A Stratified View of Programming Language Parallelism for Undergraduate CS Education

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

Joel C. Adams
Calvin College
adams@calvin.edu

David P. Bunde
Knox College
bunde@knox.edu

Jens Mache
Lewis and Clark College
jmache@lclark.edu

Elizabeth Shoop
Macalester College
shoop@macalester.edu

SUMMARY

It is no longer news that undergraduates in computer science need to learn more about parallelism. CS graduates who enter the computing workforce without substantial knowledge of parallel computing do so at their peril, since all new computers feature multi-core processors, with the number of cores expected to increase exponentially over time [3]. Also, our work and home lives have grown dependent on web services fueled by distributed parallel computing on cloud platforms.

Students learn parallel computing well through hands-on exercises and projects using programming languages with support for parallelism [4]. The range of options for parallel programming is truly staggering, involving hundreds of languages. How can a CS instructor make informed choices among all the options?

This panel provides a guided introduction to parallelism in programming languages and their potential for undergraduate CS education, organized into four progressive categories:

- low-level libraries and features that are close to the hardware and operating system;
- higher-level libraries and features, providing a layer of abstraction or management;
- programming languages that incorporate parallelism; and
- frameworks for productive parallel programming.

The four panelists, who direct NSF-funded projects on languages and instructional materials for teaching parallelism and/or led a recent international study on adding parallelism to undergraduate CS curricula [4], will present representative examples in their categories, then present viewpoints on how those categories relate to coursework, curriculum, and trends in parallelism.

A wiki accompanies this panel, with references on relevant language resources and curricular materials (hopper.macalester.edu/groups/sigcse2012languageparallelism/).

Categories and Subject Descriptors

D.3.2 [**Programming Languages**]: Language classifications – *Concurrent, distributed, and parallel languages.*

General Terms

Languages, Design, Performance.

Keywords

Programming languages, parallel computing, parallelism, education.

1. RICHARD BROWN (MODERATOR)

Dick Brown co-directs (with panelist Shoop) the CSinParallel project (csinparallel.org) for producing and sharing modular materials for incrementally adding parallelism to existing undergraduate computer science courses [6]. He has directed the creation of parallel platform resources, including the MistRider virtual cluster [5] and the WebMapReduce interface to Hadoop map-reduce computing [9]. A longtime director of CS at St. Olaf, he serves as an executive board member of the EAPF (eapf.org).

Brown will briefly review the goals of the session (3-4 min), introduce each panelist (10 min each), and moderate discussion.

2. JENS MACHE

Jens Mache is Professor of Computer Science at Lewis & Clark College. He has taught parallelism to undergraduates since 2001. He and panelist Bunde direct an NSF TUES project about teaching parallel computing with activity-based laboratories.

Statement: One way to introduce concurrency is through "low-level" multi-threaded programming (in C/C++, Java, or Python). These programs include explicit instructions for parallelism: each core executes an explicitly assigned task, all cores synchronize, they execute another task, and so on. Even though threads are unlikely to work in most CS 1 courses, they fit well into OS, networking, and system programming types of courses. Alternatively, CUDA and OpenCL enable the use of graphics processing units (GPUs) for general purpose parallel computing. Unlike current CPUs, many GPUs already have hundreds of cores. All of these approaches give students the ability to map tasks to threads as well as more explicitly manage memory, potentially allowing high performance and helping students learn about the hardware.

I will present an overview, as well as selected examples from teaching materials.

3. 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 20 years. To provide platforms for his students to experience the benefits of distributed parallelism, he has designed and built a variety of Beowulf clusters, ranging from the personal cluster *Microwulf* [1] to the terascale interdisciplinary science cluster *Dahl* [2].

Statement: A relatively easy way to introduce parallelism into the computer science curriculum is through high-level libraries that add parallel capabilities to a sequential language. Such libraries

provide abstraction mechanisms that simplify process/thread creation, communication, management, synchronization, and destruction, freeing a student (and software engineer) to focus on parallel algorithm development, debugging, and tuning. Some make it quite easy to add parallelism to legacy applications. Many of these libraries can also be used with different languages (e.g., C, C++, Fortran, Python), allowing a typical software engineer to incorporate parallelism into her projects without having to learn a new language.

I will present an overview of libraries that support shared-memory parallelism and distributed-memory parallelism; and will present selected examples from teaching materials as illustrations.

