EAPoster: Parallelism Throughout the Computer Science Curriculum

Steven Bogaerts, Brian Howard, Scott Thede, Gloria Townsend
Department of Computer Science
DePauw University
Greencastle, IN, U.S.A.
{*stevenbogaerts, bhoward, sthede, gct*}*@depauw.edu*

***Abstract*—This document provides an update in our efforts to integrate parallelism into a wide range of computer science courses at DePauw University, following [1]. This time we choose to provide the most comments on our newest effort, an introductory interdisciplinary course entitled Paradigm Shifts in Science. We also report continued work in CS1, Data Structures, Computer Systems, Foundations of Computation, Parallel Computing, Programming Languages, Web Development, and Artificial Intelligence.**

***Keywords*-parallelism; CS education; curriculum**

I. Paradigm Shifts In Science

Paradigm Shifts in Science is a new interdisciplinary course at DePauw. It is marketed to first-year students that have not shown interest in science in other ways, with the intent of building confidence and interest. It allows students to get a taste of various science disciplines, seeing how they differ and also how they integrate, in some cases working on the same problem from different angles. As suggested by the course title, it focuses on "paradigm shifts" in which new ideas forever change the way some concept is understood. In fall 2015, the course included five modules taught by five professors in biology, chemistry, and computer science.

Commonplace parallelism is clearly a paradigm shift away from sequential-only approaches of the past, and so this was the topic of Bogaerts' module in the course in fall 2015. The module proved a challenging context in several ways. The audience consisted of first-year students that did not consider themselves potential science majors, and in some cases may even be science- and math-phobic. Each module had just seven 1-hour "lectures" of contact time. Since the students had no prior programming experience, this is a very tight timeframe for an exploration of parallelism.

The content of the module was adapted from material for about 12 hours of contact time in a computer literacy course that Bogaerts had previously taught, described in more detail in [2]. PDC curriculum topics covered include Algorithms (Broadcast: A), Architecture (Multicore: C, Shared vs. Distributed Memory: A), Programming (Shared Memory: A, Message Passing: A, Tasks and Threads: C, Criticial Regions: C, Deadlocks: C, Data Races: C), and Cross-Cutting(Why and What is PDC?: C, Concurrency: K). The vehicle for hands-on work was the Scratch programming language, designed to be particularly easy for novices to pick up quickly. Scratch also provides an accessible context for the exploration of some important concepts in parallelism, since objects in Scratch are able to act in parallel and send messages to each other. This enables the exploration of basic task decomposition, communication, race conditions, and blocking and non-blocking commands. Thus Scratch provides a context for hands-on exploration of these topics, even for students with no prior programming experience and a mere 7 hours of contact time.

The other major component of the parallelism module was the use of analogies and demonstrations to illustrate concepts of parallelism, both before and after their hands-on exploration in Scratch. Some analogies are short and simple, like an illustration of walking with distracted children in a park to describe the `join` command, or a description of the use of a `lock` by analogy to the conch shell from the novel *Lord of the Flies*. Other analogies are more complex. For example, message-passing versus shared memory models of inter-process communication were illustrated by having students work together to sort cards under different constraints. Specifically, the shared memory group could sit next to each other but had to share a limited amount of "working memory", while the message-passing group could only communicate by passing notes from across the room.

Anonymous student evaluations indicate that students found the material interesting and manageable. Due to Bogaerts' previous experience using Scratch to introduce parallelism in similar contexts, the primary challenges of the module were not about the introduction of parallelism itself, but rather in forming deeper integrations with other modules. The module did briefly discuss parallelism in artificial neural networks and genetic algorithms, making a connection to other modules through these topics, but a tighter integration will be key to making further improvements in this course in future offerings involving a parallelism module.

II. Computer Science I

We have reported extensively on our **Computer Science I** (CS1) efforts in the past. We are pleased to report that this material is now stable and being adopted more widely. To summarize as described in previous reports, the CS1 material is a careful combination of illustration by analogy and basic programming in Java using a CS1Thread class.

