Fall-12: Using C++11 to teach Concurrency and Parallelism Concepts

Dhananjai M. Rao
Computer Science and Software Engineering (CSE) Department
Miami University, Oxford, OH 45056–1601, USA
Email: raodm@muohio.edu

***Abstract*—In our current undergraduate curriculum, the Operating Systems (OS) course provides in-depth understanding of concepts related to concurrency and parallelism. Typically, the laboratory exercises and programming projects in this course are performed in C-language under Linux, an OS that is unfamiliar to many students. Most of the students primarily have only Java experience under Windows^{TM}. Consequently they experience steep, multifaceted learning curves that distracts them from focusing on core concepts in the course. In an endeavor to address this issue, the Fall 2012 offering of this course, adopted C++11 as the programming language to build on the object oriented skills of the students, utilize Standard Template Library (STL) containers, and algorithms to ease programming while maintaining focus on concepts related to concurrency and parallelism. In addition to leveraging standard C++ threads and parallelized STL algorithms, the course utilized several newly introduced libraries and design patterns to explore modern solutions to many standard multithreading and synchronization issues.**

This paper and poster presents experiences with using C++11 language and standard libraries to teach several concurrency and parallelism concepts in the OS course. The paper also summarizes data from direct assessments, indirect assessments, and student evaluations which indicates that use of C++11 standard enabled maintaining focus on core concepts, motivated learning of novel object oriented design patterns, and facilitated effective application of concurrency and parallelism concepts.

***Keywords*-Operating System, C++11, Concurrency & Parallelism, Standard Template Library (STL), STL algorithms**

I. Introduction

Catalyzed by numerous changes in hardware technologies, modern software systems have also rapidly evolved into multithreaded systems in which concepts related to concurrency and parallelism play a fundamental role in their design, development, and maintenance. Analogous to efforts in many institutions [1], the Computer Science and Software Engineering (CSE) department at Miami University [2] is enhancing its ABET accredited undergraduate curriculum to increase coverage of contemporary concepts and skills related to Parallel and Distributed Computing (PDC). In the CSE department's undergraduate curriculum, introduction to core PDC concepts such as multiprocessing, multithreading, and synchronization occurs in the Operating Systems (OS) course (CSE 381 [3]). Accordingly, in conjunction with the NSF/IEEE-TCPP Curriculum Initiative on Parallel and Distributed Computing [1], the several topics covered in the Fall 2012 offering of the OS course were streamlined to align with the NSF/IEEE-TCPP curriculum.

In conjunction with aligning several core topics with the NSF/IEEE-TCPP curriculum, the primary programming language utilized in the course was revised from C-language to C++11, the latest ISO/IEC 14882:2011 [4] standard version of C++. The motivation to transition from C to C++ was multifold. The primary motivation was to minimize the steep, multifaceted learning curve experienced by students due to:

- Introduction of new programming language, as most of the prerequisite courses are taught in Java. Given the students background, utilizing C-language involving heavy use of pointers, need for dynamic memory management, and its non-object-oriented nature often distracted students from core concepts and hindered effective coverage of topics in the course.
- Initiation to Linux via a predominantly Command Line Interface (CLI) based laboratory environment which is a significant change from a Graphical User Interface (GUI) driven Windows^{TM} based laboratory setting for many students.
- Introduction and extensive use of PDC concepts, a paradigmatic change from the traditional serial programming that the students are conversant with.

The Fall 2012 offering of the OS course [3], used C++ as the programming language to build on the object oriented skills of the students. The course utilized Standard Template Library (STL) containers to practically eliminate complex pointer usage and memory management issues. In addition, the use of STL algorithms was highly encouraged to ease development of sophisticated applications while maintaining focus on concepts related to concurrency and parallelism. Furthermore, all of the standard system calls that expose a C-language API could be readily invoked from C++ permitting a convenient mix of techniques. The C++11 standard was adopted to leverage standard C++11 threads (`std::thread`), parallelized STL algorithms, and design patterns (such as: `std::future`, `std::async`, `std::promise`, `std::atomic` etc.) to explore modern solutions to many standard multithreading and synchronization issues.

Remainder of the paper is organized as follows. Section II provides an overview of the course settings and the student body in the Fall 2012 offering of this course. A brief overview of various lectures, laboratory exercises, and homework projects used in the course are discussed in Section III. Feedback and statistics collated from both formative direct assessments and summative indirect assessments conducted during the course to evaluate the effectiveness of using C++11 are discussed in Section IV along with brief concluding remarks.

