ABSTRACT
The recent model Computer Science Curricula 2013 (CS2013) is spurring much discussion about the newly added Knowledge Area, PD - Parallel and Distributed Computing, which was promoted to the core. Courses in parallel computing have long been offered as elective courses but may now become a requirement in undergraduate level programs of study for majors. This paper explores the implications for undergraduate degree programs, K-12 curricula, and potential CS education research concerning best practices, pedagogy, and effects of early introduction of concurrency and parallelism concepts.

Categories and Subject Descriptors
K.3.2 [Computers and Education]: Computer and Information Science Education—Computer Science Education

General Terms
Human Factors, Languages, Experimentation.

Keywords
Introductory programming; concurrency concepts; computer science education; curriculum design.

1. INTRODUCTION
Multicore processors have experienced rapid proliferation in computer hardware and are now considered the standard. [13] The innovation of multicore processors is recognized by IEEE as one of the top 11 technologies of the 2001-2010 decade. [14] As a result, demand is growing for computer science graduates who can exploit the power of these architectures through programming that makes use of parallel, concurrency, and distributed computing concepts. Silicon Valley companies, such as Intel, have been leaders in pushing for educational efforts to help students learn how to “think parallel,” employers in other industries are also recognizing the need for programmers who can write for multicore chip architectures. For example, Jeffrey Birnbaum, chief technology architect at Bank of America, notes that Wall Street has a growing concern about the dearth of programmers with skills in parallel computing. Birnbaum says, “With the sea change that’s coming -- parallel computing, multicore processors -- the skill of the programmer matters more.” [5]

Thanks to the foresight of members of the ACM/IEEE-CS Joint Task Force, the CS2013 model curriculum recognizes the “vastly increased importance of parallel and distributed computing.” [1] The recently released 2013 curriculum draft identifies essential concepts of concurrency and parallelism in programming models and pragmatics, algorithms, computer architecture and distributed systems. In previous ACM/IEEE-CS curricula, these concept topics were distributed across other Knowledge Areas as elective topics. CS2013 dedicates an entire new Knowledge Area (PD) to parallel and distributed computing topics and promotes it to the core of topics that should be included in an undergraduate computer science curriculum.

Although the newly added Knowledge Area includes both parallel and distributed computing topics, this paper focuses only on parallel and concurrency and the impact of infusing these concepts in the model curriculum. Both immediate and longer term potential impact on introductory programming courses and K-12 curricula and pedagogy are considered. Of particular interest is possible CS education research directions that investigate effective techniques and best practices for integrating “concurrent thinking” into “computational thinking”.

2. IMMEDIATE IMPACT
The most immediate impact of the new PD Knowledge Area in CS2013 will likely be modifications to the classification of upper division courses in undergraduate computer science programs of study. Courses in parallel programming and distributed computing may well be reclassified as required, rather than elective. The quick and easy way to manage this change is to continue to offer these courses in the upper division level (typically 300-400 level courses). Students taking these required course(s) will be expected to have already completed introductory programming and computer architecture courses.

The justification for this expected impact is the current state of affairs in many computer science departments across the nation. Computer science department enrollments plummeted after the dot-com bust in 2004, as illustrated by Figure 1 which is based on source data from the Computing Research Association (CRA). Data collected since 2010 shows in increase in enrollment but this trend may be deceptive in that numbers reported prior to 2010 were typically for Computer Science (CS) and Computer Information Science (CIS) majors only but starting in 2010 the
As a result of declining CS major enrollments in the last decade, the number of faculty and the amount of financial support has also decreased dramatically. Many CS departments will, therefore, likely respond to curriculum model changes in a path requiring the least stress on faculty teaching loads and departmental budgets – notwithstanding the squabbles over how many credit hours can be required and which courses should trade status, required vs. elective.

3. FUTURE DIRECTIONS

While the most likely immediate impact is not dramatic, it is likely to lead to a rippling effect through the lower-division of the undergraduate curriculum and even into K-12. Signs of such changes are already on the horizon. Over the last three years, four proponent presentations for early introduction of concurrency were given at ACM’s SIGCSE (Special Interest Group in Computer Science Education) Symposium.[3, 4, 9, 12] Further, a working group report was issued at ITiCSE 10 on strategies for preparing students for a multicore world.[2] In other words, the call is out for integrating parallelism and concurrency concepts “early and often” in the undergraduate curriculum. The challenge for CS education researchers is to develop pedagogy and teaching methodologies for teaching and learning parallel and concurrency concepts and to investigate the effectiveness of different approaches. This research is not limited to undergraduate degree programs. We also need to consider what infusions in the K-12 curriculum would help prepare students for a multicore world.

