Teaching Parallel Design Patterns to Undergraduates in Computer Science

Richard A. Brown (Moderator)
St. Olaf College
rab@stolaf.edu

Joel C. Adams
Calvin College
adams@calvin.edu

Clayton Ferner
UNC Wilmington
cferner@uncw.edu

Elizabeth Shoop
Macalester College
shoop@macalester.edu

Barry Wilkinson
UNC Charlotte
abw@uncc.edu

SUMMARY

The industry shift emerging forms of parallel and distributed computing (PDC), including multi-core CPUs, cloud computing, and general-purpose use of GPUs, have naturally led to increased presence of PDC elements undergraduate Computer Science curriculum recommendations, such as the new and substantial "PD" knowledge area in the joint ACM/IEEE CS2013 recommendations[1]. How can undergraduate students grasp the extensive and complex range of PDC principles and practices, and apply that knowledge in problem solving, while PDC technologies continue to evolve rapidly?

Parallel design patterns are descriptions of effective solutions to recurring parallel programming problems in particular contexts and have emerged from long-standing industry practice. Parallel patterns occur at all computational levels, ranging from low-level concurrent execution patterns (such as *message passing* or *thread pool* patterns) to high-level software design patterns suitable for organizing entire systems or their components (such as *model-view-control* or *pipe and filter* patterns). The sheer number of parallel patterns, which reflect the full breadth and complexity of PDC, can be quite daunting for a newcomer. However, the ubiquity of parallel patterns in all forms of parallel and distributed computation makes these patterns relevant and illuminating at all undergraduate levels. Knowledge of parallel patterns, being reusable elements of parallel design, guides problem-solving during the creation of parallel programs; and those enduring design patterns remain relevant and useful as new PDC infrastructure emerges in this rapidly evolving field.

This panel presents four viewpoints representing various approaches for teaching parallel patterns to CS undergraduates at multiple academic levels. Moderator Dick Brown co-directs (with Adams and Shoop) the CSinParallel project (csinparallel.org), which produces and shares modular materials for incrementally adding parallelism to existing undergraduate computer science courses [2]. A former director of CS at St. Olaf, he serves as an officer and executive board member of EAPF (eapf.org). Brown will briefly review the goals of the session (3-4 min), introduce each panelist (10 min each), and moderate discussion.

SIGCSE'14, March 5–8, 2014, Atlanta, Georgia, USA.
ACM 978-1-4503-2605-6/14/03.
http://dx.doi.org/10.1145/2538862.2538875

Categories and Subject Descriptors

K.3.2 [**Computers and Education**]: Computer and Information Science Education – *Curriculum.*
D.1.3 [**Programming Techniques**]: Concurrent Programming – *Distributed programming, Parallel programming.*
D.2.11 [**Software Engineering**]: Software Architectures – *Patterns.*
D.1.3 [**Programming Languages**]: Language Constructs and Features – *Patterns.*

General Terms: Algorithms, Design, Performance.

Keywords

Parallel computing, parallelism, distributed computing, education, parallel design patterns, curriculum, design patterns, Seeds, Paraguin, patternlets, exemplars.

1. JOEL ADAMS

Joel Adams is Professor and Chair of the Department of Computer Science at Calvin College, and has been teaching his students about concurrency and parallelism for more than 20 years. He has been the principle architect of four Beowulf clusters, is a two-time Fulbright scholar, and is an ACM Distinguished Educator.

Statement: A *patternlet* is an executable program that shows the behavior of a parallel pattern, that is:

- *Minimalist* in terms of eliminating non-essential features, so that students can easily grasp the program's essential concept.
- *Scalable* so that students can vary the number of threads and observe how the program's behavior changes.
- *Syntactically correct* so that students can use the program as a working model for their own coding.

Each patternlet provides a simple way to introduce students to a particular pattern. Patternlets are designed to be a flexible pedagogical tool to help students master a particular parallel concept in any appropriate course, whether as a hands-on activity by students or as a live-coding demonstration by instructors.

Patternlets are an evolving collection; there are at this time 25 of them – 14 in OpenMP and 11 in MPI – illustrating the *single program multiple data (*SPMD), *fork-join*, *master-worker*, *parallel for loop*, *barrier*, *message passing*, *mutual exclusion*, *scatter, gather*, *broadcast, reduction*, *data decomposition*, and other parallel patterns. Adams will present several patternlets

and discuss different ways they can be used to introduce parallelism into the CS curriculum.

2. ELIZABETH SHOOP

Libby Shoop co-directs (with moderator Brown and co-panelist Adams) the CSinParallel project (csinparallel.org) [2]. She teaches several courses in the CS curriculum, ranging from the introductory level to computer systems organization, software development, and parallel and distributed systems.