II. Course Setting

The CSE-381: Operating Systems (OS) [3] is a core course, requiring Data Structures (CSE 274 [5]) and Computer Architecture (CSE 278 [6]) as prerequisite courses. Consequently, most students have a junior or senior standing. The Fall 2012 offering of the OS course was a three credit hour course that consisted of two sections that were taught in a lecture-laboratory setting with two lecture hours and two laboratory hours each week. The lectures were conducted in a traditional class room setting for all students from the two sections combined together. The Fall 2012 course consisted of 39 students with an almost even split between juniors and seniors, with seniors slightly outnumbering the juniors. Furthermore, the class included students from the following four majors from two different departments: 23 computer science and 4 software engineering students from the CSE department; 9 computer engineering and 3 electrical engineering students from the Electrical and Computer Engineering (ECE) department. The course used the following two textbooks that were available in electronic form to all students: "Operating System Concepts" by Silberschatz *et al* [7] and "C++ Concurrency in Action: Practical Multithreading" by Williams [8].

III. Coursework

The Operating Systems (OS) [3] course provides a broad coverage of OS concepts including the topics pertaining to parallel and distributed computing, such as: multitasking, process management, Inter-Process Communication (IPC), multithreading, race conditions, process synchronization, deadlocks, and brief introduction to distributed operating systems [9]. Various course materials including revised course syllabus, PowerPoint presentations, laboratory exercises, homework assignments, and handouts are available as Supplementary Online Material (SOM) for this paper via http://www.users.muohio.edu/raodm/cse381_edupar13/.

Table I summarizes the time invested in covering various concurrency and parallelism topics in the course. The course increased emphasis on multithreading which required an extra week of course work. Consequently, to accommodate this extra week, the first week that was reserved for review of pertinent computer architecture concepts was coverted into a comprehensive homework assignment involving extensive literature review (see Homework #2 in SOM [9]). Furthermore, the lecture session on introduction to Linux was substituted with a comprehensive instructor-guided laboratory exercise (see Topic #1 in SOM [9] for exercise).

Table I
SUMMARY OF TIME (IN HOURS) INVESTED IN VARIOUS NSF/IEEE-TCPP CURRICULUM TOPICS. THE COLUMNS TITLED KNOW., COMP., APPL. CORRESPOND TO BLOOM'S TAXONOMY LEVELS OF KNOWLEDGE, COMPREHENSION, AND APPLICATION.

Topic	Know.	Comp.	Appl.
Language Extensions	0.25	-	-
Compiler directives	0.25	0.25	-
Libraries	0.1	0.5	-
Data Races	0.25	0.5	-
Synchronization	0.25	0.25	2
Critical regions	0.25	0.5	2
Producer-consumer	0.25	0.5	1
Monitors	0.25	0.25	1

Course materials are available as Supplementary Online Material (SOM) via http://www.users.muohio.edu/raodm/cse381_edupar13/.

The course commenced with rapid review of core concepts in C++ along with laboratory exercises providing applied skills required to develop programs under Linux using the GNU Compiler Collection (GCC) tool-set. Next the course covered Standard Template Library (STL) containers, primarily `std::vector` and `std::unordered_map`, iterators, and introduced STL algorithms (refer to Topic #2 in SOM [9]). The STL containers internally manage memory thereby initially insulating students from issues associated with dynamic memory management, which was taught later in the course. The students were motivated to work with STL algorithms and C++11 lambdas to streamline their programs and ease development of sophisticated software. Laboratory exercises and homework projects (see SOM [9]) were used to further students understanding and use of STL algorithms.

The STL algorithms do require a declarative-programming style thought process which the students found a bit more challenging than anticipated. Consequently, it was necessary to invest additional lecture time and laboratory exercise to ensure students were confortable with the thought process and strategies for problem solving with declarative programming. However, the learning curves were not too steep to distract the class from core concepts. Laboratory exercises, further explored the benefits of STL algorithms that perform automatic multithreading using OpenMP. The simplicity and significant performance improvements realized using multithreading excited the students and set the stage for introducing concurrency and parallelism concepts. However, in retrospect, reiteration of the benfits automatic multithreading seems to be needed to ensure students fully appreciate its advantages. Consequently, in the next offering of this course, a suitable homework exercise reemphasizing this topic will be introcued.

