DePauw University Proposal for Status as a Fall 2013 Early Adopter of the NSF/IEEE-TCPP Curriculum Initiative on Parallel and Distributed Computing

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

Introduction

We propose a multi-year effort to integrate topics from the NSF/IEEE-TCPP curriculum initiative on parallel and distributed computing (PDC) into multiple core and elective courses throughout the DePauw University computer science curriculum. The courses we plan on targeting first are: Computer Science I, Data Structures, Computer Systems, Parallel Programming and Advanced Algorithms, and Programming Languages. Additional courses will likely follow. This document will describe our context at DePauw, goals for participation, overall strategy for the work, early plans for particular courses, evaluation plans, biographies, and a requested budget.

Computer Science at DePauw

DePauw University is an undergraduate liberal arts institution, with seven tenured/tenure-track faculty members in the computer science department. The department graduates a few dozen majors each year, and also serves many non-majors. A wide range of electives is offered to complement the core course sequence in programming, data structures, algorithms, systems, object-oriented design, and foundations of computation. The faculty members are particularly active in educational research, with consistent efforts to create highly effective and relevant learning experiences for our students.

Goals for Participation as Early Adopters

An important continuation of these efforts includes the recognition of the increasingly strong role of PDC throughout all areas of computer science. The principle goal of our work as early adopters will be to prepare our students for the impending ordinariness of PDC. With even low-budget desktop computers attesting to the ubiquity of hardware support for parallelism, it is clear that PDC is too important to relegate to an upper-level elective or two. Rather, recognizing that "early and often" is the key to mastery, PDC concepts must be integrated *throughout* the curriculum in a level-appropriate manner, while taking care not to harm coverage of traditional course content. Our work as early adopters will be an important step in continuing to prepare our students for these developments in the discipline.

We have various subgoals related to the principle goal described above. As our students become more experienced in the many aspects of PDC, we expect to see more integration of this material arising in their own work in course and senior capstone projects. We intend to carefully study and document our successes and failures, sharing both with departmental colleagues for possible future PDC integration, and also with colleagues worldwide through presentations at conferences such as SIGCSE and the EduPar workshop at IPDPS. We hope in turn to learn from our colleagues at these events to inform our continued course refinements. Finally, we intend to be a part of the ongoing conversation about what a core PDC curriculum should be, along with best practices and pitfalls in its adoption.

Our Overall Integration Strategy

One significant challenge of including PDC material in existing courses is that existing courses are already very full. The question often arises, "What do I have to take out to fit this in?" For courses in which a significant PDC component is desired, it is true that some traditional material may need to be condensed or removed. However, we aim to mitigate and in some cases eliminate this need through the use of PDC as a *medium* for certain traditional course topics. Co-PI Steven Bogaerts has already had some success considering this approach explicitly, as described in [1].

One way to describe this approach is to make an analogy to the initial rise to prominence of object-oriented programming (OOP) years ago. Of course, OOP is a fundamentally different paradigm from imperative programming. We might imagine questions years ago about whether OOP should be relegated to an upper-level elective course while "traditional" courses continue in the imperative paradigm, or if, with greater effort, OOP should be somehow worked into several existing courses. We can see in history how these questions have been answered. Introductory courses may have lost some aspects of imperative programming, but many of the same concepts are still taught, only in the *medium* of OOP. In upper level courses we are not at all surprised to see that OOP is now essential for effective implementation of many course topics (whether in graphics, artificial intelligence, etc.). Classes and objects are just ordinary tools we use throughout much of computer science.

Similarly, we argue that this is the direction in which PDC is moving. For the most part, courses will still cover traditional topics. It is simply that some of those topics will be considered in the medium of parallelism rather than the currently more-familiar medium of sequentialism. Thus we aim to keep this big picture strategy in mind throughout our work: Do what you usually do, just do (some of) it in parallel. Our discussion of specific courses below will outline some preliminary plans to follow this strategy.

Courses Identified for First Efforts

CSC 121 – Computer Science I

This is the introductory computer science course, taken both by majors and non-majors. It is designed to offer not just narrow programming knowledge, but broad exposure to the scope of computer science. Thus this course provides an excellent context in which to consider big picture ideas in PDC apart from their particular implementation in a language. Coverage may include PDC curriculum areas such as Architecture: Taxonomy (C), Architecture: Multicore (K), Programming: Shared Memory (C), Programming: Distributed Memory (C), Programming: Synchronization (C), and Algorithms: Speedup (K).

The programming component of the course is in Java, and there are many opportunities for parallelism in Java that we look forward to exploring for potential integration in a level-appropriate manner. Specifically, the Fork/Join framework, threads and thread pools, and in fact the entire java.util.concurrent package provide many opportunities to consider various PDC curriculum areas, including Programming: Shared Memory (A), Programming: Task/Thread Spawning (A), and Programming: Synchronization (A) to name just a few.

CSC 122 – Data Structures

This second course in the introductory sequence considers common data structures and associated algorithms, with programming work done in C++. The study of data structures and algorithms goes hand-in-hand with discussions of efficiency, and efficiency today goes hand-in-hand with parallelism. We expect that an introduction to some facilities in OpenMP and perhaps MPI would provide an effective medium for exploring some traditional data structures topics, with coverage of areas including

Programming: Shared Memory (A) (including compiler directives and libraries), Programming: Distributed Memory (A) (including message passing), Programming: Data Parallel (A) (including loops and distributed memory), and Programming: Synchronization (A) (and sub-topics). Additional algorithmic coverage would include Algorithms: Speedup (K), Algorithmic Paradigms: Divide & Conquer (C), and Algorithmic Paradigms: Recursion (C).

