Incorporating the NSF/TCPP Curriculum Recommendations in a Liberal Arts Setting

Akshaye Dhawan
Department of Computer Science
Ursinus College
Collegeville, PA
adhawan@ursinus.edu

***Abstract*—This paper [1]examines the integration of the NSF/TCPP Core Curriculum Recommendations in a liberal arts undergraduate setting. We examine how parallel and distributed computing concepts can be incorporated across the breadth of the undergraduate curriculum. As a model of such an integration, changes are proposed to Data Structures and Design and Analysis of Algorithms. These changes were implemented in Design and Analysis of Algorithms and the results were compared to previous iterations of that course taught by the same instructor. The student feedback received shows that the introduction of these topics made the course more engaging and conveyed an adequate introduction to this material.**

***Keywords*-Distributed Computing; Parallel Processing; Curriculum development;**

I. Introduction

This paper examines the integration of the NSF/TCPP Core Curriculum Recommendations in an undergraduate liberal arts setting. In addition to examining how the ideas presented here can be incorporated into the existing Liberal Arts Model Curriculum guidelines provided by the 2007 report of the Liberal Arts Computer Science Consortium (LACS) [1], we also examine and evaluate the adoption of these ideas in one course of the undergraduate curriculum at Ursinus College and present the planned future adoption of these ideas in another course of the undergraduate curriculum.

In the first phase of this implementation, Design and Analysis of Algorithms was modified to introduce Parallel and Distributed Computing Concepts (PDC). In the second phase, a similar modification is proposed to Data Structures. Both of these are presented in this paper. The results show that the modified Algorithms course was extremely well received by students and the introduction of these topics made the course relevant. For the purpose of comparison, student feedback was compared to the unmodified version of the same course taught by the same instructor.

The remainder of this paper is as follows. Section II provides some background to the liberal arts model of higher education and the role of computer science in the liberal arts. Section III presents the undergraduate curriculum in computer science at Ursinus College. In Section IV we introduce our proposed changes to Data Structures and Design and Analysis of Algorithms. Section V examines the student feedback for the changed course. Finally, we conclude in Section VI.

II. Computer Science and the Liberal Arts

A liberal arts curriculum aims at developing intellectual capabilities through a breadth of educational experiences that encourage the pursuit of knowledge. Historically, the seven liberal arts were grammar, rhetoric and logic (the trivium) and geometry, arithmetic, music, and astronomy (the quadrivium). Over time this model of education has evolved. As practiced in American colleges today, liberal arts is often described as the study of three core divisions:

- the humanities (literature, language, philosophy, the fine arts, and history),
- the sciences (biology, chemistry, computer science, mathematics, physics) ,
- the social sciences (politics, economics, sociology)

It is worth noting that computer science is placed in the middle category and indeed primarily evolved from Mathematics Departments in the liberal arts setting. Computer Science is of increasing relevance to the liberal arts curriculum for much the same reasons that Mathematics was originally a part of the core of liberal studies. A working knowledge of computer science is now a requirement for students in most disciplines of the sciences and even in the social sciences. Indeed this is reflected in the science curriculum at Ursinus. Introduction to Computer Science (CS 173) is required of all Mathematics and Physics majors and recommended for Biology and Neuroscience majors at Ursinus. Also, a new course titled In Silico: Experimental science via Computer Science (CS 170) is planned for the Fall 2012 and is to be taught by the author. CS 173 consists of an object oriented introduction to computer science and uses Java. CS 170 on the other hand uses a general purpose language like Python and has the students build programs that analyze big data and carry out simulations on problems drawn from a number of

[1]This work was supported by the NSF-TCPP Early Adopters Fall 2011 program

disciplines like Biology, Chemistry, Physics, Geographical Information Systems, Medical Imaging and Business.

In addition to covering disciplinary material, liberal arts programs in computer science emphasize some larger concepts. Three general-purpose capabilities that are fundamental characteristics of a liberal arts education are the ability to organize and synthesize ideas, the ability to reason, and the ability to communicate ideas to others. The design, implementation, and analysis of algorithms and data structures uses and develops all three of these capabilities. This liberal arts emphasis on broader concepts that are not tied to specific languages, architectures or operating systems is very much in line with the broader spirit of the recommendations stated in the NSF/TCPP Curriculum Initiative [2].

