# Spring-12: Thinking in Parallel: Hardware to Software— Adopting the TCPP Core Curriculum in Computer Systems Principles

Tim Richards
*School of Computer Science*
*University of Massachusetts Amherst*
*Amherst, Massachusetts*
*richards@cs.umass.edu*

***Abstract*—Introducing parallelism into an existing course is difficult if it is not the underlying theme. At the same time it is possible to re-focus existing course material by making it a common "thread" starting on the first day of class. This approach was used to improve student success rates in *CMPSCI 230: Computer Systems Principles*, a core course in the School of Computer Science at The University of Massachusetts Amherst. This course covers many aspects of the TCPP Core Curriculum [1] with special attention to processes, threads, networking, and web services toward the second half of the semester. Prior to making parallelism a topic of study from day one students encountered severe difficulties with later programming assignments that focus entirely on processes, threads, concurrency, and synchronization. This gap in understanding was attributed to the disconnect in student understanding when transitioning from a sequential to a parallel programming model. Making parallelism present in lectures, assignments, exercises throughout has drastically improved the success in later projects and assignments. This paper reports on the experience of the author and the modifications to this course that led to the improvement. We also discuss how this experience will help shape the objectives and re-design of the course that will be introduced in subsequent offerings.**

***Keywords*-Parallel programming education; multicore computing development/education; course design**

## I. Introduction

Students encounter great difficulty when transitioning from a sequential programming model to one that involves concurrency, parallelism, and synchronization. This difficulty can be attributed to the emphasis placed on programming single process/threaded programs in prior programming courses. To this end students must unlearn what they have been taught previously to comprehend parallelism, to *think in parallel*, and to design and implement programs that take advantage of multiple processes, threads, and the underlying hardware that supports it.

Introducing parallelism as an "afterthought" is the approach used in most computer science curricula to expose students to the latest trends in computer architecture and systems programming. The reason is that it is difficult to rethink existing courses and expensive to develop new course material that is both horizontally and vertically aligned. At the same time it is important for students to be knowledgeable and skilled in parallel concepts.

To make this transition manageable migration toward parallelism can be integrated at the individual course-level. Instead of re-designing a course from scratch students can be made to think in parallel starting with the first lecture in a way that complements existing material. Instead of abruptly switching from sequential to parallel students are exposed incrementally. This helps modernize course content over time and introduces concepts students need to succeed.

This migratory approach was used to improve student success rates in *CMPSCI 230: Computer Systems Principles*, a core course in the School of Computer Science at the University of Massachusetts Amherst. The purpose of this course is to provide students a good foundation in systems programming exposing them to aspects of computer architecture and a programmer's perspective of the operating system including multi-processing/threading, synchronization, and network services. As such it already covers many topics proposed by the TCPP Core Curriculum [1]. At the same time we found that students had tremendous difficulty toward the later part of the course when the emphasis turns toward topics in parallel programming—they simply did not understand how to *think in parallel*.

To overcome this barrier this course was modified to introduce parallelism up front. These changes resulted in higher success rates in later programming assignments that focus specifically on solving problems using processes and threads. The rest of this paper describes some of the changes that were made and how it impacted student performance.

## II. Background

The objective of this course is to give students grounding in systems topics and programming. The course covers compilation, how programs are executed by hardware, caching, virtual memory and allocation, exceptional control flow, multi-processing, concurrency and threads, synchronization, network programming, and web services. The ultimate goal is to provide students an introduction to systems and a good foundation to take upper-level systems courses on operating systems, compilers, and computer networks.

The assignments in the original course consisted of four projects—the first two related to the first half of the course and the second two emphasized topics in the second half. The intention is that each project incrementally builds on the previous. The first project required students to implement an inverted index generator to generate an inverted index mapping words to locations in text files. The goal of the assignment was to familiarize students with C/C++ programming. The second project had students implement a simple memory allocator to have students manipulate memory at a very low level. The third project required students to implement a multi-threaded HTTP web crawler to crawl and download HTML documents. In the final project students integrated the inverted index generator and web crawler into an HTTP search engine to return search results of the HTML documents that had been downloaded by the crawler and indexed by the inverted indexer.

The first two projects challenged students and were often completed by the due date. This is not surprising as it related to the material covered by the first half of the course and aligned to the familiar sequential programming model. The later projects often resulted in substantial difficulties and failed attempts. Projects submissions clearly demonstrated that students were still thinking sequentially. In the next section we describe the changes to the course that improved student understanding and success rates on the later two projects.

## III. Course Changes

The course was modified in three ways to improve student success rates in later topics. First, lecture material was updated to include content related to parallelism. This included extending the coverage of material in several topic areas with content related to parallelism. For example, we extended the coverage of the micro-architecture to include instruction-level parallelism, multicore, and GPUs. We also introduced new content on measuring parallel performance with Amdahl's Law versus Gustafson's Law. For those topics that did not lend themselves directly to parallelism we included short digressions. For example, when we cover the instruction set architecture and assembly we talk about how multiple programs or instruction streams could execute at the same time. We also made an effort to bring content related to threads and processes into earlier lectures.

The second change to this course focused on assignments. We kept the same projects, however, the memory allocator was replaced with a memory pool—in the spirit of the rest of the projects. We introduced four additional assignments that are a combination of written and programming problems. This allowed us to exercise concepts on parallelism earlier without major disruptions to the existing project assignments. For example, the first assignment asks students to look at the process of wood stacking. A single person must cut the wood, split the wood, and stack the wood. Students are then asked to solve the same wood stacking problem but with multiple wood stackers in parallel. This simple problem starts students thinking about solving problems in parallel and difficulties that one encounters, such as, what happens when two people try to stack the same piece of wood? To compensate for the additional time necessary for the new assignments we reduced the amount of programming required for the projects.

The third change involved modifying the first two projects to include aspects of parallelism at the skill level students were capable of. Because the inverted index generator is assigned before programming processes and threads we have students invoke multiple processes from the command line and measure performance before and after. This introduces the basics of processes and gives students an introduction to coarse-grained parallelism. The second project on memory pools extends student's understanding of processes and measurement by introducing background/foreground processes and process ids to manipulate processes without knowledge of the Unix process API.

## IV. Conclusion and Future Work

The changes made to this course have had a noticeable improvement on student understanding of parallelism and its impact on program performance. In particular, performance on later projects and assignments have increased and student questions and discussions during class are more clear and articulate. Thus, the effort to make parallelism a common thread has had a positive effect on student learning and has improved the intentions of the course.

In addition, our experience in improving coverage of parallel topics has given us an understanding on how to further improve this course in the future. We are currently working toward redesigning the course to be entirely project-based where each project will involve a more integrated approach to parallelism that will improve clarity in understanding of parallelism and synchronization. In particular, we are using processes and threads to drive our discussion of hardware support. Students will use projects to experiment on different hardware platforms and explore programming in multi-core and GPGPU environments to further their understanding in different programming models and how performance is impacted by parallelism and tradeoffs in overhead.

## References

[1] S. K. Prasad, A. Chtchelkanova, S. Das, F. Dehne, M. Gouda, A. Gupta, J. Jaja, K. Kant, A. La Salle, R. LeBlanc, M. Lumsdaine, D. Padua, M. Parashar, V. Prasanna, Y. Robert, A. Rosenberg, S. Sahni, B. Shirazi, A. Sussman, C. Weems, and J. Wu. Nsf/ieee-tcpp curriculum initiative on parallel and distributed computing: core topics for undergraduates. In *Proceedings of the 42nd ACM technical symposium on Computer science education*, SIGCSE '11, pages 617–618, New York, NY, USA, 2011. ACM.