CSC 231 – Computer Systems

Students enroll in Computer Systems (cSys) after taking Data Structures, as one of three required intermediate core courses (including Software Development and Foundations of Computing). cSys begins at the point of students' familiarity, the high-level language model (HLL), and traverses the models of computation (assembly language, machine language and digital design) from more abstract models to more detailed models. Altering the HLL coverage by illustrating how HLL can utilize parallel architectures allows the department to update its curriculum without sacrificing its "models of computation" theme and also reinforces parallel programming concepts learned in Data Structures and Computer Science I.

Parallel architecture concepts offered in cSys may include: Pipelines (A), Streams (A), Message Passing (A) and Benchmarks (A). In addition, the current unit on data representation can be expanded to include the effect of parallelism on range and precision (A).

CSC 396 – Parallel Programming and Advanced Algorithms

This is an upper-level course designed to supplement our coverage of algorithms as well as include parallel programming explicitly in the curriculum. This course uses the programming language Erlang, and includes coverage of advanced algorithms, focusing particularly on running time efficiency. With the use of Erlang, coverage of the topics of Architecture: Message Passing (A), Programming: Distributed Memory (A), and Programming: Functional/Logic Languages (A) are givens. Additionally, we will focus heavily on the entire Algorithm group of topics, particularly Asymptotics, Time, Speedup, Space Compression, and Time vs. Space. Algorithmic Paradigms will also be heavily covered, focusing specifically on Divide & Conquer, Recursion, Reduction, and Graph Algorithms. We hope to cover parallel programming in other languages (particularly Java and C++), but this coverage would be fairly brief; it would, however, at least offer some exposure to the concepts of shared memory, synchronization, deadlock, and dependencies.

CSC 424 – Programming Languages

This course focuses on the range of programming paradigms and the fundamental techniques used to express them in language design and implementation. It builds on students' previous experiences with object-oriented programming in Java and C++, as well as some functional language experience with Scala in CSC 233, Foundations of Computation. It currently contains a short unit on language support for PDC, with some coverage of the following topics: Shared Memory (C), Language Extensions (K), Tasks and Threads (K), Message Passing (C), and Functional/Logic Languages (A). Students have been exposed to models such as Map-Reduce, transactional memory, and the actor model. With additional background in prerequisite core courses as described above, this coverage could be extended to include Data Parallel (A) programming and Parallel Loops (A), more material on Memory Models (C), as well as additional experience with common Libraries (A).

Additional Courses

We expect the courses listed above to be just the beginning. In time we hope to integrate concepts of PDC into additional courses to the extent that this is topical.

Evaluation Plan

We intend to evaluate our work in several ways. At the end of each revised course, we will statistically analyze student surveys of interest and understanding of PDC concepts. Particularly in introductory courses, these surveys can also ask about interest in learning additional PDC concepts in later courses. We will also analyze grades on PDC assignments and test questions to gauge student understanding. Finally, we will evaluate our work from our own perspectives, considering the successes and failures of our efforts, obtaining feedback from colleagues, and considering the degree to which our work is adopted and refined further. This will be done both within our department and externally through our contributions to the CDER courseware website.

Biographies

Steven Bogaerts is new to the DePauw faculty. After much work in early adoption efforts at Wittenberg University (e.g. [2,3,4]), he is excited to explore early adopter efforts in a new context and continue to contribute to the international discussion of this topic. Steve has a wide range of interests, including parallel computing, artificial intelligence, and cybersecurity, and is particularly interested in work that combines these areas.

Brian Howard earned a B.S. degree from Northwestern University and a Ph.D. from Stanford, both in Computer Science. Before coming to DePauw, he held positions at the University of Pennsylvania, Kansas State University, and Bridgewater College. He has taught a wide range of CS courses, with a focus on programming languages, mathematical foundations, and databases; his research centers on ways to make higher-level languages more accessible to students.

Scott Thede received a B.S. in Electrical Engineering from the University of Toledo, and an M.S. in Electrical Engineering and a Ph.D. in Computer and Electrical Engineering from Purdue University. He teaches a wide range of computer science courses, and has particular research and teaching interests in artificial intelligence and parallel computing.

Gloria Townsend began teaching most of the DePauw University courses listed in this proposal thirty-four years ago. Her research and teaching interests lie in evolutionary computation, robotics, computer systems and gender issues of computing.

Budget Requested

Given that three professors are planning on work in multiple courses for years to come, we are requesting the full $2,500.00 to support this long-term effort.

References

[1] Bogaerts, S., Burke, K., Shelburne, B., and Stahlberg, E. Concurrency and Parallelism as a Medium for Computer Science Concepts. Curricula for Concurrency and Parallelism workshop at Systems, Programming, Languages, and Applications: Software for Humanity 2010 (SPLASH-2010), Reno, NV, October 2010.

[2] Bogaerts, S. Hands-On Exploration of Parallelism for Absolute Beginners with Scratch. Third NSF/TCPP Workshop on Parallel and Distributed Computing Education (EduPar-13), Boston, MA, May 2013.

[3] Bogaerts, S., Burke, K., and Stahlberg, E. Experiences in Parallel and Distributed Computing Education. Third NSF/TCPP Workshop on Parallel and Distributed Computing Education (EduPar-13), Boston, MA, May 2013.

[4] Bogaerts, S. and Stough, J. Python for Parallelism in Introductory Computer Science Education. Educators Program, Supercomputing 2012: The International Conference for High Performance Computing, Networking, Storage, and Analysis, Salt Lake City, UT, November 2012.