According to LACS, a key component of any liberal arts perspective to computer science should include multiple problem-solving paradigms. However, they acknowledge the difficulty of doing this effectively within existing curriculum requirements. This is where we believe that the NSF/TCPP curriculum suggestions can play a role. They allow for the introduction of parallelism as a problem-solving paradigm through the introduction of key ideas across the breath of the curriculum. Our approach to this integration of the NSF/TCPP curriculum with the LACS Liberal Arts Curriculum requires no additional courses to be offered. Instead, it challenges instructors to rework existing courses to introduce these ideas often as a supplement to existing material. We strongly believe that a key skill that undergraduate computer science students (particular those in the liberal arts) should have is that of being able to break down and design solutions to a problem in more than one paradigm. However, the tension emerges from the acknowledgment that in a liberal arts setting, a typical computer science curriculum [1] consists of:

- Computer Science courses - about 30%
- Mathematics courses - about 10%
- Science courses - about 10%
- Humanities, Social Sciences and Languages - about 50%

What this means in very real terms is that there is little room for adding new requirements to the computer science major. Hence, the inclusion of key concepts of parallel and distributed computing must necessarily come through a reworking of these ideas into existing courses of the curriculum.

In the next section we provide a broader introduction to the Computer Science curriculum at Ursinus College.

III. The Computer Science Curriculum at Ursinus

The computer science curriculum at Ursinus is fairly representative of a typical liberal arts program in computer science. In order to major in computer science there are four required core courses (Introduction to Programming, Data Structures, Architecture, Algorithms), any one of two theory courses (Theory of Computation, Programming Languages) and four other electives one of which must be a capstone course (Software Engineering, Operating Systems, Computer Networks, Databases, Graphics, Artificial Intelligence and High Performance Distributed Computing). Currently, the only course that explores parallel and distributed computing in some detail is the elective *CS-478 High Performance Distributed Computing*. Additionally, a CS 0 course in Greenfoot [3] (Computer Science for the liberal arts) is offered to non-majors in the arts and humanities and the CS 170 course targeted to the experimental science mentioned in Section II is offered to students in the

CS-478 High Performance Distributed Computing is typical of elective courses on parallel and high performance computing and focuses on teaching methods in parallel computation on a variety of parallel architectures including multicore, cluster and grid computers. In addition to theoretical concepts like work, cost, Amdahl's law etc., the course also covers programming using MPI and Open-MP. Over the course of the semester, the students typically work on four involved projects that require understanding the nuances of designing and implementing software that takes advantage of a parallel system. However, the main shortcomings of this course is that not all majors take it since it is offered every other year and is an elective.

IV. Adoption of the NSF/TCPP Curriculum

We believe that it is important to continue offering an elective that is focused on parallel computation like CS-478. However, given recent trends in the widespread use of multicore processors, GPU's, cloud computing and clusters it is essential that *every* computer science major have an introduction to the key ideas behind parallel and distributed computing. This can be achieved by introducing these ideas earlier in the curriculum via two core courses - **Data Structures** and **Design and Analysis of Algorithms**. By incorporating these changes into core courses, we can ensure that parallel computing as a problem solving paradigm and indeed as a way of thought is introduced earlier in the curriculum. This may also serve to ameliorate the difficulty some students have in moving to a parallel approach to problem solving after spending years being trained to break problems down iteratively in a serial model.

We will now outline specific changes to the two courses. The author has taught CS 174 Data Structures in Spring 2010, Fall 2010 and Spring 2011 and CS 371 Design and Analysis of Algorithms in Fall 2009, Fall 2010 and Fall 2011. The changes being proposed were incorporated into Cs 371 Design and Analysis of Algorithms in the Fall of 2001. We will also be incorporating the proposed changes into CS 174 in the Fall of 2012 but some of the ideas are presented here. For prospective adopters who question the time constraints that the introduction of these topics may impose to

Table I
DEFINITION OF LEARNING LEVELS FROM BLOOM'S TAXONOMY [7]

Learning Level	Abbreviation	Definition
Knowledge	K	Student recalls or recognizes information, ideas, and principles in the approximate form in which they were learned.
Comprehension	C	Student translates, comprehends, or interprets information based on prior learning.
Application	A	Student selects, transfers, and uses data and principles to complete a problem or task with a minimum of direction.

the schedule of these classes, the hope is that these changes can be introduced as a supplement to existing topics and by streamlining the time spent in discussing certain topics. It is also worth pointing out that as a pilot we are introducing these ideas in only two courses but such an adoption can be made across the curriculum in a number of different courses including Computer Architecture, Programming Languages and Operating Systems [2].