3.1 Undergraduate early introduction

First and second year computing courses have traditionally taught program design and implementation where a single processor, a single process, and sequential execution is the way inner working of a computer is envisioned. This results, however, in programmers who have a logical “top to bottom” and “left to right” ingrained mental model of programs that execute in sequential fashion. Subsequently, students who are introduced to parallel programming in upper level courses struggle with how to design, implement, and debug code for doing things in parallel or concurrently. As observed by Ko, Burgstaller, and Scholz, “When students finally encounter parallel programming, many perceive the transition from sequential to parallel thinking as difficult.” [12] For this reason, Ko and other CS educators are stepping up to argue for teaching parallel programming much earlier in the curriculum.

3.2 Mainstream approach

A mainstream approach is advocated by the CRA. In its “Revitalizing Computer Architecture Research” report, the CRA argues that: “Parallel programming must be made easier and must become the mainstream programming method.” [10] Leading CS educators such as Kim Bruce are suggesting that concurrency and parallel programming concepts will be needed in many STEM disciplines.

Today’s students in many STEM disciplines are now required to take an introductory level programming course but the demands of credit hours necessary to complete their major make it difficult to take additional higher-level courses such as a course in parallel computing. Also, the higher-level courses are designed for majors and therefore include an in-depth parallelism body of knowledge far beyond the general concepts needed by other STEM majors.

One possible solution is to integrate a selection of fundamental parallel and concurrency topics into lower-level, introductory courses by using educational software tools that support a gentle introduction. This is the approach used by Dann, Slater, and Cooper in Alice software tools, curriculum and instructional materials. [7, 8] For example, in an Alice 3 and Java introductory programming course [7] both “do in order” and “do together” control structures are used to create 3D animation programs. With two control structures – one sequential and one concurrent – students design and learn to think algorithmically for both sequential and concurrent execution. Bruce, Danyluk, and Murtagh use a similar approach in their CS1 course, where they make use of a customized objectdraw library for Java to implement an objects-first approach that “incorporates event-driven programming techniques from the beginning, and includes concurrency quite early in the course.” [3]

Another approach is to offer a lower-division course specifically aimed at parallel programming concepts. This approach would allow a more in-depth study of parallel concepts (such as parallelism and execution indeterminism, thread-and-lock based programming, and stream-parallel programming) without requiring multiple pre-requisites, other than CS1. Ko, Burgstaller, and Scholz have experimented with using a GPU language (CUDA) to develop a one semester parallel programming course for students in their second or third semester who had experience with C programming but had not taken courses in operating systems or algorithms. [12]

These efforts toward a mainstream approach in undergraduate institutions are in the early stages. One may expect that more investigation and reporting will be used to develop and test pedagogy and methodologies for effecting teaching and learning parallel and concurrency concepts.
3.3 K-12

Any discussion of CS education research directions for K-12 curriculum must occur within a framework that recognizes the current struggle to gain acceptance for CS education as an essential component in K-12 education, overall.

Over the last decade, several influential forces have waged powerful battles to advocate for the inclusion of CS education in K-12 curricula. One notable force is the Computer Science Teachers Association (CSTA), led by Chris Stephenson. A second influential force has been Jeannette Wing’s campaign for Computational Thinking as a universally applicable attitude and skill set for everyone. In the media, corporate representatives such as Jordan Lloyd Bookey, Head of K-12 Outreach at Google, have lent their weight to argue for inclusion of Computer Science in the Common Core Standards. [11]

At the time of this writing, 45 schools have adopted the Common Core Standards, but Computer Science and Information Technology are largely left out of the picture. The only mention of Computer Science in the Mathematics Standards is a reference to the AP Computer Science A course as an elective.

On the other hand, in those schools where the Computer Science Concepts and Principles course (CSTA curriculum, grades 9-12) is being offered, the Computational Thinking objectives include a “…demonstration of concurrency by separating processes into threads.”[6]

An encouraging note is the widespread usage of Alice and Scratch in K-12 schools where programming is being taught. Concurrency underlies the interactive programming capabilities of each of these educational tools. In Scratch, concurrency is implicit in that multiple scripts can be created for a single event. At runtime, these scripts will execute concurrently. In Alice, concurrency is explicit as students can write code in a “do together” code block. The code within the “do together” block creates threads that run concurrently.

CS education research into pedagogy and effective methodologies for introducing fundamental concepts of concurrency and parallelism is a promising direction, still in the launch phase.

4. SUMMARY

The recent model Computer Science Curricula 2013 (CS2013) newly added Knowledge Area, PD - Parallel and Distributed Computing and its promotion to the core is likely to catalyze changes in CS undergraduate degree programs as well as in K-12 curricula. These changes offer opportunities for new CS education research directions.

5. REFERENCES