Statement: The parallel and distributed computing (PDC) curricular recommendations a decade from now may differ as radically from today's curricular recommendations as the PDC aspects of CS2013 differ from those of CS2001. However, PDC in some form is here to stay in the CS curriculum. As educators we must ask ourselves two key questions: 1) Apart from parallel algorithms, *are there principles that we can teach our students in the context of today's PDC hardware and software milieu that will remain relevant and abiding for the unforeseeable technologies of the next decade?* 2) Since all computing students must now learn about PDC and some may not be interested in PDC, *how can we motivate today's computing students to engage with this rapidly evolving field*?

We propose a two-pronged approach to addressing these issues. Our first "prong" is *parallel programming patterns*, which emerge directly from the experience of professional practitioners. These offer a practical and relevant collection of problem-solving strategies that transcend particular technologies, enabling students to acquire enduring intellectual and useful skills in PDC. Our second "prong" is *exemplar applications* that apply PDC topics and techniques in solving some (more or less) realistic problem. Such exemplars provide convenient and useful resources for presenting and comparing parallel and distributed computing technologies. They also motivate students to learn PDC principles and practices by helping them see how PDC makes an impact in other fields. The combination of patterns and exemplars provides a unified approach for motivating undergraduates to develop parallel thinking, often with modest or even negligible added costs to existing course syllabi.

3. BARRY WILKINSON

Barry Wilkinson is Professor of Department of Computer Science at University of North Carolina, Charlotte and has been involved in parallel computing since the 1970's. He was a *Supercomputing 2011* Undergraduate Engineering and Sciences award finalist in 2011. He has published six textbooks, four with second editions, including *Programming: Techniques and Application Using Networked Workstations and Parallel Computers*, with M. Allen, 1999, 2nd ed. 2005, Prentice Hall.

Statement: A pattern programming framework called Seeds has been developed at UNC-Charlotte that enables programmers to select a parallel pattern such as *workpool* and create parallel executable code without writing any low-level message passing code (MPI) [3]. The framework self-deploys on any parallel or distributed computing platform including a single multicore computer. This framework is currently being used in a senior undergraduate parallel programming class taught to six universities across North Carolina on the NCREN televideo network. We believe pattern programming offers a very promising approach to teaching parallel programming. We are strong advocates for using patterns as they provide a guide to best practices and provide scalable design structures. Parallel programming is much easier for students, less likely to be flawed, more scalable, and it allows others to understand the code better. One can concentrate upon issues such as data partitioning and performance. Wilkinson will present the Seeds framework and its use in the undergraduate CS curriculum.

4. CLAYTON FERNER

Clayton Ferner is a professor of Computer Science at the University of North Carolina Wilmington (UNCW). He has been a faculty member at UNCW for 14 years and taught courses on parallel computing, grid computing, programming languages, compiler construction, operating systems, and data structures. He has developed a parallelizing compiler that generates MPI code suitable to run on a distributed-memory parallel system.

Statement: The *Paraguin* compiler, being developed at UNC-Wilmington, is a source-to-source compiler that will translate programs written in a style similar to OpenMP to a program that runs in parallel on a distributed-memory system using MPI to handle communication. Like OpenMP, the programmer directs the parallelization through the use of `pragma` statements to create the MPI code. The level of abstraction is much higher using the Paraguin compiler than with MPI. Because it is a source-to-source compiler, the user is free to inspect the result to see how their algorithm was implemented as well as modify it.

Since OpenMP uses `pragmas` to create parallel code for execution on shared-memory systems, our compiler can easily create a hybrid program by passing the OpenMP `pragmas` through to the MPI output code. The resulting output code is a hybrid parallel program that uses MPI to create parallelization for distributed-memory systems while simultaneously using OpenMP to create parallelization between cores of processors.

The compiler also supports a few patterns through the use of `pragmas`. The *scatter/gather* and *stencil* patterns have been implemented with more patterns planned in the next year. Students can create correct parallel programs implemented in MPI using these patterns while avoiding issues such as deadlocks and race conditions.

ACKNOWLEDGMENTS

Adams, Brown, and Shoop are supported by NSF DUE-1225739/1225796/1226172. Ferner and Wilkinson are supported by NSF DUE-1141005/1141006.

REFERENCES

[1] ACM/IEEE-CS Joint Task Force, "Computer Science Curricula 2013." [Online]. [Accessed: 07-Sep-2013]: http://ai.stanford.edu/users/sahami/CS2013/.

[2] R. Brown and E. Shoop, "Modules in community: injecting more parallelism into computer science curricula," in *Proceedings of the 42nd ACM technical symposium on Computer science education*, New York, NY, USA, 2011, pp. 447–452.

[3] B. Wilkinson, J. Villalobos, and C. Ferner, "Pattern programming approach for teaching parallel and distributed computing," in *Proceeding of the 44th ACM technical symposium on Computer science education*, New York, NY, USA, 2013, pp. 409–414.