Preparing Students for Ubiquitous Parallelism

Daniel Ernst (Moderator)
Department of Computer Science
University of Wisconsin – Eau Claire
Eau Claire, WI, USA 54702
ernstdj@uwec.edu

Barry Wittman
Department of Computer Science
Purdue University
West Lafayette, IN 47907
bwittman@purdue.edu

Brian Harvey
Division of Computer Science
University of California, Berkeley
Berkeley, CA 94720
bh@cs.berkeley.edu

Tom Murphy
Department of Computer Science
Contra Costa College
San Pablo, CA 94806
tmurphy@contracosta.edu

Michael Wrinn
Intel Corporation
Hillsboro, OR 97124
Michael.Wrinn@intel.com

SUMMARY

Current trends in microprocessor design are fundamentally changing the way that performance is extracted from computer systems. The previous programming model for sequential uniprocessor execution is being replaced quickly with a need to write software for tightly-coupled shared memory multiprocessor systems. Both academicians and business leaders have challenged programmers to update their skill sets and their tools to effectively tackle software development for these newer platforms [1].

In addition, the availability of cheap processor cycles has created an explosion in capability for large data set computation. Applications in this category require programmers to strongly grasp the concepts of scaling parallelism.

This new tack in computing towards ubiquitous parallelism raises several questions for computer science educators: How do we best prepare students for a world where parallel resources are always available? What role should parallel programming play in our curricula? What parallel programming concepts and skills are most important? What are the best ways in which to get students to learn them? Which are the most difficult for students to grasp?

In this panel, we will discuss our experiences engaging with students, teachers, and researchers on the topic of parallel programming. The participants come from diverse backgrounds and have varied approaches to this issue. We hope that educators attending this panel will gain insights into this new direction in general-purpose computing, while taking home some concrete frameworks they can use at their own institutions.

Categories and Subject Descriptors

K.3.2 [**Computer and Information Science Education**]: Computer Science Education, Curriculum

SIGCSE'09, March 3–7, 2009, Chattanooga, Tennessee, USA.
ACM 978-1-60558-183-5/09/03.

General Terms

Design, Standardization

Keywords

parallel programming, curriculum design

Daniel Ernst

As moderator, I will introduce the other panelists and briefly discuss the steps we have taken at UW – Eau Claire to integrate parallelism throughout our curriculum [2].

Barry Wittman

Prompted by the changing landscape and by active encouragement from our corporate partners, the Purdue University Department of Computer Sciences has created a Multicore Initiative to substantially increase the multicore content in our curriculum.

The most innovative element of the Multicore Initiative is a course for first or second semester freshmen designed to teach the fundamentals of programming with concurrency integrated into as many topics as possible. The course fulfills our CS1 requirement but ranges a little beyond a typical CS1 course in difficulty. Students are eligible to enroll with either a strong high school background in programming or successful completion of our CS0 course. Our course is taught in Java, and all concurrency is based on threading.

We have faced challenges in fitting both the traditional CS1 material and concurrent topics into our course. Because some programming experience is a prerequisite, we have accelerated the first month of Java fundamentals and left out some of the more obscure features of the language. Having no textbook which unites introductory programming with concurrency, we were forced to write our own. We are excited to share our experiences of wrestling with the design of this course for over a year and the results of teaching the course for the first time in fall 2008.

Brian Harvey

As part of a new emphasis on parallelism at all student levels, we have developed curriculum units for each of the three courses in our lower division sequence; the first two focus on the MapReduce approach. Our first course uses Scheme, a language whose emphasis on functional programming techniques fits well with MapReduce, whose name suggests a composition of two higher order functions.

While the MapReduce design (we use the open-source Hadoop implementation) is not, in fact, functional – the underlying elegant ideas are obscured by sequential use of iterators – we have created a Scheme interface to Hadoop that hides the messy details. An invocation of

```
(mapreduce <mapper> <reducer> <base-case> <datafile>)
```

is almost exactly equivalent to the familiar

```
(reduce <reducer> <base-case> (map <mapper> <data>))
```

The one complication visible to students is the use of key-value pairs in the data, so that results from the map phase can be distributed among the processors for the reduce phase based on their keys.

Using functional programming as the setting for parallelism eliminates the need for students to worry about concurrency control; in this first introduction to the subject, there are no semaphores or other locking constructs, and a substantial source of bugs is eliminated.

Tom Murphy

I am a member of a three person team of Computer Science professors spanning geography and college levels (community college, liberal arts, and PhD granting) focused on effecting change in the Computer Science curriculum though hands-on week long parallel and distributed workshops held for faculty each summer. We reach more students by working with the faculty that teach those students.

Computational science looks to be at the center of science for this millennium, where a specific scientific discipline defines the problem to be modeled, mathematics verifies that the model is correct, and computer science helps to efficiently generate, and make sense of, the mountains of resulting data. Many-core clusters are our computational future, which means the various faces of parallelism and concurrency should be part of the future for computer science education [3].

We have taught data parallelism through use of GPUs, memory parallelism though use of SMPs and multi-core, and message parallelism through use of clusters. To convey parallelism we have used rope tricks, line dances, and many analogies. A large part of our efforts have been in simplifying the use of MPI.

We have also surrounded ourselves with a similarly eclectic graphically dispersed team of student and former student apprentices to help us in projects ranging from hardware to software engineering, as well as helping instruct in workshops, which fosters their education in ways not typically available to students (and former students).

Michael Wrinn

With the migration to multi-core processors well underway, the academic and commercial communities are becoming alert to the need for a corresponding shift in the approach to software: parallelism is essentially everywhere, in the hardware; how ought parallelism influence the teachings in universities?

Curricular timeframes make this particularly challenging: given the latencies in developing new material, and the subsequent time before those enlightened students graduate and enter the work force, we are trying to anticipate the state of parallel systems years into the future. Current trends point toward a much larger number of cores, with a mix of specialized processing units, and non-uniform memory access – indeed, such platforms exist today. What is the best foundation for those in computational professions to work well with, say, heterogeneous manycore systems?

I am part of a team at Intel chartered to respond to that question. Sitting at the intersection of internal development, university research, and university education, we create and share (for free) training material based on connections found among those domains; faculty may use this content directly in their courseware. We continue to explore, with the academic community, ways to address the broad challenge of "thinking parallel".

REFERENCES

[1] Krste Asanovic et al. *The Landscape of Parallel Computing Research: A View from Berkeley*. Electrical Engineering and Computer Sciences, University of California, Berkeley. Technical Report No. UCB/EECS-2006-183, Dec. 18, 2006; www.eecs.berkeley.edu/Pubs/TechRpts/2006/EECS-2006-183.html

[2] Daniel J. Ernst and Daniel E. Stevenson. "Concurrent CS: Preparing Students for a Multicore World". Appears in the *13th Annual Conference on Innovation and Technology in Computer Science (ITiCSE '08)*, 2008.

[3] Scott Lathrop and Thomas Murphy, "High-Performance Computing Education," *Computing in Science & Engineering*, vol. 10, no. 5, Sept/Oct 2008, pp. 9-11.