4. DAVID BUNDE

David Bunde is Assistant Professor of Computer Science at Knox College. He has taught concurrency and parallelism in a variety of courses. He and panelist Mache have written about the need for higher-level approaches in parallel education [7] and now direct an NSF TUES project developing this approach.

Statement: One can also teach parallelism and concurrency by using languages with high-level constructs to manage them. As with a library-based approach, these languages seek to let the programmer focus on algorithms, with some low-level details being handled by the compiler or run-time system. In fact, many parallel languages are close relatives of traditional sequential languages to simplify adoption. The language-based and library-based approaches differ in that a parallel language can add new keywords and constructs rather than needing to fit its form entirely within the structure of the base language.

There is a tremendous variety of parallel programming languages to select from. Many are created by extending a base language with mechanisms for lightweight task creation; Charm++ and Cilk extend C++, Habanero Java extends Java, and Scala is a Java-like but functionally-flavored language. Other functional language options are Erlang, Haskell, and occam. Some languages rely more heavily on the compiler by including high-level operations that it parallelizes; examples include loops and reductions in Chapel and the use of math symbols in Fortress.

I will present a brief overview of these options, exploring a couple in greater depth and describing our experiences at Knox. With so many language choices, the goal is to be a "commercial" so attendees become aware of interesting options, with more detailed information available on the wiki.

5. ELIZABETH SHOOP

Libby Shoop co-directs (with moderator Brown) the CSinParallel project (csinparallel.org) [6]. She teaches several courses in the CS curriculum, ranging from the introductory level to computer systems organization, software development, and parallel and distributed systems. She is developing a parallel computing course structured according to OPL patterns (parlab.eecs.berkeley.edu/wiki/patterns/patterns).

Statement: The technologies of data-intensive scalable computing on cloud platforms provide an ever-expanding array of web services that have become staples in the computing consumer market. Scalable map-reduce frameworks as described in [8] lie behind most of the web services we have come to expect. The open-source Hadoop framework (hadoop.apache.org), Yahoo-initiated software that makes it possible for tiny startups with good ideas to launch new web services on rented cloud computing resources, is equally available as a teaching platform for undergraduate CS students. We have been teaching these technologies to students as early as the introductory course [9].

Map-reduce is not the only pattern of parallel computing that might be encapsulated in a convenient framework, although it is the first to have such great success. I will present several other patterns that might be encapsulated in frameworks, and ask the community to consider whether interfaces to such frameworks might become available for languages of the future, or even be integrated into them.

6. ACKNOWLEDGMENTS

Brown and Shoop are supported by NSF DUE-0941962/0942190. Bunde and Mache are supported by NSF DUE-1044299/1044932.

7. REFERENCES

[1] J. Adams, T. Brom. Microwulf: A Beowulf Cluster For Every Desk. *Proc. 39th ACM Technical Symp. Computer Science Education* (SIGCSE 2008), pp. 121-125.

[2] J. Adams, K. Hoogeboom, and J. Walz, A Cluster for CS Education in the Multicore Era, *Proc. 42nd ACM Technical Symp. Computer Science Education* (SIGCSE 2011), pp. 27-31.

[3] Asanovic, K., Bodik, R. et al. A view of the parallel computing landscape. *CACM*. 52, 10 (2009), pp. 56-67.

[4] Brown, R., Shoop, E., Adams, J., et al. Strategies for Preparing Computer Science Students for the Multicore World. *Proc. 15th Annual Conf. Innovation and Technology in Computer Science Education (ITiCSE 2010) Working Group Reports*, pp. 97-115.

[5] Brown, R.A. Hadoop at home: large-scale computing at a small college. *Proc. 40th ACM Technical Symp. Computer Science Education* (SIGCSE 2009), pp. 106-110.

[6] Brown, R., Shoop, E. Modules in community: Injecting more parallelism into computer science curricula. *Proc. 42nd ACM Technical Symp. Computer Science Education* (SIGCSE 2011), pp. 447-452.

[7] D.P. Bunde and J. Mache. Teaching concurrency beyond HPC. Position paper presented at *1st Workshop on Curricula in Concurrency and Parallelism*, OOPSLA, 2009.

[8] Dean, J. and Ghemawat, S. MapReduce: Simplified Data Processing on Large Clusters. *Operating System Design and Implementation* (OSDI 2004), pp. 137-150.

[9] Garrity, P., Yates, T., Brown, R., Shoop, E. WebMapReduce: An accessible and adaptable tool for teaching map-reduce computing. *Proc. 42nd ACM Technical Symp. Computer Science Education* (SIGCSE 2011), pp. 183-188.