The analogies leverage students' intuition about how the world works in parallel to give them insight into parallel computation. The CS1Thread class hides a few of the Java-specific details from the students so that they can focus on the parallel ideas without getting caught up in the syntax of extending classes or catching exceptions (neither of which we traditionally covered in CS1). Curriculum topics include Architecture (Multi-Core: K), Programming (Shared Memory: C, Distributed Memory: C, Task/Thread Spwaning: A, Tasks and Threads: C, Synchronization: C, Concurrency Defects: C, Performance Metrics: K) and Cross Cutting (Why and What is PDC?: C, Non-Determinism: K, Power Consumption: K). The material is showing strong success, with pre/post test results showing good increases both in understanding and interest. Students also report that the topic sparks their interest more strongly than standard CS1 topics like iteration and arrays.

III. Data Structures

Our **Data Structures** course is the follow-up to CS1. Curriculum topics include Programming (Shared Memory: C, Distributed Memory: C, Data Parallel: A, Synchronization: A) and Algorithms (Speedup: A, Divide and Conquer: A, Recursion: A). This material is presented in a way that fits within the standard topics of the course. In our experience, divide-and-conquer parallelism is an excellent choice, because it provides a relevant context for the practice of recursion and is naturally modeled by a tree – both important topics in data structures.

We consider a simple application of adding numbers in a large array, applying various decomposition strategies developed by analogy. The analogy is one of a head chef dividing a large pile of potatoes to be cleaned and peeled by four cooks in a kitchen. We consider issues that can arise if one cook is busier than another, or more efficient than another, or if one end of the potato pile is older and thus requires more work. We also consider the possibility of more or fewer cooks being available in the kitchen. This leads to lessons about writing algorithms for any number of processing units, taking only small amounts of work at a time to better accommodate potential differences in load and computational complexity. We explore divide-and-conquer parallelism as one possible implementation of these goals. Preliminary pre- and post-test results indicate that this effort is successful, both in increasing understanding of parallelism concepts, and in increasing interest. This applies both to those that were exposed to some parallelism concepts in the past, and also those with no prior exposure.

IV. Other Courses and Future Work

We continue efforts in a variety of other courses as well. In **Programming Languages**, the focus is on exploring various kinds of language support for PDC issues. In particular, the course covers aspects of the shared memory model and the message passing model, and how they impact programming in both imperative and functional languages. Students are evaluated through pre- and post-tests on their interest and understanding of PDC concepts. In future versions of the course, additional material will be developed to build on prior experience with data parallel programming and common libraries, and to go into more depth on memory models and how they relate to language design.

In **Computer Systems**, we discuss parallel processing briefly in the context of process management at the operating system level. In particular, we consider management of shared memory and other resources, including issues of scheduling and deadlock. In **Foundations of Computation**, we discuss parallel programming in Erlang briefly in the context of functional languages, considering message passing and process management. We also have a special topics course in **Parallel Computing**. The course focuses on programming in Java, C++, and Erlang. The Java and C++ portions expose students to the topic, give them some experience with using parallel thinking in two programming languages that they are already familiar with, and show them how to deal with the shared memory/resource model. Then, the Erlang section gives them some exposure to the message-passing model of parallelism. We discuss big data and the map-reduce algorithm as well. The ultimate goal of the course is to help the students think in terms of parallelism when solving problems, regardless of the particular technology in use.

In the near future, we plan to integrate concepts of parallelism into both the Web Development and Artificial Intelligence courses. In **Web Development**, we will discuss how a web server for any site handling large amounts of traffic is an excellent illustration of parallelism, as numerous HTTP requests must be handled simultaneously in order to provide a smooth experience for many simultaneous users. Related cybersecurity concerns include the use of distributed denial of service attacks to overwhelm a web server's resources. **Artificial Intelligence** (AI) is a field in which many algorithms are particularly computationally intensive, and thus can benefit greatly from parallelism. Both Bogaerts and Thede have supervised student research projects applying parallelism to AI, and we plan to integrate these ideas into the AI course itself soon.

References

[1] S. K. Prasad, A. Y. Chtchelkanova, S. K. Das, F. Dehne, M. G. Gouda, A. Gupta, J. Jaja, K. Kant, A. La Salle, R. LeBlanc *et al.*, "NSF/IEEE-TCPP curriculum initiative on parallel and distributed computing: core topics for undergraduates." in *SIGCSE*, vol. 11, 2011, pp. 617–618.

[2] S. Bogaerts, "Hands-on parallelism with no prerequisites and little time using Scratch," in *Topics in Parallel and Distributed Computing: Introducing Concurrency in Undergraduate Courses*, Prasad, Gupta, Rosenberg, Sussman, and Weems, Eds. Morgan Kaufmann, 2015.