Having explored implicit multithreading, the course proceeded to cover explicit multithreading. The C++11 `std::thread` class makes creating and use of threads a straightforward task as it provides more convenient and intuitive wrappers around the underlying `pthreads` library [8]. Consequently, one lecture was sufficient to introduce the concept of multithreading, use of the `std::thread` class, and discuss basics of race conditions (see Topic #3 in SOM [9]). Having covered race conditions, the `std::mutex` and `std::lock_guard` classes in C++11 were used to develop critical sections. The `std::lock_guard` uses the conventional RAII approach to provide an exception safe mechanism for locking and unlocking the mutex [8].

Next, the concept of monitors were covered along with C++11's `std::condition_variable` class that provides a powerful yet intuitive implementation for monitors. The `std::condition_variable` also provides wait and notify methods that can be further streamlined using C++11 lambdas. Using these classes a complex producer-consumer type interaction is simplified to few lines of elegant C++11 code (refer to Topic #3 PowerPoint presentation in SOM [9]). Combined with STL algorithms the students were able to develop a sophisticated multithreaded file processing application (see Homework #9 in SOM [9] for details). Another homework project also required students to write an essay to compare and contrast the C++11 solution against the performance and elegance of a Java-based solution (see Homework #10 in SOM [9] for details).

The course also introduced students to new concurrency design patterns, namely: `std::future`, `std::async`, and `std::promise` [8]. These design patterns provide convenient API for utilizing concurrency of multi-core architectures by developing programs involving asynchronous methods in the background. These concepts were covered as part of lectures and empirically explored using laboratory exercises (see [9] for details).

IV. Results

Several direct formative assessments were conducted as an integral part of the course in the form of laboratory exercises and homework assignments. Pertinent laboratory exercises and homework are available as SOM via http://www.users.muohio.edu/raodm/cse381_edupar13/. The assessments indicated that the students were able to rapidly learn and apply new concurrency and parallelism concepts by developing complex applications. The students reported that most of the time invested in multithreading-related homework dealt with understanding, implementing, and troubleshooting concurrency issues in their programs. Furthermore, the students reported that they did not struggle with memory management and their negative perceptions about C++ where significantly changed by the course.

Laboratory experiments involving performance improvements and performance comparisons seemed to pique the students interest. For example, laboratory experiments exploring the effectiveness of using parallel version of standard algorithms elicited verbose commentary in lab reports from the students as they explored various configurations and varying number of threads (see Exercise #13 in SOM [9] for details). Similarly, experiments exploring the effect of compiler optimizations (using compiler flags `-O2` and `-O3`) also excited the students. These performance optimizations, particularly the ability to automatically multithread STL algorithms, seemed to swing the student's interest to focus on PDC concepts and questions about internals of how the libraries accomplished the task significantly increased. Furthermore, the compiler optimizations triggered active discussions on instruction set architectures, pipelining, Streaming SIMD Extensions (SSE), and superscalar operations on modern CPUs.

Indirect assessments to selected questions were obtained at the end of the course through free-form responses in the summative assessments. Specifically, student responses to the following question was analyzed: *"The STL algorithms such as `std::sort`, `std::find`, etc. provide a mechanism to automatically (via compiler flags) multi-thread calls to these methods. Briefly comment on whether if you think this approach was useful to learn to appreciate threading and effective use of multi-core CPUs? Note: This question is soliciting your honest opinion and therefore there is no correct or incorrect answer for this question."*

Analysis of students' responses to the above question indicated that a majority of students perceived automatic multithreading performed by the parallel implementation of standard C++ algorithms was useful and an effective way to learn about PDC concepts. Student feedback also indicated that it piqued their interest in developing parallel implementation of algorithms and thread management issues at the operating system (OS) level.

Additional indirect *anonymous* assessments were collected as part of the department's ABET accreditation process. The automated indirect assessment system compares pre- and post-course surveys of student perception of knowledge of course learning outcomes. Table II lists the pre- and post-course survey responses. These anonymous surveys are optional and the pre-course responses were obtained from 41 students while post-course surveys were completed by just 26 students. The numerical values in the range 1 through 5 correspond to the following set of choices — *1*: strongly disagree, *2*: disagree, *3*: indifferent, *4*: Agree, and *5*: strongly agree. Lower average values for pre-course outcomes survey indicates that students are not very confident about their understanding of the topic. Conversely, higher numbers recorded for post-course outcomes survey indicate that most students felt that they had gained a better understanding of the PDC concepts after completing the course.