A. Data Structures

CS 174 Data Structures is our standard CS II course and represents the second course in the major for most students. This course is a great candidate for introducing the shared memory model of parallel computation and for introducing thread level parallelism. We believe that by focusing on threading as the primary means to expose the student to parallelism, the instructor can teach basic techniques and problems like concurrency control. This is essential because a typical CS-II course does not usually allow the time needed to introduce the novice programmer to MPI, PVM or Open-MP. Introducing threads and concurrency control early in the curriculum will also allow for a more thorough discussion of these concepts in upper-level systems courses like Operating Systems. Additionally, all students have had Java by this point in our curriculum and threading in Java can be introduced without much effort. This is relevant since Data Structures is a core course that already introduces several important concepts.

For the proposed changes, we used the Bloom's Taxonomy for learning outcomes [4], [5]. Additionally, [6] provides an introduction to using Bloom's Taxonomy in Technology courses. The changes proposed target different learning objectives. The changes impacted three primary learning levels - Knowledge (K), Comprehension (C) and Application (A). These are defined in Table I.

The topics proposed to be included along with their Bloom Number [7], [4] and Learning Outcomes are shown in Table II. As can be seen from the table an introduction to basic models of the shared memory model and speedup were presented and threading was used to illustrate these ideas. This was also used to show problems like concurrency control and race conditions.

B. Design and Analysis of Algorithms

CS 371 Design and Analysis of Algorithms is also a core course required of all majors. This course is typically taken by students in the fall semester of their junior or senior year *before* they may take the elective CS 478 High Performance Distributed Computing in the spring. The course is an ideal candidate for introducing parallelism in more detail since the material in the course lends itself well to a discussion of speedup and the comparison between parallel and serial algorithms. We propose introducing these ideas through studying parallel versions of algorithms (like Merge Sort, Matrix Multiply) whose serial versions are already a part of the course.

In the Fall of 2011, the course was revised to introduce parallel methods. The topics included along with their Bloom Number [7] and Learning Outcomes are shown in Table III.

In this class, the PRAM model was presented as a theoretical construct. The students were introduced to shared memory and message passing models and some basic analysis of time complexity was presented for parallel algorithms. An effective approach that emerged as the class went on was that of presenting a parallelized approach to a serial solution they had already seen. Merge Sort was used for this purpose and was presented with a thorough analysis and pseudo-code. This seemed to make a real impact on students and drove home to the power of effective parallelization.

V. EVALUATION

As a first measure of evaluating the success of these redesigned courses in conveying key concepts of parallel and distributed computing to students we compare anonymous end-of-semester student evaluations for CS 371 Design and Analysis of Algorithms in Fall 2011 to the unmodified version of this course offered in Fall 2010 and Fall 2009. Note that all three classes were taught by the author. In addition to these evaluation surveys, an independent teaching observer was also used every other class in CS 371 during Fall 2011. Also, it will be worth looking at the performance of these students in the High Performance Computing course as explained in Section VI. We are in the process of collecting this data during Spring 2012.

The results of these student surveys are present in Table IV. The response rates were 77% for Fall 2009, 85% for Fall 2010 and 90% for Fall 2011. For the questions below, the students were asked to respond on a scale of 1 to 5 with 1 being the best possible score. The table shows the means scores in the survey. Additionally, they had the option of providing comments some of which are reproduced below

Table II
TOPICS PROPOSED FOR INCLUSION IN DATA STRUCTURES

Topic	Bloom #	Learning Outcome
Recursion	C/A	Show independent subproblems in recursion tree eg. Fibonacci
Shared memory	K/C	Basic model, speedup discussion
Threads	A	Creation, dynamic multithreading, spawn, sync operations
Concurrency Control	A	Illustrate race conditions, need for synchronization

