

Lessons from Integrating Parallelism into Undergraduate Curriculum at UMKC

Mohammad Amin Kuhail
School of Computing and Engineering
University of Missouri-Kansas City
Kansas City, MO, US
e-mail: kuhailm@umkc.edu

Abstract—At the University of Missouri-Kansas City (UMKC), we successfully integrated parallelism into the undergraduate Data Structures course. In this paper, we share lessons learned from the teaching strategies that have successfully worked and the strategies that unfortunately did not. Our results are based on three years of continuous work on integrating parallelism into the course. We collected the students’ feedback using surveys. We refined our approach over several semesters to improve our teaching techniques.

Keywords: *Parallel programming, teaching, OpenMP, data structures, unplugged activity, live code, visualizations.*

For more than three years, we have been integrating parallel computing into the undergraduate Data Structures course using various teaching techniques: unplugged activities, visualizations, examples, live code, and active learning. The topics that were integrated include OpenMP architecture, threads, load balancing, for-loop parallelization, sections, and data synchronization. We started the integration in the fall semester 2015 after receiving the NSF/IEEE-TCPPE Early Adopters award. We introduced OpenMP briefly, and then showed the students the difference in how long it takes to run serial merge sort vs parallel merge sort, which was sorting 10 million integers.

Merge Sort Unplugged Activity: In the summer semester 2016, we experimented with an unplugged activity, in the form of a game to teach parallel computation. The activity used Merge Sort. Twenty seven students participated in an interactive activity where some students played integer by holding a number and other students acted like a processor and ordered the students. To simulate a real computing environment, each processor could move only one student at a time. The numbers of processors were increased each time from one to two to six with time being tracked in each round of sorting. Adding extra processors did not result in the improvement expected because many students forgot how Merge Sort worked. We collected the students’ feedback on the activity. In general, the students thought that the activity itself was engaging. However, it was not easy to follow at times, and some students thought that it was very time consuming. The activity lasted for about fifty minutes. In conclusion, we think such an activity should have been simpler (fewer rules and less expectations on students’ prior knowledge), shorter, and clearer (use bigger numbers and fewer processors).

Live Code: Another activity we implemented in class was giving the students tasks on basic parallelism, in the form of parallelizing serial code. A few students volunteered to code

the solutions live in front of the class. The students were thinking aloud as they were problem solving and coding. The students experienced syntactical and conceptual errors. They were corrected by other fellow students as well as the instructor. While this was an engaging activity for some students. Other students mentioned that it was time consuming and confusing at times, and it was more suited for a lab environment as opposed to a regular classroom setting. In conclusion, we think such an activity can be used in a lab provided that the students have prepared before for similar problems. This will ensure an engaging and concise activity.

Visualization: In the spring semester 2017, we used visualizations and animation to convey basic parallelism concepts such as multi-threading and for loop parallelization. The visualizations were designed using MS PowerPoint. For instance, we explained serial code that finds an item in an array with linear search. Next, rather than using one thread, we used two threads to find an item in an array, and we showed the students with a step-by-step animation how the item can be found fast with two threads (Figure 1).

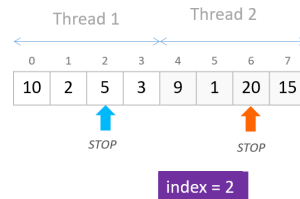


Figure 1. Visualizing finding an item in an array via two threads

Furthermore, we used an animated visualization to explain how parallel merge sort works using two threads first, and then four threads. Using animation, the visualization shows step-by-step how multiple threads side-by-side sort different portions of the array simultaneously. Students found the visualizations helpful, and they thought it improved their understanding of parallel programming. In conclusion, if designed right, we think this technique can significantly improve students’ understanding of parallel computing. In designing the visualizations, we made each visualization narrowly focused on a single topic (e.g. multi-threading), and showed the animation step by step to allow the students to notice the change. In future semesters, we are planning to programmatically write JavaScript code that visualizes how multi-threading works to sort arrays with quick and merge sort. The students will be able to control the settings of the visualization such as animation speed, data type, data size, number of threads, sorting algorithm.