The indirect assessments also confirmed that students gained a good understanding of PDC concepts and were

Table II
STUDENT FEEDBACK TO SELECTED SUBSET OF ANONYMOUS OUTCOMES SURVEY QUESTIONS PERTAINING TO PDC CONCEPTS. SET OF VALUES WITHIN { } ARE NUMBER OF STUDENTS SELECTING THE FOLLOWING CHOICES (IN THE ORDER): *5: strongly agree*, *4: agree*, *3: indifferent*, *2: disagree*, AND *1: strongly disagree*. NUMERICAL SCORE WAS COMPUTED AS WEIGHTED AVERAGE OF THE CHOICES.

Survey question	Pre-course Data	Post-course Data
I am able to understand fundamental issues in concurrency	2.15 {11, 18, 8, 3, 1}	4.15 {0, 1, 2, 15, 8}
I am able to demonstrate the potential run-time problems arising from the concurrent operation of many separate tasks	2.05 {0, 2, 9, 18, 11}	4.19 {9, 15, 1, 0, 1}
I am able to apply Concurrency and Parallelism in Practice	2.02 {0, 3, 9, 13, 14}	4.32 {9, 15, 1, 0, 0}
I am able to summarize the range of mechanisms that can be used to realize concurrent systems & describe their benefits	1.9 {0, 0, 8, 20, 12 }	4.27 {8, 17, 1, 0, 0}
I am able to Implement a deadlock-free multi-threaded program using suitable concurrent constructs in major languages	1.93 {1, 1, 8, 15, 17}	4.16 {8, 12, 4, 0, 1}
I am able to develop multi-threaded program using appropriate design patterns (such as: promise, future, etc.)	1.84 {0, 0, 7, 18, 13}	4.3 {11, 12, 3, 0, 0}
I am able to describe the use of High Level Language libraries for multi-threaded algorithms (such as: parallel sorting)	1.78 {0, 0, 8, 16, 17}	4.16 {7, 14, 3, 0, 0}

excited about the the style and elegance of the programming solutions explored in the course. Furthermore, routine University-administered *anonymous* course evaluations were also collected from students electronically. The course evaluations are voluntary and 92% of the students enrolled in the course completed the University's online course evaluations. Overall the evaluations for the course were very positive with an student rating of 3.38 (SD: 0.84) out of 4.0, which is well above the department average of 2.8. The students also highly rated the contribution of this course to their education and they reported that their appreciation and understanding of the topics were significantly increased.

V. CONCLUSION

The pedagogical experiences and various assessments indicate that the use of C++11 in teaching concepts of concurrency and parallelism had a very positive outcome. Consequently, the department is planning to pursue the same strategy in Fall 2013. We expect that the cumulative assessments will provide statistically significant quantitative data to back the qualitative feedback from students to permanently revise the syllabus of the course. Our current experiences indicate that C++11 is an excellent programming language to adopt in an systems course to introduce concurrency and parallelism while kindling and fostering excitement and interest about PDC concepts in the student community.

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*, ser. SIGCSE '11. New York, NY, USA: ACM, 2011, pp. 617–618. [Online]. Available: http://doi.acm.org/10.1145/1953163.1953336

[2] CSE Department, "Computer science and software engineering," 2013. [Online]. Available: http://www.eas.miamioh.edu/departments/cse/cse/

[3] CSE Department, "CSE 381: Operating systems (topics & outcomes)," 2012. [Online]. Available: http://www.eas.muohio.edu/?id=9233

[4] *ISO/IEC 14882:2011. Information technology – Programming languages – C++*, International Organization for Standardization, 1, ch. de la Voie-Creuse, CP 56, CH-1211 Geneva 20, Switzerland, Sep 2011. [Online]. Available: http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=50372

[5] CSE Department, "CSE 274: Data abstraction and data structures," 2012. [Online]. Available: http://eas.miamioh.edu/?id=9246

[6] CSE Department, "CSE 278: Computer architecture (topics & outcomes)," 2012. [Online]. Available: http://www.eas.muohio.edu/?id=9244

[7] A. Silberschatz, P. B. Galvin, and G. Gagne, *Operating System Concepts*. 440-4th Ave., New York 16, N.Y, USA: John Wiley & Sons, Jul. 2008.

[8] A. Williams, *C++ Concurrency in Action: Practical Multithreading*. 20 Baldwin Road, Shelter Island, NY 1196332, USA: Manning Publications, Feb. 2012. [Online]. Available: http://my.safaribooksonline.com/9781933988-771

[9] D. M. Rao, "Supplementary material for edupar-13," 2013. [Online]. Available: http://www.users.muohio.edu/raodm/cse381_edupar13/