Applying the NSF/TCPP Curriculum Recommendations to a Liberal Arts Curriculum

Akshaye Dhawan
Ursinus College
Collegeville, PA
adhawan@ursinus.edu

***Abstract*—This poster examines the integration of the NSF/TCPP Curriculum Recommendations as a whole into an undergraduate Liberal Arts curriculum as proposed by the Liberal Arts Computer Science Consortium (LACS). As part of our work as Early Adopters, a model of such an integration was carried out with changes to two core courses. 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. In this poster, we investigate a curriculum wide adoption and integration that would be useful for tailoring the recommendations to a liberal arts setting.**

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

I. Introduction

This poster examines the integration of the NSF/TCPP Core Curriculum Recommendations [2] in an undergraduate liberal arts setting. Our focus in this poster is to examine how the ideas presented in the NSF/TCPP recommendations can be incorporated within the existing Liberal Arts Computer Science Curriculum guidelines provided by the 2007 report of the Liberal Arts Computer Science Consortium (LACS) [1].

II. Adoption of the NSF/TCPP curriculum in a Liberal Arts Setting

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.

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.

According to the 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 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.

III. Implementation of the NSF/TCPP Recommendations in the liberal arts

We are currently working on a paper that will present possible integrations of the NSF/TCPP guidelines with the LACS curriculum. In this section we briefly present the

LACS curriculum and identify courses that can be modified. A detailed description of this integration along with Bloom outcomes for each topic will be presented in the poster. This is omitted here for lack of space.

A. LACS curriculum

The LACS curriculum [1] is typical of most liberal arts colleges. The requirements for the major typically include 9 computer science courses and 2-3 mathematics courses. Summarized briefly, it consists of the following:

- Introductory Courses: Three basic courses form a part of the introductory course requirement. These include a two sequence programming course in the form of CS 1, CS 2 and a course on Discrete Mathematics.
- Core Courses: A core of five courses is recommended by the LACS. These are Computer Organization, Design and Analysis of Algorithms, Principles of Software Development, Theory of Computation and Principles of Programming Languages.
- Electives: The curriculum recommends that major choose 3-4 elective courses. Typical suggested offerings include Operating Systems, Computer Networks, Artificial Intelligence, Graphics, Distributed and High Performance Computing and Compiler Design.

B. Adapting the NSF/TCPP recommendations to the Liberal Arts

As can be seen from the preceding description, there is little room to offer new courses in the liberal arts model. Indeed a typical liberal arts program will offer only one elective course on parallel programming/high performance computing. While this may be perceived as a disadvantage, in reality it provides for enormous opportunities to use the NSF/TCPP recommendations on modifying existing introductory and core courses to introduce parallel thinking early and often. We have adopted some of the recommendations to an introductory course (CS 2 Data Structures) and a core course (Design and Analysis of Algorithms) with considerable success. This work was presented in [3].

Next we present our proposed integration along the lines of the NSF/TCPP recommendations in the three main areas of Architecture, Programming and Algorithms.

- Programming
  We strongly believe that parallel programming models need to be introduced early in the curriculum. However, the complexity of this material needs to be kept at a level where it is still approachable by a freshmen or sophomore student. Our approach involves modifying CS 1 to introduce thread-level parallelism through Java threads (or their equivalent in other languages). Introducing parallelism through language constructs without the use of additional libraries allows for an efficient means to introducing parallel thinking. This needs to be reinforced in CS 2 through the discussion of data structures in parallel setting. This also allows for a gentle introduction of topics like the use of multi cores, shared memory and message passing models - often as questions raised by students as they think of parallel models alongside serial ones. This is essential because a typical CS-I and CS-II course sequence 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.
- Algorithms
  A course in the Design and Analysis of Algorithms is a core part of the LACS curriculum. This 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. Additionally, theoretical constructs like the PRAM model can also be introduced along with shared memory and message passing models and some basic analysis of time complexity was presented for parallel algorithms. An effective approach may be that of presenting a parallelized approach to a serial solution they had already seen. This will allow for students who do not take an elective in High Performance Computing to be introduced to basic concepts in parallel computing.
- Architecture
  We strongly agree with the NSF/TCPP recommendation that topics in parallel architecture can be interwoven in multiple lower level courses. We felt that both Computer Architecture and Operating Systems were well suited for a discussion of memory hierarchies, pipelining, cache coherence and race conditions. In particular, since students in CS 1 and CS 2 would have been introduced to thread level parallelism, some of these concepts can be introduced through programming assignments.

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-version1.pdf

[3] Akshaye Dhawan, Incorporating the NSF/TCPP Curriculum Recommendations in a Liberal Arts Setting, EduPar Workshop, 26th IEEE International Parallel and Distributed Processing Symposium (IPDPS), Shanghai, China, 2012.