Table III
TOPICS INCLUDED IN ALGORITHMS

Topic	Bloom #	Learning Outcome
Asymptotics	A	Introduce parallel time/space complexity vs. serial
Speedup	C	Experiment with timing, discuss measuring speedup, trade offs
Task Graphs	C	Introduce decomposition, dependencies and pipelines
Divide and conquer	A	Merge Sort, Naive multithreaded and Strassen's matrix multiply
Sorting	A	Parallel Merge Sort with analysis and recurrence relation

Table IV
STUDENT SURVEY SUMMARY FOR DESIGN AND ANALYSIS OF ALGORITHMS

Question	Fall 2011	Fall 2010	Fall 2009
Did the course increase your knowledge or skills in the subject area?	1.11	1.17	1.29
Clarity of presentation of the material	1.1	1.17	1.57
On average, how many hours do you estimate you have spent on this class outside the classroom per week?	5-7	5-7	3-5

for the modified course. Note that in every measure the modified course was better received by students. Also, these changes came with a similar time commitmment requirement from students.

Student comments from the modified class include the following:

> The use of current topics and examples made this class very interesting.
>
> I learned a great deal in this class. It was a tough and important class and I learned a lot.
>
> Discussion of real world applications made this class relevant.
>
> Gave us a practiced understanding of the material.

Additionally, an independent consultant who observed the modified class had the following comments on a class where parallel algorithms were introduced using Merge Sort as an example. The teaching consultant was trained and used the methods detailed in [8], [9]. The consultant was made available through the Teaching and Learning Initiative at Ursinus College funded by the Mellon Foundation [10].

> Introducing Moore's Law and its limits was effective. This is interesting and students seem very engaged by the topic. Pretty much everyone has their notebooks out and is paying attention.
>
> Since a lot of students are familiar with mobile devices, using them to introduce parallel computing is a good real-world example of the topic.
>
> You use real-world examples such as writing a book and painting a fence as examples of why using multiple processors to solve a problem isn't always the best way. This is a good way to illustrate the point. I like your use of the projector during these examples.
>
> You used a lot of really interesting real-world examples to illustrate the topic you're teaching. Students seemed really engaged in today's class.

VI. CONCLUSION AND FUTURE WORK

It is our sincere hope that this paper presents evidence that in addition to offering an upper level course in Parallel and Distributed Computing (PDC), modifying lower level courses in the undergraduate computer science curriculum can be an effective way of introducing these concepts to a wider audience. Student responses show that these changes made these courses more relevant and interesting to students.

In order to evaluate the impact of the proposed changes, we are collecting, synthesizing and interpreting information about student understanding, reasoning and practical skills for students who have taken the modified Data Structures and Algorithms sequence before taking CS 478 High Performance Distributed Computing and comparing their performance in this course to that of students who have not had this prior introduction to parallelism. Data collected will include in-class feedback, performance on assignments

and programming tasks and midterm/end of semester student evaluations. CS 478 is currently being offered in the Spring 2012.

REFERENCES

[1] 2007 report of the Liberal Arts Computer Science Consortium (LACS), http://cs.wellesley.edu/ pmetaxas/LACS2007report.pdf

[2] NSF/TCPP Curriculum Initiative, http://www.cs.gsu.edu/ tcpp/curriculum/sites/default/files/NSF-TCPP-curriculum-Dec23.pdf

[3] Greenfoot, http://www.greenfoot.org/door

[4] Bloom, B. S., Engelhart, M. D., Furst, E. J., Hill, W. H., Krathwohl, D. R. (1956). Taxonomy of educational objectives: the classification of educational goals; Handbook I: Cognitive Domain New York, Longmans, Green, 1956.

[5] Anderson, L.W., Krathwohl (Eds.). (2001). A Taxonomy for Learning, Teaching, and Assessing: A Revision of Bloom's Taxonomy of Educational Objectives. New York: Longman

[6] Forehand, M. (2005). Bloom's taxonomy: Original and revised.. In M. Orey (Ed.), Emerging perspectives on learning, teaching, and technology.

[7] Bloom's Taxonomy, http://www.edpsycinteractive.org/topics/cogsys/bloom.html

[8] Derek Bo Center for Teaching and Learning, Harvard University, http://isites.harvard.edu/icb/icb.do?keyword=k1985&pageid=icb.page29735

[9] Bryn Mawr Teaching and Learning Initiative, http://www.brynmawr.edu/tli/

[10] The Andrew Mellon Foundation, http://www.mellon.org/