teacher. To encourage students to acquire the skills of performance engineering, the class assigned four major hands-on projects of increasing scope and complexity. Each project came with a baseline code which the students modified in any way they liked, from high-level algorithmic changes to low-level performance tuning, to speed up the code as much as they could. Each project had a beta and a final submission. We ranked all the beta submissions, so students could see how fast their code ran relative to others in the class. We then selected a performance goal based on the beta submissions, which the students were asked to achieve for the final submission. At the end of the term, we invited the teams who had the fastest code to share their strategies. Although at the beginning, I worried about the competitive nature of how the projects were graded, the competition seemed to fuel students' motivations, and teams that had high-performing code earned the respect of their peers.

The most powerful thing that an educator can do is to engender passion and excitement towards the subject, because an interested and motivated student will continue on an independent quest for knowledge. I am currently working with Kerry, a top student from our class. As part of the class, Kerry was introduced to and became interested in Cilk, a parallel language developed within my research group. After the class ended, he started working with me on a performance analysis tool for Cilk, which constitutes the basis of his Master's thesis. It has been a wonderful experience working with Kerry, who is enthusiastic and self-motivated. While I cannot claim credit for his enthusiasm towards the subject — he is naturally curious and inquisitive — I hope to engender the same kind of enthusiasm in my future students.

[1] The course materials can be found at `http://stellar.mit.edu/S/course/6/fa12/6.172/index.html`